On-Demand Service Composition Among Autonomous Self-interested Software Agents in Open Environments

Ingo Müller

PhD
2009
Abstract

Service-oriented computing (SOC) has evolved in recent years as a major paradigm for programming distributed software systems. SOC proposes a unified standards-based approach to leverage new levels of interoperability and automation to the implementation of business processes across heterogeneous applications and across organisational boundaries. The core principle of SOC is to enable business process automation through service composition, that is by combining several existing services into a new composite service. Although the concept of service composition is a core principle of SOC, it still requires more research to instantiate it. Prevailing service composition solutions in industry are workflow-driven. They are focused on ensuring full control over business processes at predictable costs. These benefits are, however, achieved at the expense of a major shortcoming: workflow-driven approaches are of rigid hard-coded nature. They typically utilise design-time service composition only, making resulting implementations vulnerable to unforeseen dynamic changes in the environment or to the business processes themselves.

While workflow-driven approaches are suitable for application within a single organisation, more agile and flexible solutions are necessary for facilitating inter-organisational service composition, especially among autonomous peers with, in general, variable relationships and dependencies. These peers may experience volatile network conditions or leave and enter environments at will. Moreover, peers typically do not disclose their potentially conflicting private information and perform autonomous decentralised decision-making while competing for limited resources or collaborating to gain profits. The resulting complex environments spanned by such peers are known as open environments. They exhibit the characteristics of partial observability, non-determinism and dynamics, which together imply that no single peer can impose full control nor obtain complete knowledge at any time during the service composition process.
The central problem addressed in this thesis is the dynamic composition of services in open environments which may evolve in a number of application scenarios, e.g. in volatile ad hoc and peer-to-peer networks or in dynamic on-demand e-business settings.

Given its characteristics, the problem of dynamic service composition has been intensively studied by the software agent community because the concept of software agents provides a natural metaphor for modelling the aforementioned peers. Software agents are autonomous goal-oriented problem-solvers that interact based on decentralised reasoning and decision-making. A system of multiple software agents matches the characteristics of open environments due to agent autonomy and decentralisation and thus facilitates the modelling of dynamic service composition in open environments as agent-based collaborative problem-solving. This hypothesis forms the foundation of this thesis.

A review of related agent-based approaches, however, reveals limitations in their applicability in open environments that usually result in closed systems despite the intrinsic similarities stressed above. The apparent gap stems from the absence of a universal software agent definition in general and the inconsistent use of a particular definition for modelling the fundamental building blocks of the software agent paradigm. Both problems are well known and have been repeatedly argued in related literature as major obstacles to a large-scale adoption of agent technology in industry.

The aim of this thesis is to provide an agent-based approach to the problem of dynamic service composition in open environments according to a uniform software-engineering-like definition and use of the fundamental building blocks of the software agent paradigm. The contributions of this thesis are:

- A detailed analysis of a concrete interpretation of the weak notion of agency and the subsequent derivation of a software agent component type that lays the foundation of a uniform agent-based software engineering approach. The resulting fundamental building blocks establish multiagent systems as open systems that exhibit the characteristics of open environments. They also utilise service-oriented design principles in support of a successful mapping of the problem of service composition in open environments into the software agent domain.
• The concept and design of a multiagent coalition formation framework models on-demand service composition in open environments as collaborative problem-solving in an open market. In this open market, supply and demand confront each other as equal-righted autonomous peers in the form of service consumer agents and service provider agents. The design of the respective multiagent system is influenced by a number of concepts such as decentralisation, on-demand service provision, self-organisation, emergence and market-based control.

• The implementation of the multiagent coalition formation framework facilitates coalition formation among service provider agents with lightweight interaction protocols and decentralised multi-attribute utility-based decision-making based on client-defined quality-of-service (QoS) requirements and private preferences. It shapes an on-demand approach to dynamic service composition in open environments.

• A validation of the multiagent coalition formation framework is provided with a demo application for demonstrating dynamic service composition in the domain of travel planning.

• The experimental evaluation of the dynamic behaviour of the multiagent coalition formation process investigates non-functional properties such as termination, completeness, stability and scalability. It further identifies the conditions under which an optimal/good solution can be achieved despite the non-deterministic behaviour of interacting autonomous self-interested software agents.

• A critical analysis of design and implementation issues of the multiagent coalition formation framework documents inherent difficulties and limitations of the software agent paradigm when based on the weak notion of agency. These shortcomings stem from the weak notion’s complexity that is implied by the combination of characteristics of concurrent systems, real-time systems, asynchronous messaging, efficient message multicasting and local decision-making under uncertainty.
Acknowledgements

The candidate acknowledges the services of Dr. Geoffrey Marnell, a professional proof-reader and editor, who marked up a copy of this thesis to indicate flaws in:

- syntax
- grammar
- spelling
- punctuation
- idiom.

Contact:
Dr Geoffrey Marnell
Abelard Consulting Pty Ltd
Level 8, 350 Collins Street
Melbourne, Victoria, 3000
Australia
www.abelard.com.au
Declaration

I hereby declare that this thesis only contains material which has been obtained as the outcome of research work conducted by the candidate. To the best of the candidate’s knowledge, this thesis does not contain material previously published or written by another person except where due reference is made in the text. Also, this thesis contains no material which has been accepted for the award to the candidate of any other degree or diploma.

Melbourne, July 2009
Contents

1 Introduction ................................. 1
  1.1 Background ................................ 1
  1.2 Motivation ................................. 6
  1.3 Research Contribution ..................... 11
  1.4 Example Application Scenario .............. 14
  1.5 Thesis Outline ............................. 16

Part I Foundation .......................... 19

2 Prerequisites ............................. 21
  2.1 Service-Oriented Computing ............... 21
    2.1.1 Origins of Service-Oriented Computing ... 23
    2.1.2 Benefits of Service-Oriented Computing ... 26
  2.2 Service Composition ...................... 28
  2.3 Open Environments ....................... 31
  2.4 Notion of Agency .......................... 34
    2.4.1 Definition of the Weak Notion of Agency ... 36
    2.4.2 Interpretation of the Weak Notion of Agency ... 37
    2.4.3 Multiagent Systems and Open Environments ... 45
  2.5 Agent-Oriented Software Engineering ....... 46
    2.5.1 Component Level Definition ............... 49
    2.5.2 System Level Definition .................. 50
    2.5.3 Agent Orientation and Service-Oriented Paradigm ... 51
  2.6 Summary .................................. 53
CONTENTS

3 Related Work

3.1 Semantic Web Approaches ........................................ 56
  3.1.1 Service Brokers ........................................... 56
  3.1.2 Central Composers ....................................... 59
3.2 Multiagent Systems .................................................. 60
  3.2.1 Multiagent Middleware .................................... 60
  3.2.2 Agent-based Frameworks ................................ 63
3.3 Automated Planning and Formal Methods ............................. 67
  3.3.1 Petri Nets ............................................... 69
  3.3.2 Constraint Satisfaction ................................... 69
  3.3.3 Case-based Reasoning ................................... 70
  3.3.4 Procedural Reasoning ................................... 70
3.4 Other Approaches .................................................... 71
  3.4.1 Negotiations and Auctions ................................ 71
  3.4.2 Declarative Specification Languages ..................... 73
  3.4.3 Compositional Agents ................................... 74
  3.4.4 Mobile Agents ........................................... 76
3.5 Summary .............................................................. 78

Part II Contribution

4 On-demand Service Composition in Open Environments ........... 83
  4.1 Scope and Environment ........................................ 83
  4.2 On-Demand Service Composition Concept .......................... 88
    4.2.1 Service Request Creation ................................ 90
    4.2.2 Service Planning ....................................... 92
    4.2.3 Service Discovery and Matchmaking ..................... 98
    4.2.4 Service Selection ....................................... 103
    4.2.5 Service Contracting ................................... 108
  4.3 Discussion of Key Characteristics ................................ 110
    4.3.1 Key Concepts and Principles ............................. 112
## CONTENTS

4.3.2 Advantages .......................................................... 120
4.3.3 Shortcomings .......................................................... 122
4.4 Constraints and Assumptions ........................................... 123
4.5 Summary ................................................................. 125

5 On-demand Coalition Formation in Open Environments ........ 127
  5.1 Concept of Coalition Formation ..................................... 127
  5.2 Relevant Coalition Formation Approaches ......................... 131
  5.3 Model for Dynamic Coalition Formation ........................... 140
    5.3.1 Coalition Structure Generation ............................... 142
    5.3.2 Solution of the Optimisation Problem ....................... 145
    5.3.3 Payoff Distribution Calculation .............................. 149
  5.4 Summary ................................................................. 151

6 Multiagent System Design ............................................. 153
  6.1 Multiagent System Architecture ................................... 153
  6.2 Software Agent Roles .............................................. 156
    6.2.1 Service Consumer Agent Roles ............................... 157
    6.2.2 Service Provider Agent Roles ............................... 158
  6.3 Data Structures ..................................................... 160
    6.3.1 OWL-S Process Model ........................................... 161
    6.3.2 Atomic Processes .............................................. 163
    6.3.3 Composite Processes .......................................... 164
    6.3.4 On-demand Service Composition Elements ................... 167
  6.4 Interaction Protocols ................................................ 169
    6.4.1 Composite Service Request Interaction Protocol ........... 171
    6.4.2 Service Advertisement Dissemination Interaction Protocol ... 178
    6.4.3 Coalition Initiation Interaction Protocol ................... 181
    6.4.4 Coalition Extension Interaction Protocol ................... 187
    6.4.5 Leader Leave Interaction Protocol ........................... 190
    6.4.6 Member Leave Interaction Protocol ........................... 192
    6.4.7 Coalition Completion and Contracting Interaction Protocol ... 194
6.5 Summary ......................................................... 196

7 Software Agent Design ........................................ 203

7.1 Agent-internal Structure ...................................... 203
7.2 Agent-internal Behaviour ........................................ 207
  7.2.1 Service Consumer Agent ..................................... 209
  7.2.2 Service Provider Agent ..................................... 215
7.3 Automated Planning ............................................. 228
  7.3.1 HTN Planning ................................................ 231
7.4 Capability Matchmaking ........................................ 241
  7.4.1 Matching Algorithm ......................................... 247
7.5 Expected Utility Calculation ................................. 252
  7.5.1 Expected Utility of Service Provider Agents ............. 253
  7.5.2 Expected Utility of Success and Failure Consequence .... 254
  7.5.3 Probability of Success and Failure Consequences ....... 257
  7.5.4 Expected Utility of Coalitions ............................ 259
7.6 Basic Decision-making Capabilities ......................... 260
  7.6.1 Coalition Complete Decision ............................... 260
  7.6.2 Proposal Ranking and Selection ............................ 260
7.7 Summary ......................................................... 261

Part III Assessment .............................................. 263

8 Demo Application ............................................... 265

8.1 Scope .......................................................... 265
8.2 Overview ....................................................... 266
8.3 Graphical User Interface ....................................... 269
8.4 Use Case Scenarios ............................................. 271
  8.4.1 Scenario 1 .................................................. 272
  8.4.2 Scenario 2 .................................................. 275
  8.4.3 Scenario 3 .................................................. 277
8.5 Summary ......................................................... 279
## 9 Experimental Evaluation

9.1 Experimental Setup ........................................... 281
9.2 Termination and Completeness ................................. 284
9.3 Optimality ..................................................... 288
9.4 Scalability ..................................................... 294
9.5 Stability ....................................................... 298
9.6 Summary ....................................................... 301

## 10 Lessons Learnt

10.1 General Conceptual and Design Issues ....................... 303
  10.1.1 Agent-Oriented Abstraction and Decomposition .......... 304
  10.1.2 Human–Agent Interaction ................................ 306
  10.1.3 Integration of Legacy Software .......................... 308
  10.1.4 Integration of Middleware Services ....................... 309
  10.1.5 Agent Orientation as Uniform Paradigm ................. 310
  10.1.6 Agent Orientation as a Universal Paradigm ............. 311
10.2 Design and Implementation Issues of Interaction Protocols .. 313
  10.2.1 Concurrency Issues in Agent Interactions ............... 314
  10.2.2 Implementation Issues of Unicast Message Timeouts ...... 322
  10.2.3 Implementation Issues of Multicast Message Timeouts ... 327
  10.2.4 Calculation of Message Timeouts ........................ 330
  10.2.5 Race Conditions ......................................... 332
10.3 Runtime Issues in Multiagent Systems ....................... 333
  10.3.1 Ageing Information ....................................... 334
  10.3.2 Ageing of Messages ...................................... 334
  10.3.3 Uncertainty about the Status of Communication Partners .. 335
  10.3.4 Limited Exception Handling .............................. 336
  10.3.5 Adaptation to Failure .................................... 337
  10.3.6 Balanced Behaviour ...................................... 339
10.4 Summary ....................................................... 340

## 11 Conclusions

11.1 Lessons Learnt ................................................ 341
# CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.1 Summary</td>
<td>341</td>
</tr>
<tr>
<td>11.2 Contributions</td>
<td>343</td>
</tr>
<tr>
<td>11.3 Future Work</td>
<td>348</td>
</tr>
<tr>
<td>References</td>
<td>351</td>
</tr>
<tr>
<td>Appendix</td>
<td>375</td>
</tr>
<tr>
<td>A AUML Quick Reference</td>
<td>A-1</td>
</tr>
<tr>
<td>B List of Publications</td>
<td>B-1</td>
</tr>
</tbody>
</table>
List of Figures

1.1 Stepwise specialisation of the problem, from the general background to the specific research outcome. ........................................ 12
1.2 Alternative transportation combinations for an outbound trip from Melbourne to Sydney. ........................................ 15
1.3 Alternative plan refinements of an outbound trip from Melbourne to Sydney. 16
1.4 Thesis Structure. ......................................................... 18

2.1 Conceptual overview over the Basic SOA. ................................ 24
2.2 Conceptual overview of the Extended SOA according to Papazoglou [134]. 26
2.3 Outline of composite service life-cycle stages and service composition activities. ......................................................... 29
2.4 UML specification of a software agent component. .................... 49

3.1 UML component diagram of the concept of a Semantic Web service broker approach. .................................................. 56
3.2 UML component diagram of the concept of a middleware multiagent system approach. ............................................... 61
3.3 UML component diagram of the concept of an automated planner approach. 68
3.4 UML component diagram of the concept of a compositional agent approach. 75

4.1 Conceptual model of an open market environment. ....................... 85
4.2 Visualisation of the complexity of the open market concept with the dungeon metaphor. ................................................ 87
4.3 Visualisation of the on-demand service composition concept with the classroom metaphor. ........................................... 89
LIST OF FIGURES

4.4 Example decomposition tree for Task 1: outbound trip from Hawthorn, Melbourne to Sydney city centre. ................................................................. 96
4.5 Visualisation of the linking of service advertisements to the tasks of Composite Service Request 1 for Task 1: outbound trip from Hawthorn, Melbourne to Sydney city centre. ................................................................. 102
4.6 Comparison of data and control flows in a) central top-down and b) decentralised bottom-up approaches. ............................................................... 112

5.1 UML state diagram illustrating the steps of the proposed coalition algorithm. 143

6.1 Generic architecture of a multiagent system that models on-demand service composition in an open market environment. ................................. 154
6.2 UML class diagram of the class hierarchy of service consumer agent roles. 157
6.3 UML class diagram of the service provider agent roles. .............................. 158
6.4 OWL-S Process Model as specified in Martin et al. [114]. ............................ 162
6.5 Schematic overview of the interaction protocols and their relationships. .... 171
6.6 AUML sequence diagram of the composite service request interaction protocol. ........................................................................................................ 173
6.7 AUML sequence diagram of the service advertisement dissemination interaction protocol. .................................................................................... 179
6.8 AUML sequence diagram of the coalition initiation interaction protocol. .. 182
6.9 AUML sequence diagram of the coalition extension interaction protocol. .. 188
6.10 AUML sequence diagram of the leader leave interaction protocol. .......... 190
6.11 AUML sequence diagram of the member leave interaction protocol. ....... 192
6.12 AUML sequence diagram of the coalition completion and composite service contracting interaction protocol. ...................................................... 195

7.1 UML component diagram of the main elements of the service consumer agent type. ............................................................................................. 204
7.2 UML component diagram of the internal elements of the service provider agent type. ....................................................................................... 205
7.3 UML state machine diagram of the StateMachine component of the service consumer agent type. ........................................................................ 209
LIST OF FIGURES

7.4 UML state machine diagram of the StateMachine component of the service provider agent type. .................................................. 216
7.5 Simplified visualisation of a planning process to illustrate the shortcomings of the SHOP2 algorithm with respect to split and split-join control structures. 237
7.6 Venn diagrams visualising different degrees of similarity of capability matching matching classes. ............................................. 244
7.7 Simple Illustration of plug-in matching between a service advertisement and a service request. ...................................................... 245

8.1 Architecture of the multiagent system of the demo application. ........ 267
8.2 Overview of the graphical user interface of the demo application. ...... 269
8.3 Screen shot of the Scenario 1 service request for a return trip from Melbourne to Sydney including a one-night stay. ......................... 272
8.4 Screen shot of the overview of the most beneficial composite service proposal of Scenario 1. ....................................................... 273
8.5 Screen shot of the expanded view of the outbound trip of the most beneficial composite service proposal of Scenario 1. ...................... 274
8.6 Screen shot of the overview of the second most beneficial composite service proposal of Scenario 1. ............................................. 274
8.7 Screen shot of the Scenario 2 service request for a return trip from Melbourne to Sydney including a one-night stay. ......................... 275
8.8 Screen shot of the overview of the most beneficial composite service proposal of Scenario 2. ....................................................... 276
8.9 Screen shot of the expanded view of the outbound trip of the most beneficial composite service proposal of Scenario 2. ...................... 276
8.10 Screen shot of the Scenario 3 service request for a return trip from Melbourne to Sydney including a one-night stay. ......................... 277
8.11 Screen shot of the overview of the most beneficial composite service proposal of Scenario 3. ....................................................... 278
8.12 Screen shot of the expanded view of the outbound trip of the most beneficial composite service proposal of Scenario 3. ...................... 278
LIST OF FIGURES

9.1 Completeness of test runs of test scenarios A–E. ......................... 285
9.2 Registration times of the first, second and third complete coalition of test runs of test scenarios A–E. .............................................. 286
9.3 Completeness of test runs of the test scenarios E and E’. ................ 287
9.4 Utility of the first, second and third registered coalitions of test runs of test scenarios A–E. .............................................. 289
9.5 Average utility of the most beneficial registered coalition of test runs of test scenarios A–E. .............................................. 291
9.6 Number of messages exchanged (excluding broadcasting) in test scenarios A–E. .............................................. 295
9.7 Number of messages exchanged (excluding broadcasting) in test scenarios E and E’. .............................................. 298
9.8 Average occurrences of members leaving and leaders abandoning coalitions in test scenarios A–E. .............................................. 299

10.1 Visualisation of the external and internal view to the design of a user agent.307
10.2 Sketch of the integration of legacy software systems into a multiagent system using proxy agents. .............................. 308
10.3 UML sequence diagram of a starvation situation of the sender agent. . . 317
10.4 UML sequence diagram of a deadlock situation between two bi-directionally communicating software agents in blocking mode and no message timeouts. 318
10.5 UML sequence diagram of a logical livelock situation between two bi-directionally communicating software agents without timeout and blocking modes. .............................................. 319
10.6 UML sequence diagram of a temporal livelock situation between two bi-directionally communicating software agents in blocking and timeout modes.320
10.7 UML sequence diagram of a uni-directional communication without concurrency problems due to timeout and blocking modes. ................ 321
10.8 UML sequence diagram illustrating the uncertainty of a recipient agent about the exact timing of its response message. ................. 323
10.9 UML sequence diagram showing the uncertainty of a sender agent about the exact timing of a response message. ......................... 324
10.10 UML sequence diagram depicting the general interactions between software agent and mailbox components during interactions with active message timeout. ................................................................. 325
10.11 UML sequence diagram of alternative A – response sent on time. .... 326
10.12 UML sequence diagram of alternative B – message timeout elapsed. .... 326
10.13 UML sequence diagram of alternative C – late response with session object. 327
10.14 UML sequence diagram of alternative D – late response without session object. ................................................................. 327
10.15 UML sequence diagram of a race condition caused by volatile network conditions and/or network distances between software agents. ................. 332
10.16 UML sequence diagram of an example of adaptation to the failure of communication partners across long-term interaction protocols. ................. 338

A.1 Overview of the key elements of the AUML interaction diagram notation. A-2
List of Tables

4.1 Simplified illustration of an example service request for a return trip from Hawthorn, Melbourne to Sydney city centre including a one-night stay. 92
4.2 Simplified illustration of an example method Method 1 decomposing Task 1: outbound trip from Hawthorn, Melbourne to Sydney city centre. 95
4.3 Simplified illustration of an example composite service request Composite Service Request 1 for Task 1: outbound trip from Hawthorn, Melbourne to Sydney city centre. 97
4.4 Simplified illustration of an example service advertisement for Task 12: outbound trip from Melbourne city centre to Sydney city centre. 101
4.5 Simplified illustration of possible complete example coalitions for Task 1: outbound travel from Hawthorn, Melbourne to Sydney city centre. 108
4.6 Simplified unnormalised ranking of two coalitions for three scenarios of client-defined QoS preferences for Task 1: outbound travel from Hawthorn, Melbourne to Sydney city centre. 109
5.1 Illustration of the dependency of the search space of possible coalitions for a single service provider agent on the number of tasks of a generic plan for games of different size. 145
5.2 Comparison of the complexity of the search space of possible coalition structures for a single service provider agent between a general cooperative game and a game based on this thesis’s coalition formation model for games of different size. 146
6.1 Semi-formal description of the composite-service-request message type. 174
6.2 Semi-formal description of the composite-service-proposal message type. 175
6.3 Semi-formal description of the composite-service-accept message type. 176
6.4 Semi-formal description of the composite-service-confirm message type. 177
6.5 Semi-formal description of the composite-service-reject message type. 178
6.6 Semi-formal description of the service-advertisement message type. 180
6.7 Semi-formal description of the coalition-request message type. 184
6.8 Semi-formal description of the coalition-accept message type. 185
6.9 Semi-formal description of the coalition-reject message type. 186
6.10 Semi-formal description of the coalition-inform message type. 189
6.11 Semi-formal description of the leader-leave message type. 191
6.12 Semi-formal description of the member-leave message type. 193
6.13 Semi-formal description of the coalition-complete-query message type. 197
6.14 Semi-formal description of the coalition-complete-inform message type. 198
6.15 Semi-formal description of the coalition-registration-confirm message type. 199
6.16 Semi-formal description of the coalition-registration-disconfirm message type. 200
6.17 Semi-formal description of the coalition-confirm message type. 201
6.18 Semi-formal description of the coalition-disconfirm message type. 202

7.1 Different terms, same concepts; a comparison of terminology of HTN planning and the on-demand service composition concept. 235

8.1 Concrete service advertisement of an intercity transportation service that offers airfares between Melbourne and Sydney. 268
8.2 Illustration of the use case of a return trip from Melbourne to Sydney including a one-night stay. 271
8.3 QoS preferences for three scenarios of the use case of a return trip from Melbourne to Sydney including a one-night stay. 271
8.4 Comparison of most beneficial proposals for the three scenarios of the use case of a return trip from Melbourne to Sydney including a one-night stay. 279

9.1 Experimental setup of the 5 representative test scenarios A–E. 282
9.2 Parameters of the modified test scenario E'. 287
9.3 Parameters of test scenarios F, F' and F''. 291
9.4 Comparison of the local preferences of Agent 3 in the test scenarios F, F' and F''. 292
<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.5</td>
<td>Utility of registered coalitions of test scenarios F and F'.</td>
<td>293</td>
</tr>
<tr>
<td>9.6</td>
<td>Number and optimality of outcomes of test scenarios F and F''</td>
<td>293</td>
</tr>
<tr>
<td>9.7</td>
<td>Communication costs of the broadcasting component of test scenarios A–E.</td>
<td>294</td>
</tr>
<tr>
<td>9.8</td>
<td>Comparison of the stability of test scenarios E and E'</td>
<td>301</td>
</tr>
<tr>
<td>10.1</td>
<td>Concurrency problems of different combinations of blocking, timeout and communication mode.</td>
<td>316</td>
</tr>
<tr>
<td>10.2</td>
<td>Comparison of the support of sender agent autonomy and recipient agent fairness of available multicast message and message timeout settings.</td>
<td>330</td>
</tr>
<tr>
<td>11.1</td>
<td>Overview of the advantages and shortcomings of the approach of on-demand coalition formation in open environments.</td>
<td>345</td>
</tr>
</tbody>
</table>
Listings

6.1 Extended-BNF specification of the structural definition of an atomic process based on the OWL-S process model. . . . . . . . . . . . . . . . . . . . . . 164
6.2 Extended-BNF specification of the structural definition of a composite process based on the OWL-S process model. . . . . . . . . . . . . . . . . 165
6.3 Extended-BNF specification based on the Standard Upper Ontology Knowledge Interchange Format (SUO-KIF) language, as proposed by Pease [135] and Niles and Pease [128], for the declarative description of the semantics of preconditions, effects, conditions and bindings. . . . . . . . . . . . . . . 167
6.4 Extended-BNF specification of the structural definition of key data structures of the on-demand service composition process. . . . . . . . . 168
7.1 Pseudocode of the definition of the class State. . . . . . . . . . . . . . . . 208
7.2 Pseudocode of an example main loop. . . . . . . . . . . . . . . . . . . . . 209
7.3 Pseudocode of the service consumer agent listener state reactive behaviour. 210
7.4 Pseudocode of the service consumer agent listener state proactive behaviour. 210
7.5 Pseudocode of the service consumer agent planner state reactive behaviour. 211
7.6 Pseudocode of the service consumer agent planner state proactive behaviour. 211
7.7 Pseudocode of the service consumer agent requester state reactive behaviour. 212
7.8 Pseudocode of the service consumer agent requester state proactive behaviour. 212
7.9 Pseudocode of the service consumer agent requester state reactive behaviour. 213
7.10 Pseudocode of the service consumer agent requester state proactive behaviour. 214
7.11 Pseudocode of the service provider agent idle state reactive behaviour. . . 216
7.12 Pseudocode of the service provider agent idle state proactive behaviour. . 217
7.13 Pseudocode of the service provider agent idle state proactive behaviour. . 218
7.14 Pseudocode of the service provider agent idle state proactive behaviour. . 218
7.15 Pseudocode of the service provider agent candidate state reactive behaviour. 219
7.16 Pseudocode of the service provider agent candidate state reactive behaviour auxiliary procedure. .......................................................... 220
7.17 Pseudocode of the service provider agent candidate state proactive behaviour. 221
7.18 Pseudocode of the service provider agent leader state reactive behaviour. 222
7.19 Pseudocode of the service provider agent leader state proactive behaviour. 224
7.20 Pseudocode of the service provider agent leader state reactive behaviour auxiliary procedure. .......................................................... 224
7.21 Pseudocode of the service provider agent member state reactive behaviour. 225
7.22 Pseudocode of the service provider agent member state reactive behaviour auxiliary procedure. .......................................................... 226
7.23 Pseudocode of the service provider agent member state proactive behaviour. 227
7.24 Pseudocode listing of the SHOP2 planning procedure according to Nau et al. [126]. .......................................................... 234
7.25 Pseudocode listing of the recursive planning procedure. ................. 239
7.26 Pseudocode listing of the main matchmaking procedure. .................. 248
7.27 Pseudocode listing of the context filter. ........................................ 248
7.28 Pseudocode listing of the QoS filter. .......................................... 249
7.29 Pseudocode listing of the signature filter. .................................... 249
7.30 Pseudocode listing of the type matching procedure. ....................... 250
7.31 Pseudocode listing of the constraint filter. .................................... 251
7.32 Pseudocode listing of the atom matching procedure. ....................... 252
Chapter 1

Introduction

The central problem addressed in this thesis is the dynamic composition of services in open environments. Service composition embodies the core principle of service-oriented computing (SOC), which in turn is a lively field of research aimed at developing a universal software paradigm for programming distributed systems.

1.1 Background

A distributed system is a computer system comprising multiple independent computers that are connected by a network in order to coordinate their activities and to share resources. Distributed systems are characterised by a number of specific properties that make them a difficult problem to deal with. For example, computers usually operate with differing types of processors, memory capacities and operating systems. Network connections are not always reliable and also vary in bandwidth and throughput because they connect over disparate physical media and communication channels, e.g. fibre and copper wires or wireless, via Wireless-LAN and Bluetooth. In addition, software components of different vendors are implemented in various programming languages and often communicate based on dissimilar message and transport protocols.

The foundation of distributed systems, as they are referred to today, was laid in the early 1990s with the creation of the Internet as the ‘network of networks’ by interconnecting several existing computer networks using the then increasingly popular TCP/IP protocol. The major advantage of this relatively young package-switching technique to
work seamlessly across heterogeneous pre-existing networks on the one hand and strong commercial interests on the other hand drove the rapid growth in networking over the last two decades. Today, advanced networking technology is employed for seamlessly interconnecting a wide spectrum of electronic devices, from powerful stationary mainframes to small resource-limited mobile devices.

The last two decades have also witnessed a tremendous proliferation of computational devices into almost all aspects of modern life. Significant improvements in performance, energy consumption and miniaturisation of computer hardware, as well as dramatic drops in production costs, leveraged computer technology to penetrate at large business and manufacturing systems as well as offices and homes.

Consequently, both trends – advanced networking and ongoing pervasion of computing devices – provide the technical foundation for ever more complex distributed systems. Firstly, they fuel the development of new software applications that provide functionality for a wider range of tasks and more complex processes. Secondly, they pave the way for coordinating and automating these processes and tasks across originally isolated software applications. Larger companies, for example, push the automation of their business processes across existing operational systems, such as supply chain management, customer relationship management, content management, enterprise resource planning, internal communication, etc., in order to increase efficiency and governance of their businesses. The current Internet trend of Cloud Computing takes the integration and automation challenges beyond even organisational boundaries by propagating a vision of seamless access to data, applications and resources anywhere and anytime.

A universal programming paradigm is desirable in order to successfully achieve the envisioned level of integration and automation of business processes and to foster seamless information exchange in complex distributed systems. Such a programming paradigm must be highly interoperable, must consequently be centred on networked resources and must address aforementioned characteristics of distributed systems. In this context, SOC has evolved in recent years as a unified standards-based paradigm for programming distributed systems that is aimed at leveraging new levels of interoperability and automation to the implementation of business processes across heterogeneous applications and across organisational boundaries.
1.1. BACKGROUND

Service-Oriented Computing

The foundation of SOC was laid with the development of the Web-service concept in 2000. The W3C (Booth et al. [16]) defines a Web service as “an abstract notion describing a software system designed to support interoperable machine-to-machine interaction over a network.” A Web service specifies and exposes a programming interface of an associated remote application or resource in a universal and uniform way. Thus Web services support language and platform independent integration of originally heterogeneous networked applications and resources. All Web-service standards are specified with the extensible mark-up language (XML) to ensure that interoperability. Besides interoperability, the Web-service notion offers additional beneficial features such as loose coupling between different Web service components, increased reusability, late binding and runtime client code generation. All properties are aimed at supporting rapid application development based on code reuse and easy component reconfiguration or adaptation.

This so-called basic service-oriented architecture (Basic SOA) paves the way for uniform network-wide programming of distributed systems. However, it is predominantly designed for point-to-point integration purposes. Papazoglou [134] describes the extended service-oriented architecture model (Extended SOA) that has been introduced to achieve a paradigm shift away from pure integration aspects towards a universal service-oriented paradigm for programming distributed systems. The cornerstone of the paradigm shift is the abstraction from concrete Web-service technology towards an abstract service notion in order to establish a universal foundation for the fundamental building blocks of SOC in general. Extended SOA defines an augmented framework adding two additional layers for service composition and composition management respectively on top of the Basic SOA model. The extra layers add standards for specifying activities such as service discovery, service selection, service orchestration or choreography as well as access and security management. As a result, Extended SOA supports the design, implementation and execution of complex business processes by encompassing the full life-cycle of a composite service from integration and composition to run-time management.

This thesis is concerned with service-oriented principles in general. Therefore, the remainder of this thesis refers to the abstract service notion. A brief introduction to service-oriented key principles is presented in Section 2.1.
Service Composition

The concept of service composition is the core principle and key challenge of SOC. Service composition transforms logical business processes into concrete service-oriented implementations. It can only be performed if two basic prerequisites are satisfied. On the one hand, business analysts must have identified one or multiple business processes based on a thorough analysis of organisational structures and mission statements. On the other hand, software architects must have established an inventory of existing dissimilar atomic IT services agnostic to any particular business process. The atomic services are then used to create composite services with regard to the business processes. This process of planning, discovering, arranging, selecting and connecting several atomic services – each providing complementary functionalities – is called service composition. Service composition is central to this thesis. A full definition of the term is presented in Section 2.2.

Given its close relationship to business process modelling, a natural approach to service composition is to apply techniques from the workflow domain. The Workflow Management Coalition [160] specify a workflow as an “automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules.” Accordingly, a composite service can be understood as a service workflow that implements a business process as a set of linked activities embodied by interconnected concrete services (e.g. Web services).

Prevailing service composition solutions in industry are workflow-driven. They are focused on ensuring full control over business processes at predictable costs. These major benefits are, however, achieved at the expense of a significant shortcoming: workflow-driven approaches are of rigid hard-coded nature. They typically utilise design-time service composition only, making resulting implementations vulnerable to unforeseen dynamic changes in the environment or to the business processes themselves. For example, what happens if:

- Particular services are temporarily unavailable or become obsolete?
- Quality-of-service (QoS) requirements that vary for different service consumers need to be satisfied?
- Service-level agreements (SLA) need to be negotiated before service use?
• Multiple alternative services exist for a particular service type?

• Relationships between service providers and service consumers change dynamically?

Pure workflow-centric approaches to service composition do not support run-time service composition because of the rigid nature of the workflow concept. Therefore, all possible anticipated changes or exceptions must be explicitly modelled beforehand, which is rendered impossible considering the increasing complexity of distributed systems and of the business processes implemented on top of them.

Thus workflow-driven approaches may be suitable for application within a single organisation in which full control over the processes is feasible. However, more agile and flexible solutions are necessary for facilitating cross-organisational service composition when full control cannot be imposed, e.g. among autonomous peers with, in general, variable relationships and dependencies. These peers may experience volatile network conditions or leave and enter environments at will. Moreover, peers typically do not disclose their potentially conflicting private information and perform autonomous decentralised decision-making while competing for limited resources or collaborating to gain profits. The resulting complex environments spanned by such peers are known as open environments. They exhibit the characteristics of partial observability, non-determinism and dynamics, which together imply that no single peer can impose full control nor obtain complete knowledge at any time during the service composition process. A detailed definition of open environments is presented in Section 2.3.

In short, open environments evolve as particularly complex distributed systems in a number of application scenarios, e.g. in volatile ad hoc and peer-to-peer networks or in dynamic on-demand e-business settings. Due to their inherent uncertainty, open environments introduce additional challenges with respect to service composition that cannot be dealt with adequately using static design-time approaches. Hence the central problem addressed in this thesis is the dynamic composition of services in open environments.

Agent-based Service Composition

Given its characteristics, the problem of dynamic service composition has been intensively studied by the software agent community because the concept of software agents provides
a natural metaphor for modelling the aforementioned peers. In fact, the software agent community has been establishing another school of research into distributed systems over the past two decades using a different perspective centred on flexible interactions and distributed decision-making among sets of software agents (see Weiss [167] for an overview; particularly Chapter 3 by Durfee [38]). Even though no universal definition of the term exists, a software agent, in general, can be characterised as an autonomous problem-solver that acts in a goal-oriented manner based on decentralised reasoning and decision-making capabilities it is endowed with. A detailed discussion of the definition and interpretation of the most commonly quoted weak notion of agency is provided in Section 2.4.

The agent-based approach to distributed systems can be seen as complementary to service-oriented endeavours as discussed in detail in Foster et al. [54]. Foster et al. remark that service orientation is focused on “brawn” by defining stable infrastructures and standards to support seamless interoperability and integration. In contrast, agent orientation is centred on the “brain” by modelling flexible dynamic decision-making and goal-oriented problem-solving techniques. As a consequence, Foster et al. identify synergies between the two research domains and concludes that “brain and brawn need each other” in order to tackle future challenges of programming distributed systems.

In this respect, agent technology can be applied for augmenting static service workflows with dynamic decision-making and flexible interactions at several stages during the service composition life-cycle in order to increase overall runtime flexibility.

Moreover, a system of multiple software agents naturally matches the characteristics of open environments due to agent autonomy and multiagent-induced decentralisation. As a result, the problem of dynamic service composition in open environments can be modelled and implemented as collaborative problem-solving in multiagent systems. The existence of such a mapping between service composition in open environments and multiagent-based collaborative problem-solving forms the foundation of this thesis.

1.2 Motivation

Existing agent-based approaches to dynamic service composition are technically sound. They improve flexibility of the service composition process in comparison to workflow-driven solutions. However, a review of related agent-based approaches (presented in Chap-
Chapter 3) reveals limitations of their applicability in open environments that usually result in closed systems despite the intrinsic similarities and complementary character stressed above. Existing agent-based approaches to dynamic service composition are based on assumptions and constraints that do not hold for open environments, particularly with regard to the aspect of uncertainty in open environments. They usually exhibit one or more of the following shortcomings:

- **Complete knowledge**
  The agent system architecture incorporates either a single central broker agent or a static hierarchy of distributed middle agents. Both agent types are crucial for the mediation between service consumers and service providers. They perform and fully control the processing of service requests. To do so, the central broker or middle agents need to possess access to complete knowledge about the availability and state of other software agents and services in the system. These assumptions violate the partial observability of open environments (see Section 2.3).

- **Central repositories**
  Knowledge is stored in central repositories (e.g. blackboards, service registries, relational databases). Central repositories are hard to maintain with regard to completeness and timeliness of information, as the little success of service registries on the Web suggests. (Big players such as IBM, Microsoft and SAP announced the closing down of their public UDDI registries in January 2006.) Reasons for the maintenance difficulties are the dynamic and non-deterministic nature of open environments (compare Section 2.3).

- **Centralised data and control flows**
  Centralised data and control flows imply that a single software agent can exhibit full control over its environment. This constraint violates partial observability and non-determinism of open environments (see Section 2.3) because it is based on the assumption that all other software agents in a multiagent system interact as triggered in a strict client–server fashion.

- **Centralised decision-making**
  Broker agent approaches are obviously based on centralised decision-making. Mid-
dle agents in multiagent approaches also act as central components and thus initiate and control interactions with other software agents in that role in a strictly centralised and hierarchical manner. The hierarchy may incorporate distributed decision-making at lower levels. However, the decision-making of multiple software agents is still centrally controlled due to central data and control flows in the agent hierarchy. This constraint violates partial observability, dynamics and non-determinism of open environments (see Section 2.3).

- **Neglect of private information**
  Central broker or middle agents consider only information for decision-making that contributes positively to a problem solution in order to guarantee optimal outcomes. All other software agents are assumed to share a common goal or incentive, especially in consent with the central broker or middle agent. These assumptions violate the non-determinism and partial observability of open environments (see Section 2.3). In open environments, different software agents may not share a common goal due to local potentially conflicting information and preferences. They may lead to competition over limited resources and/or to the reduction in the number and size of sets of potential collaboration partners.

- **Focus on service selection**
  Service composition is in many cases centred only on dynamic service discovery and selection. It is assumed that pre-defined workflow process definitions for composite services exist as well as a sufficient number of appropriate software agents and services. This assumption violates the dynamics and non-determinism properties of open environments (compare Section 2.3). The availability of software agents in the system cannot be controlled and may change at any time, increasing the risk that existing approaches fail under these conditions.

The apparent gap between the characteristics of open environment and concrete agent-based service composition approaches stems from the absence of a universal software agent definition in general and the inconsistent use of a particular definition for modelling the fundamental building blocks of the software agent paradigm.
1.2. MOTIVATION

Firstly, agent-oriented software engineering is aimed at promoting the agent paradigm as a uniform and universal software engineering paradigm for programming distributed systems in an agent-based, systematic and disciplined fashion. As such, Agent-oriented Software Engineering needs to be based on a universal definition of its fundamental building blocks. However, the agent community still lacks general consensus on a universal definition of the notion of agency.

Secondly, existing definitions are specified with complex terms (e.g. autonomy, proactiveness) that are subject to subjective interpretation and therefore result in ambiguity. As a consequence, various agent-based approaches to dynamic service composition have been developed according to different agent definitions. Furthermore, these approaches typically do not apply their underlying agent definition in a universal manner, resulting in multiagent systems that comprise software agent types with differing basic properties.

Both problems are well known and have been repeatedly argued in related literature as major setbacks of the large-scale adoption of agent technology in industry (compare Section 2.5). They materialise in the limitations of agent-based approaches to dynamic service composition stressed above. Existing agent-based approaches exhibit one or more of the following inconsistencies with respect to the most common notion of agency (see Section 2.4) and a unified agent-oriented software engineering paradigm (see Section 2.5):

- **Component-level inconsistencies**
  Centralisation of data and control flow, centralised decision-making and the neglect of local preferences in central broker agents or middle agents often result in violation of the autonomy (see Section 2.4) of other software agent types in a multiagent system because the central components exhibit full control. Non-central software agents are usually designed to react to requests in a strict client–server fashion. They do not act independently according to local, possibly conflicting, preferences and goals. The bi-directional character of such implicit dependencies between two agent types also causes violation of the proactiveness (see Section 2.4) of non-central software agents, leaving them purely reactive (compare Section 2.4) in many cases. A second aspect is the application of wrapper or proxy agents for the integration of external server functionality (e.g. legacy applications, Web services) into multiagent systems. These software agents usually only convert requests and responses to
passive entities based on the client–server paradigm. They violate the proactiveness property of the agent definition.

- **System-level inconsistencies**

  Asynchronous messaging is crucial for ensuring agent autonomy. However, asynchronous messaging incorporates uncertainty (see Section 2.4) because the status of a request and the timeliness of a response are undecidable for sender and receiver agent respectively. Such temporal issues can be partially remedied with the use of a message timeout mechanism (see Section 2.4). Nevertheless, the intrinsic uncertainty of asynchronous messaging among autonomous decision-makers leads to non-deterministic system behaviour because of missing synchronisation as well as unknown network latencies and decision-making delays. In contrast, the majority of agent-based approaches exhibit deterministic system behaviour and typically do not address message timeouts in their interaction protocol designs.

  A second aspect is the inconsistency of different software agent types within a multi-agent system with respect to a particular agent definition. For example, central broker or middle agents exhibit high degrees of autonomy and proactiveness, whereas proxy agents can be characterised as passive and reactive components with little or no autonomy. The variation between different agent types in a single multi-agent system is typically caused by a purely functional decomposition of a problem space, according to the design principle of separation of concerns, instead of agent-based organisational abstraction and decomposition as proposed by Zambonelli et al. [177]. Functional decomposition of a multiagent system imposes fixed dependencies between agent types a priori. The semantics of agent types are explicitly and invariably defined in terms of unbound benevolence: software agents always adopt the goals of others. Castelfranchi et al. [25] argue that benevolence limits the flexibility of agent behaviour and also proposes to base agent collaboration not on functional but social dependencies. Agent collaboration should emerge from the reciprocation between software agents based on disparate limited capabilities and resources. Thus existing approaches constrain flexibility because they contain pre-defined dependencies that do not emerge dynamically, e.g. with respect to local goals and preferences of different software agents and the system state at a specific point during run-time.
1.3. RESEARCH CONTRIBUTION

- **Paradigm-level inconsistencies**

Agent-based service composition approaches are not built consistently using exclusively software agents as building blocks. Most broker agent systems explicitly comprise Web-service components. Other multiagent systems incorporate additional Web services, UDDI registries, blackboards and databases. These additional components are included in the system design because they cannot be easily modelled as software agents, usually due to the complex characteristics of the weak notion of agency (e.g. autonomy, proactiveness, etc.). However, this way the uniformity of agent-oriented software engineering as a software engineering paradigm is violated. The mix of different notions and paradigms that can be found in many multiagent systems does not enforce characteristics of open environments and subsequently does not support a full and implicit mapping of the problem of dynamic service composition in open environments.

In other cases, the strict separation of agent-oriented and service-oriented principles into different layers also poses problems. A layered integration is typically implemented with gateway or proxy agents that are problematic on the component and system level as outlined above. Moreover, layered integration provides an explicit mapping between agent orientation and service orientation. It may lead to a strong alignment of agent-oriented principles to service-oriented concepts (e.g. reduction of social interactions to communicating simple service requests/responses). The resulting dependencies and implications limit the uniformity and independence of agent-oriented software engineering.

The motivation and aim of this thesis is to provide an agent-based approach to the problem of dynamic service composition in open environments that is based on a uniform software-engineering-like definition and uses the fundamental building blocks of the software agent paradigm.

### 1.3 Research Contribution

The main hypothesis underlying this thesis is that the software agent paradigm is well suited for developing an approach to dynamic service composition in open environments.
1. INTRODUCTION

As depicted in Figure 1.1, the problem of service composition in open environments embodies a specific subdomain of the general problem of service composition. One approach is to model it as multiagent-based collaborative problem-solving due to the similarities between multiagent systems and open environments and the complementary character of service orientation and agent orientation.

![Figure 1.1: Stepwise specialisation of the problem, from the general background to the specific research outcome.](image)

The main contribution of this thesis is the development of an on-demand multiagent coalition formation approach to dynamic service composition in open environments (the top of the problem stack shown in Figure 1.1). This thesis presents a practical and paradigmatic multiagent approach to dynamic service composition in open environments based on the weak notion of agency (see Section 2.4). The weak notion of agency can be interpreted to specify multiagent systems matching the characteristics of open environments. It can also be used to define a uniform agent-oriented software engineering paradigm that implicitly maps service-oriented principles (compare Section 2.5).

The approach presented is consequently designed to match the characteristics of open environments: it is on-demand to adapt to changes in the environment (thus matches the dynamics of open environments). It is decentralised to cope with uncertainty (matches the non-determinism and partial observability of open environments). Software agents are endowed with full autonomy. They provide services in a proactive manner according to local preferences and the basic incentive to gain profits by satisfying service consumer
1.3. RESEARCH CONTRIBUTION

requests in a market environment. The following are the particular contributions of this thesis:

- A detailed analysis of a concrete interpretation of the weak notion of agency and the subsequent derivation of a software agent component type that lays the foundation of a uniform agent-based software engineering approach. The resulting fundamental building blocks establish multiagent systems as open systems that exhibit the characteristics of open environments. They also utilise service-oriented design principles in support of a successful mapping of the problem of service composition in open environments into the software agent domain.

- The concept and design of a multiagent coalition formation framework models dynamic service composition in open environments as collaborative problem-solving in an open market. In this open market, supply and demand confront each other as autonomous peers in the form of service consumer agents and service provider agents, each with equal rights. The design is influenced by a number of concepts such as decentralisation, on-demand service provision, self-organisation, emergence and market-based control.

- The implementation of the multiagent coalition formation framework facilitates coalition formation among service provider agents with lightweight interaction protocols and decentralised multi-attribute utility-based decision-making according to client-defined QoS requirements as well as private preferences. It shapes an on-demand approach for dynamic service composition in open environments.

- A validation of the multiagent coalition formation framework is provided with a demo application for demonstrating dynamic service composition in the domain of travel planning.

- The experimental evaluation of the dynamic behaviour of the multiagent coalition formation process investigates non-functional properties such as completeness, stability, scalability and robustness. It further identifies the conditions under which an optimal/good solution can be achieved despite the non-deterministic behaviour of interacting autonomous self-interested software agents.
1. INTRODUCTION

- A critical analysis of design and implementation issues of the multiagent coalition formation framework documents inherent difficulties and limitations of the software agent paradigm when based on the weak notion of agency. These shortcomings stem from the weak notion’s complexity that is implied by the combination of characteristics of concurrent systems, real-time systems, distributed systems, the concept of agent autonomy and the openness of multiagent systems.

1.4 Example Application Scenario

One of the most referenced service composition problems in literature is travel planning. Wooldridge [169] for example gives one account of travel planning: “After specifying your requirements to your personal digital assistant (PDA), it converses with a number of different Web sites, which sell services such as flights, hotel rooms and hire cars. After hard negotiation on your behalf with a range of sites, your PDA presents you with a package holiday.”

Similarly, a travel planning scenario is also stressed in this thesis, although some generally known shortcomings exist. For example, service composition for travel planning does not exactly follow the classical iterative input–processing–output principle in which outputs of an activity form inputs of subsequent activities. Travel planning rather deals with strings of activities concatenated by temporal constraints. Intermediary results are not passed on for further processing but are rather accumulated into a final itinerary at the end of processing. However, it is still a good example domain because it supports easy visualisation and provides a vast number of scenarios: from simple to complex.

In general, travel planning is the process of organising a trip (e.g. business, conference, recreational) for one or more people with consideration of their preferences. The goal is to find the best itinerary which includes a proper organisation of means of transport and accommodation given well-defined constraints such as budget and time limitations or other qualitative requirements (e.g. first class seats, dietary needs, etc).

A manual approach to the problem usually results in an extensive time-consuming Web search. Service composition can be understood as a means of automating travel planning. The aim of automation is to reduce the amount of time spent on the task in hand and to increase the optimality of the outcome. A service composition approach to
travel planning models the resulting itinerary as a composite service, in which several services of available transportation and accommodation providers are combined. Thus a composite service forms merely an expression of intent between service consumer and service providers. Only after execution of a composite service, bookings will be arranged, payment will be made and e-tickets will be issued.

Service composition is aimed at automating the search for candidate services and the composition of an optimal set of these services. The determination of the optimal set of services is difficult. It is a combinatorial problem that depends on the number and type of available candidate services. In addition, service proposals may be obtained in different ways, e.g. by negotiation or by leaving them variable over time.

Consider a return trip from Melbourne to Sydney as an example. The trip can be decomposed into three activities: outbound travel, accommodation and inbound travel. Each activity itself is subject to further refinement. One possible refinement of the outbound activity could be the following sequence of activities: local transport from Melbourne city to airport, intercity transport from airport to airport and local transport from Sydney airport to city. Figure 1.2 illustrates possible combinations for the outbound trip based on a number of available service providers and their respective services. Each combination represents a valid solution/composite service. The best combination is the...
one satisfying the user-defined constraints best. Constraints may be a limited budget of 500 AUD or less, business class seating, the travel date to be June 17 and the arrival time in Sydney to be before 12 pm.

The given example is very simple. Usually many alternative combinations are possible. Figure 1.3 shows two alternatives for an outbound trip from Melbourne to Sydney. Example a) represents a decomposition of the trip into a single activity. Example b) illustrates a further combination, with the trip starting not in Melbourne city centre but in a suburb named Hawthorn.

![Figure 1.3: Alternative plan refinements of an outbound trip from Melbourne to Sydney.](image)

1.5 Thesis Outline

The structure of this thesis comprises an introductory chapter, three main parts and a conclusion. Chapter 1 sets out the purpose of the research that led to this thesis and gives some background to the areas of SOC and agent-based service composition.

The first main part establishes the foundation of this thesis. Chapter 2 presents
a number of fundamental terms and concepts that are crucial for the remainder of this thesis. Chapter 3 analyses existing agent-based approaches to service composition in detail in order to qualify the pros and cons of these approaches and to highlight differences to the approach proposed in this thesis.

The second main part presents the research contribution of this thesis. Chapter 4 describes a general concept for dynamic agent-based service composition in open environments and discusses its key characteristics, constrains and assumptions. Chapter 5 defines an on-demand coalition formation model based on this concept. Chapter 6 semi-formally specifies the system architecture of a multiagent system, including fundamental data structures and interaction protocols that facilitate on-demand coalition formation. Chapter 7 completes the second part of this thesis with the semi-formal definition of the internal designs of incorporated software agent types, including the decision-making capabilities that drive goal-oriented interactions among software agents.

The third main part is concerned with the assessment of the proposed on-demand coalition formation approach. Chapter 8 describes a demo application to illustrate and validate the on-demand coalition formation approach with a concrete use case from the travel planning domain. Chapter 9 presents the experimental setup and results of an empirical evaluation of the coalition formation based approach to dynamic service composition in open environments. Chapter 10 reports on lessons learnt during the design and implementation of the multiagent system with respect to the weak notion of agency and a uniform and universal agent-oriented software engineering paradigm.

Chapter 11 concludes this thesis by providing a summary of this thesis, including a comparison of the initial hypothesis and the outcome of this work. It furthermore draws conclusions and provides recommendations for potential future work.

The structure of this thesis is depicted in Figure 1.4.
1. INTRODUCTION

Figure 1.4: Thesis Structure.
Part I

Foundation
Chapter 2

Prerequisites

This chapter sets out the scope of the research by introducing fundamental terms and concepts that are essential for this thesis. The chapter lays the foundation for the reader to understand the general background of SOC (Section 2.1) and the concept of service composition (Section 2.2). Service composition is the core principle and key challenge of SOC and central to this thesis. Next, the characteristics of open environments are highlighted (Section 2.3). These form the target environment for the service composition approach of this thesis. The chapter further discusses the notion of agency and provides a definition and interpretation of the weak notion of agency (Section 2.4). The resulting fundamental building blocks are then used to outline a uniform agent-oriented software engineering approach for developing multiagent systems which exhibit the characteristics of open environments and also utilise service-oriented design principles (Section 2.5). This thesis develops an agent-based approach to service composition in open environments based on this foundation.

2.1 Service-Oriented Computing

SOC is a vast and rapidly growing area in computer science and the IT industry. Since it is still a relatively young paradigm, several viewpoints revolving around service-oriented principles are articulated in literature that often vary in their dissimilar focus on enabling technologies and standards. This section gives a brief outline of the universal key concepts of service orientation according to the comprehensive introductory and hands-on guides of Erl [40, 41].
2. PREREQUISITES

In general, SOC is aimed at the automation of business processes. Automation is facilitated with the composition of complex business logic from multiple smaller functional entities spanning across heterogeneous application silos of one or even multiple businesses. Thus before composition can be applied, the complexity of a problem space needs to be broken down into smaller problems and subproblems of manageable size. Service orientation utilises the fundamental design principle of separation of concerns to achieve functional decomposition of a problem. With separation of concerns, a problem is iteratively decomposed into ideally disjunct but complementary subproblems, until no further meaningful decomposition is possible. The notion of service is used to model atomic subproblems in a service-oriented way.

A service is an abstract model that defines a functional entity through a set of associated capabilities. The capabilities of a service are exposed with a number of documents called the service contract that specifies service functionality and API-like access to it as well as other meta-information, e.g. SLA properties and constraints. The service notion differs from other design approaches in that it strictly separates the descriptive service contract from any underlying technology. The resulting abstraction allows for great flexibility in service-oriented design and implementation and is a major key to support a universal approach to both service modelling and service composition. Service abstraction also yields independence of services from business processes. A service is used to model a task of a particular business process but it is neither bound nor limited to it and thus can be reused in other business process models.

The service-oriented design approach is complemented by service-oriented architecture (SOA), a distinct architectural model for facilitating strategic, goal-directed business process modelling and implementation (Erl [40]). SOA constitutes a logical framework for programming complex distributed systems with a strong focus on enhancing automation and increasing optimisation of business processes within a business. However, SOA does not exist in one single universal instance only. SOA is an abstract, technology-independent construct revolving around the notions of service, inventory and composition (compare Papazoglou [134]). Different implementations are possible depending on the use of specific technologies, tools and standards in accordance with the existing IT infrastructure and the mission statement of an organisation. SOA is governed by the service-oriented design
principles identified by Erl [41]:

- **Standardisation of service contracts**
  Standardisation is essential for enforcing interoperability. Standardised service contracts ensure a uniform and universal way to model contract design, e.g. to express service functionality, data models, data types and QoS assertions.

- **Service Abstraction**
  Service contracts expose an abstract description of a service’s interface independent of any underlying technology and implementation.

- **Service Loose coupling**
  Standardised service contracts and abstraction of contract from concrete implementation impose low coupling between different service components.

- **Service Reusability**
  Services are specified independent of underlying technology and business processes. They can be reused for creating different applications.

- **Service Autonomy**
  Due to contract abstraction and separation of contract and implementation, services exhibit a high level of independent control over their underlying implementation.

- **Services Discoverability**
  Services are published in inventories for easy access. Services can be discovered based on the interpretation of the meta information they provide in their contracts.

- **Service Composability**
  Services have the intrinsic ability to be composed into more complex services with respect to all the properties listed above.

### 2.1.1 Origins of Service-Oriented Computing

Service orientation is no revolutionary design paradigm. It evolved from preceding paradigms and technologies. Web services is the technology most closely intertwined with service orientation. In fact, most vendors of service-oriented tools and technologies have
shaped their initiatives around Web services because it naturally promotes many features that have been adopted as service-oriented design principles, such as abstraction, loose coupling, composability, discoverability and interoperability.

Web services were originally developed to facilitate point-to-point integration channels amongst heterogeneous software applications. The W3C (Booth et al. [16]) specify a Web service as “an abstract notion describing a software system designed to support interoperable machine-to-machine interaction over a network.” A Web service defines and exposes a programming interface of an associated remote resource or application with an abstract service contract in a universal and uniform manner. The aim of the Web-service platform is to provide interoperability between different systems regardless of vendor, platform and programming language. A key feature to achieve that mission is the use of the extensible mark-up language (XML) for the specification of service contracts and all further standards necessary to establish remote communication channels and data exchange between two end points. Many of those standards are adopted industry-wide.

Web services developed in two phases. Web service standards, marked as the first generation, form the fundamental building blocks of the Basic SOA as denoted by Papa-zoglou [134]. These standards specify a set of service artefacts and operations on Web services, which is often referred to as the publish–find–bind mechanism. Three roles are involved in the mechanism, as depicted in Figure 2.1:

![Diagram of Basic SOA](image)

**Figure 2.1:** Conceptual overview over the Basic SOA.
• Service providers specify service contracts with a WSDL file and publish them with a service registry. WSDL describes the operations, their parameters and the data types of a Web service as well as message formats and protocol bindings to a concrete service end point. A WSDL file is usually complemented/enhanced with an XSD file, a form of meta-information source for defining how to interpret the expressions of the WSDL file.

• Service clients use operations of the UDDI standard in order to retrieve information about available services. UDDI describes operations of a distributed directory in which service providers can list their services or the use by service clients on the Internet. UDDI is a concrete instance of the WSDL standard that is accessed with SOAP messages.

• After a service client has discovered a particular service, it establishes a communication channel and starts interacting by exchanging messages with the service based on meta information about service binding obtained from the associated WSDL. Message exchange typically occurs with the SOAP standard. SOAP defines a set of rules for designing message formats. It runs on top of a transport protocol (HTTP default) and thus supports remote procedure call.\textsuperscript{1}

The Web service standards of the first phase were augmented in a second phase with a number of additional standards in order to eradicate major shortcomings e.g. concerning message security and transaction safety. The additional standards are not presented here, since they do not change the fundamental functioning of the Basic SOA paradigm. However, it is important to note that maturity and thus acceptance of the Web service technology grew due to the emergence of additional standards. As a result an extended concept emerged in the form of the Extended SOA as described by Papazoglou [134]. This enhances the Basic SOA notion with additional layers for service composition and composition management (Figure 2.2). Extended SOA represents the concepts of the general service-oriented paradigm which has been derived through generalisation and abstraction.

Service orientation draws inspiration from several further paradigms and technologies. Two of them drive major characteristics of SOC. Firstly, business process modelling is

\textsuperscript{1}Refer to Walsh [166] for a comprehensive introduction and specification of Web services standards.
Concerned with computer-based implementation and automation of business processes in order to increase the efficiency and flexibility of businesses, it can be seen as the motor for service modelling which acts in the role of a service composition manager. Secondly, enterprise application integration deals with the creation of integration frameworks in order to federate application silos within and across businesses. It builds on interoperability and standardisation for data exchange across point-to-point communication channels. The point-to-point character, however, results in many cases in high redundancy and complexity as well as low reusability. This is because integration frameworks grow rather uncontrolled with every new integration project. Service orientation picks up the lessons learnt from enterprise application integration efforts and evolves into a universal paradigm with a unified strategy for tackling application integration and reuse, e.g., by incorporating a multi-functional and multi-directional integration middleware layer often labeled as service bus.

2.1.2 Benefits of Service-Oriented Computing

SOC has been emerging from the abovementioned technologies and paradigms. As such, it is not only aimed at pursuing the aims and goals of those existing technologies but to combine them in order to increase their value under the umbrella of one uniform and
universal software engineering paradigm for distributed systems. The foremost goal of service orientation is the automation of business processes in order to increase efficiency, agility and productivity of businesses. That strategic long-term goal is achieved through rigorous application of service-oriented design principles. They utilise intrinsic interoperability, increased scalability and improved reliability as a basis for reshaping an originally heterogeneous IT infrastructure. Erl [41] identifies the following effects of service-oriented design principles:

- **Increased federation and reduced integration needs**
  On the one hand, heterogeneous resources and applications remain fully autonomous and self-governed, but on the other hand, they can also be easily combined in a universal and uniform way. The number of proprietary incompatible application silos is reduced.

- **Additional optimisation opportunities**
  Application logic is not exclusive to one application, but can be combined with other applications. Synergies between different applications can be strengthened through a higher degree of automation between them. Hence applications can contribute to added value beyond their original scope.

- **Reduced time to market**
  Service reuse and service composition increase agility and flexibility of a business. Business processes can be adjusted to new situations without extensive coding or integration efforts in case of changing business needs.

- **Vendor-neutral strategy development**
  Businesses can develop strategies for emergence and migration of their IT infrastructure with a low dependency on vendor-specific development and rollout strategies.

- **Increased return on investment**
  All mentioned improvements have the potential to increase the efficient use of time, budget and resources.

With the emergence of electronic commerce – both business-to-business and business-to-consumer – service orientation also becomes increasingly important for modelling inter-
organisational business processes, enabling businesses to automate their entire value chain from their suppliers up to their customers. Services and actors no longer belong to a single but to multiple organisations. Besides automation and thus optimisation of existing business processes, service orientation also yields added value through opening up new business opportunities. For example, service composition utilises the creation of new complex services out of existing ones.

This thesis is centred on service-oriented principles. With a clear focus on interoperability, autonomy and composability, SOC provides a sound paradigm for integrating heterogeneous software systems in order to automate business processes. It is a compelling candidate for establishing the basic infrastructure for service composition in open environments. In open environments, business processes are embodied by composite services that are provided by one or multiple competing and collaborating independent self-interested organisations.

2.2 Service Composition

Service composition is at the core of SOC. It is the process of coordinated aggregation of a set of existing services. A composite service introduces a part–whole relation between the composite service as a whole and multiple component services as parts. Component services are also often referred to as atomic services. The complexity of component services is, however, not limited by their role in a particular composite service. Thus every component service may be the result of an earlier service composition process.

Service composition is programming in the large. It does not involve the creation of extensive business logic. Service composition rather produces documents as output that describe not only the set of incorporated component services but more importantly specify the relations between those component services by detailing data and control flow, message exchange patterns, artefacts of each component service and exception handling. They also regulate security policies, transaction handling and QoS constraints.

There are basically two different approaches available for specifying the relations between component services: orchestration and choreography. Both notions differ merely in the scope of control. Orchestration is focused on the view of one central point of control (analogue to a conductor of an orchestra), whereas choreography centres on distributed
control among all participating components that interact based on events (analogue to
dancers who react to the behaviours of their peers). Several standards have been devel-
oped in the past few years in order to facilitate service composition, such as Wf-XML,
WS-BPEL, WSCI, WSCL, WSFL and XLANG to name a few.\footnote{Refer to Mendling et al. [119] for an overview and pointers to the different standards.}

Service composition represents one of several stages embedded into a complete com-
posite service life-cycle. There is no universal definition of such a life-cycle model in the
literature. Figure 2.3 depicts a simplified life-cycle model comprising a sequence of four
major stages.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{composite_service_life_cycle.png}
\caption{Outline of composite service life-cycle stages and service composition activities.}
\end{figure}

The first stage is the business process \textit{modelling stage}. It is concerned with the analysis
of a business process to be implemented. Business process modelling produces essentially
two outcomes that are mandatory prerequisites for the subsequent service composition
stage. Firstly, a composite service aggregates multiple component services. Thus neces-
sary component services must be available and discoverable after the modelling step, e.g.
in a service inventory. Each task of a business process is identified as a conceptualised
service which is then transformed into one or multiple platform-specific service-model
representations according to existing IT infrastructure and with regard to a concrete
underlying technology. These service models are registered with a service inventory. Sec-
ondly, a blueprint, e.g. in the form of a business process model, is developed according
to which a composite service will be created. The blueprint details the tasks on the one
hand, and the design, flow and handling of messages exchanged between them on the
other hand. It forms the base of the above described eventual dependencies and relations
amongst component services.
**2. PREREQUISITES**

The *composition stage* follows next. Service composition is a complex design exercise comprising a number of activities. The composition process commences with the *service planning* activity. Planning is concerned with creating a plan of concrete service types that can be used to represent or decompose the conceptual service descriptions of the tasks of a business process model. In the simplest case, planning is reduced to a simple mapping of service models from the inventory into a business process model. However, such a simple mapping may not exist due to the characteristics and number of available services and the nature of a business process model.

All necessary service types are known after a plan has been created. This is the precondition for the next activity: *service discovery*. One or multiple service inventories are accessed in order to obtain information about concrete services. Service discovery is closely intertwined with the *service matchmaking* activity. Inventories contain service descriptions that are matched (e.g. capability matchmaking as proposed by Paolucci et al. [131]) against the service type descriptions of a plan. The result of discovery and matchmaking activities is a set of candidate component services.

The aim of the subsequent *service selection* activity is to determine the optimal set of component services out of the entire set of candidates. This is necessary for two reasons. Firstly, multiple candidate services may be available for a single task of a business process model. Secondly, business process models may contain QoS requirements for particular tasks that restrict the applicability of candidate services. QoS requirements specify non-functional requirements regarding service provision or service result, e.g. execution time constraints, security levels, costs incurred or the precision of result values. Only candidate services that comply are considered. Compliance is checked either by comparison of static QoS parameters as specified in service contracts or may be determined by more advanced methods, such as negotiation (e.g. positional bargaining as investigated by Faratin [45]), if the QoS parameters are specified variably. The outcome of comparison or negotiation is a ranking of candidate services according to their QoS parameters. Eventually, the set of optimal services is selected based on that ranking.

*Service contracting* is the last activity of the composition stage. SLAs are settled that do not only specify functional dependencies but also regulate QoS constraints, their enforcement and exception handling, e.g. regarding security and transaction safety. The
service composition stage is finalised with the publication of the composite service in a service inventory.

After service composition, subsequent stages deal with the deployment of a representation associated with a composite service’s contract into a service inventory and the run-time operation of a composite service. During operation the component services of a composition are enacted and executed according to the original business process model in order to meet its design objectives.

This thesis investigates an approach to dynamic service composition in open environments. Service composition is originally performed at design-time. However, the complexity of open environments poses direct implications that lead to a partial or even full shift of the composition process into runtime. This thesis investigates an approach to dynamic service composition based on a fully online service composition process. In particular, it is focused on dynamic service planning and service selection – the most complex activities in the service composition process – in order to tackle the challenging characteristics of open environments.

### 2.3 Open Environments

The term *open system* or *open environment* is being used with different meanings throughout academia and industry, often with emphasis on extensibility, interoperability or public access to software systems, component interfaces or source code. This thesis refers to open environments in a different context. Hewitt [69] coins the term *open system* with regard to the complexity of office information systems. Hewitt notes that office work is not that much concerned with location but rather with activities involved. Hence it can take place in many locations, e.g. in a car via a mobile phone or at a networked computer and involve numerous activities (compare Hewitt [69]) which are represented by logical and functional units that are referred to as business processes in the literature today.

Hewitt [69] is in line with current trends (see Section 1.1) by predicting an increasing proliferation of computing resources that lead to and drive a growing demand for business process automation. In this context, Hewitt [69] foresees that computer systems will increasingly exhibit the social structure of human organisations by “taking on more of the authority and responsibility for ongoing activities”. This statement can be interpreted...
as an ‘agentification’ of software components resulting inevitably in the application of intelligent agent and multiagent concepts. Therefore, this thesis follows Wooldridge [169] in using a classification scheme postulated by Russel and Norvig [146] in order to analyse the characteristics of open environments as outlined by Hewitt, based on a set of categories that are aimed at classifying the complexity of task environments of software agents.

The following list reiterates categories from Russel and Norvig [146] that directly match Hewitt’s characterisation:

- **Fully observable/partially observable**

  An environment is fully observable if a software agent can gain full knowledge about all information relevant for its decision-making at any point in time. This information includes the state of the environment, in particular the available software agents, their states and their behaviours. The opposite is a partially observable environment in which no single software agent has access to the complete state of the environment, e.g. due to inaccurate sensors, incomplete knowledge about the complexity of an environment or ageing of information.

- **Deterministic/stochastic**

  An environment is deterministic if a software agent can fully determine the next state of the environment depending on the current state and the actions it can perform. Fully observable environments are usually also deterministic and do not exhibit any uncertainty. However, partially observable environments may induce non-determinism, especially in complex environments in which it is impossible to keep track of all relevant aspects for decision-making. Non-determinism implies uncertainty about the outcome of actions, e.g. the same action performed at different times does not necessarily result in the same outcome. Non-deterministic environments are also known as stochastic environments.

- **Static/dynamic**

  An environment is static if it does not change from a software agent’s point of view while the software agent conducts reasoning and decision-making between two actions. In contrast, a dynamic environment may change from a single software
agent’s perspective even though it does not perform any actions itself. Thus the
information a decision is based on may become partially incorrect or completely
obsolete before an adequate action is taken. Dynamic environments are very complex
because a software agent needs to constantly update its knowledgebase with timely
information about its environment.

- **Discrete/continuous**

  An environment is discrete if it is represented by a finite number of discrete states. Continuous environments are characterised by an uncountable number of states. Whether an environment is discrete depends on how time is handled and whether the perception and actions of its software agents can be measured in discrete/continuous domains. Discrete environments are, according to Wooldridge [169], in any non-trivial case usually almost as complex as continuous environments. Even though they exhibit a finite number of discrete states, they are still difficult to handle because of a potentially very large number of states that may not be fully known (partially observable environments).

The most complex environments are partially observable, stochastic, dynamic and continuous. These environments are called open environments by Hewitt [69].

In the context of service composition, an open environment can be viewed as an open market which is regulated by demand and supply of multiple independent self-interested organisations. An open market is *partially observable* because self-interested organisations retain information about their internal states and local preferences. Thus no single organisation can exhibit full control over the entire environment. The number and nature of participating organisations cannot be fully determined at any point in time. Firstly, because of the dynamic nature of open environments; organisations may join and leave at any time. Secondly, because open environments span distributed systems in which potential network outages may temporarily limit observability.

An open market is *stochastic* because processing is based on concurrent local decision-making of and asynchronous messaging among self-interested organisations. Local decision-making is based on private preferences. It is also based on potentially incomplete and ageing information. As a consequence, the order, nature and impact of actions of various
organisations in the environment are uncertain from a single organisation’s point of view as well as on a global level. There are no fixed relationships; collaboration between different organisations is established with agreements. Agreement cannot be forced. Agreements cannot be enforced either. Breaches are possible.

An open market is dynamic because the environment changes constantly from a single organisation’s perspective even though it does not trigger any actions itself. Changes occur, firstly, because different organisations concurrently perform self-initiated actions and secondly because the information decisions are based on ages. The state of organisations and the services they offer (e.g. availability, functional and non-functional properties) can change at any point in time due to software or server failures, network outages, resource limitations, maintenance work, updates, etc. Whole services also may become completely obsolete. Further, new services may be provided. Thus organisations must be able to adapt to and to coordinate unplanned changes.

An open market is in most cases not continuous but complex discrete because of the nature of modelling and implementing organisations and services: their states, actions and relationships. States (e.g. a service is available/not available, executed/idle) and actions (e.g. a service is published, discovered, composed, negotiated, executed, etc.) are usually mapped into discrete domains. Their number is finite and countable, but potentially very high and in particular unknown due to the intrinsic uncertainty and decentralisation caused by the complexity of the other properties of open environments.

This thesis is focused on open environments. In the context of service composition, an open environment can be viewed as an open market inhabited by multiple self-interested organisations that compete and collaborate in order to provide requested composite services. It develops a multiagent approach for service composition that is compliant with the characteristics of open environments.

2.4 Notion of Agency

This thesis tackles dynamic service composition in open environments with a multiagent approach. Thus a proper definition of what actually constitutes a software agent is necessary. Unfortunately, no commonly accepted universal definition of the notion of agency exists. Wooldridge and Jennings [171] noted in 1995 that no universal definition has been
agreed on. d’Inverno and Luck [36] likewise noted the lack of consent in 2001 (“The contradiction of agent-based systems is that there is still an effort to provide a sound conceptual foundation...”). Padgham and Winikoff [130] do likewise in 2004 (“...there is not yet a universal consensus on the definition of an agent”). The problem appears to be unsolved to the present day.

It may be argued that the absence of a universal definition allows to explore the application of artificial intelligence techniques and methods in software system development. However, a clear profound understanding of the software agent concept is crucial in the author’s opinion for two reasons. The absence of a universal definition firstly hinders seamless exchange of research results within and across disciplines in the research field itself and secondly hampers dissemination of the software agent paradigm and relevant research results beyond the community. Given an unstable foundation, Wooldridge and Jennings [171] warned in 1995 that “…there is also the danger that unless the issue is discussed, ‘agent’ might become a ‘noise’ term, subject to both abuse and misuse...” And in fact, negative repercussions have been noticed, e.g. by d’Inverno and Luck [36] in 2001: “…the lack of a common understanding leads to difficulties in communication, a lack of precision (and sometimes even confusion) in nomenclature, vast overuse and abuse of the terminology, and a proliferation of systems adopting the agent label without obvious justification for doing so.”

For a start, consider the basic definition given by Wooldridge [169]: “An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives.” This definition gives a rough idea, but is not very concrete. As Padgham and Winikoff [130] remark, the property situatedness does not really limit the notion of agency since actually every sort of software is situated in an environment in one or another way. Also the second property, autonomy of action, is very broad. It is often interpreted that a software agent reacts to changes in its environment that it perceives through sensors. Thus the definition can be applied to any control process or monitoring demon (compare Wooldridge [169]). However, Wooldridge [169] also suggests that it is hard to see any intelligent behaviour in those simple programs. A multitude of similar definitions exist, as for example by Franklin and Graesser [55], Ferber [46], Hayes-Roth [66], Maes [112], Russel and Norvig [146] and Shoham [150]
2. PREREQUISITES

to name only a few. Most of the definitions vary in their focus on particular aspects of agency. However, they all revolve around reoccurring concepts such as autonomy, percept, reasoning and action. They are also informal and therefore ambiguous, especially in comparison to semi-formal definitions of the fundamental building blocks of other software paradigms such as object orientation or services orientation.

2.4.1 Definition of the Weak Notion of Agency

As stressed before, it is crucial to have a clear understanding of the software agent concept in order to apply it to the construction of complex software systems. Therefore, the general definition above is dropped in favour of a more concrete, more structured one. Wooldridge and Jennings [171] propose the weak notion of agency as a fundamental definition. The authors explain that even though there is no commonly accepted definition in use, the weak notion can be understood as a least common denominator across all the different schools and disciplines within the agent community. Frequent reference and strong similarities with other definitions underline a certain popularity and perhaps make this definition the most accepted one in the community. The weak notion is also useful in another aspect. By specifying a small set of distinct properties, it is better structured in comparison to many other definition. Although even this definition is not unambiguous – because of the complex characteristics of its properties – it yet allows a more detailed interpretation. The weak notion of agency specifies a software agent to be a software system possessing the following distinct properties as defined in Wooldridge and Jennings [171, p.118]: “Perhaps the most general way in which the term agent is used is to denote a hardware or (more usually) software-based computer system that enjoys the following properties:

- “Autonomy: agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state.

- “Social ability: agents interact with other agents (and possibly humans) via some kind of agent communication language.

- “Reactivity: agents perceive their environment, (which may be the physical world, a user via a graphical user interface, a collection of other agents, the Internet, or
perhaps all of these combined), and respond in a timely fashion to changes that occur in it.

- “Pro-activeness: agents do not simply act in response to their environment. They are able to exhibit goal-directed behaviour by taking initiative.”

The weak notion denotes a minimal – necessary and sufficient – set of properties a software agent exhibits. This minimal set is often extended to meet particular interests or requirements by incorporating further properties such as:

- Rationality
- Learning ability
- Benevolence
- Robustness
- Temporal continuity
- Believable personality
- Adaptability
- Veracity
- Mobility.

A more comprehensive list of additional properties of agency can be found in Georgakarakou and Economides [59].

2.4.2 Interpretation of the Weak Notion of Agency

Given its popularity, the weak notion of agency forms the foundation of this thesis although even this definition is ambiguous due to the complex characteristics and informal specification of its properties. The author argues in favour of a strict interpretation of the weak notion. This interpretation, of course, does not solve the general dilemma of the agent definition. Many other valid definitions and interpretations exist. However, the interpretation presented here conveys the author’s understanding of the notion of
agency with focus on a software-engineering-like definition of software agent components as fundamental building blocks of multiagent systems. This definition is based on the assumption that all four properties of the weak notion are compulsory; that is, they must be exhibited by every software agent. Accordingly, an interpretation of each of the four properties is outlined below.

**Interpretation of the Autonomy Property**

Autonomy refers to a software agent’s independent self-contained nature. A software agent operates without direct external manipulation whilst having exclusive control over its internal state. A software agent is an independent problem-solver. It perceives its environment ‘at will’, reasons autonomously about gathered information and performs actions accordingly with regard to its goals. Autonomy has got the following implications.

- **High level of abstraction**
  In order to prevent external interference a software agent constitutes a component type that does not publicly expose its functionality for direct external manipulation. A software agent offers a means to other software agents for exchanging messages. Thus a software agent comprises two elements: a well-defined public interface for handling incoming and outgoing messages and an internal core implementation encapsulating a software agent’s behaviour and state. The strict separation between interface and implementation results in a high degree of information hiding.

- **High degree of loose coupling**
  Autonomy implies very low coupling among software agents. Interaction between software agents is essentially reduced to the exchange of asynchronous messages. The abstraction between software agent interface and implementation reduces direct logical dependencies of one software agent on actions and state of another. Asynchrony of messaging reduces temporal binding and thus further reduces coupling. In fact, the coupling is so loose that functional dependencies need to be agreed upon prior to execution. Agreement is reached through communication based on specific interaction protocols.
2.4. NOTION OF AGENCY

• Asynchronous messaging
Software agents interact by exchanging messages. Autonomy is ensured with asynchronous messaging, leaving the full degree of freedom to a software agent as to if, when and how to process incoming messages. Thus software agents possess the intrinsic characteristic of being able to reject or refuse to reply to incoming messages. Each software agent maintains a mailbox over which it exhibits full control. A software agent may also decide to close its mailbox to specific or even all other software agents for any amount of time.

• Message timeouts
A message timeout denotes the maximum amount of time \( \Delta t \) that can elapse between sending an original message and receiving a response. If a message timeout has expired, a software agent is free to pursue alternative actions in order to achieve a goal. Message timeouts are a necessary means to enforce full autonomy of a software agent during conversation with other software agents. This is necessary because dependencies emerge between interacting software agents based on the messages exchanged, e.g. if a software agent’s internal state depends on a response to a message it has sent. Full autonomy is only assured if a software agent is free to decide for how long it waits for a response. This is particularly important for special cases such as when a communication partner failed and an agent would wait for response indefinitely. A message timeout also provides information to a responding software agent necessary for it to determine whether a response has been sent on time.

• Private thread of control
A software agent needs to be executed within its own private thread in order to achieve full control over its actions and internal state. The implementing infrastructure, such as agent toolkits, support the deployment of software agents to enforce autonomy. A software agent autonomously controls its internal states such as hibernation, reactivation or termination as well as deliberation and execution of any of its actions. Consequently, a software agent does not directly depend on external control flows.

The interpretation of the autonomy property for a single software agent component also results in several consequences on a system level. Firstly, no software agent can
obtain full control over other software agents, because the autonomy of a software agent restricts direct external manipulation. The implication of this interpretation of the autonomy property is that multiagent systems are partially observable systems. Vice versa, if a software agent allows direct manipulation, it disobeys the autonomy property, and therefore does not depict a software agent compliant with this interpretation of the weak notion of agency.

Secondly, a multiagent system based on this interpretation intrinsically exhibits uncertainty. A software agent cannot clearly determine the next state of a multiagent system from its local perspective because of partial observability and the decentralised character of multiagent systems (due to agent autonomy). Partial observability implies that even though software agents agreed to collaborate, execution of the corresponding operations cannot be enforced by any party. Even where a software agent acts rationally, it may refuse or break agreements due to changes in its knowledgebase or of its goals. A trust and reputation mechanism as in Jøsang et al. [87] may help to remedy the problem partially, in the form that a software agent assigns low scores and avoids interaction with unreliable collaboration partners. Another problem that may interrupt software agent communication is caused by potentially volatile or unreliable network connections. Hence multiagent systems are distributed systems. In this respect, asynchronous messaging contributes to the uncertainty of multiagent systems. Temporal effects may cause disruption or complete failure of interactions. Thus multiagent systems are stochastic as noted by Wooldridge [169]: “...We can sum this situation up formally by saying that environments are in general assumed to be non-deterministic.”

Thirdly, autonomy induces a decentralisation of multiagent systems as remarked in Padgham and Winikoff [130]: “...considering a system consisting of a number of agents, then a consequence of the agents being autonomous is that the system tends to be decentralised.” Decentralisation is a stronger notion than distribution because distribution refers to spatial dispersion of software components in a computer network without constraints on data and control flow. In contrast, decentralisation implies the complete lack of central data and control flow in distributed systems. Decentralised systems form a subclass of distributed systems. Accordingly, multiagent systems are dynamic systems. The decentralisation of decision-making among multiple software agents denies a single
software agent a guarantee that its environment will not change even though it does not perform any actions in its environment itself.

**Interpretation of the Social Ability Property**

Social ability highlights the character of communication among software agents. Software agents do not interact by invoking the publicly accessible operations of other software agents, but by exchanging messages based on shared domain knowledge of a particular application domain. The implications of social ability are:

- **Message orientation**
  
  Agents interact by exchanging messages. A message contains meta information in the form of performatives denoting an action or incentive of a sender agent usually based on Speech–Act theory (Austin [6]) together with the actual message content that represents data elements of the shared domain knowledge. The meta information is necessary for a receiver agent in order to be able to interpret an incoming message, to deliberate and to trigger appropriate actions, e.g. sending a particular response message. Interaction protocols specify possible valid message sequences.

- **Domain knowledge modelling**
  
  Agent orientation does not implicitly encode domain knowledge in interfaces of software agents in the form of publicly accessible operation signatures (identifier, arguments and return parameter). Hence software agents do not expose an API of their internal functionality with well-defined syntax and implicit semantics to other agents. Agent systems are distributed systems of independent problem-solvers. Domain knowledge needs to be explicitly specified in a way that multiple software agents can share it, e.g. in the form of an ontology. Domain knowledge is used to interpret and reason about incoming messages and to manipulate an agent’s internal knowledgebase in a meaningful manner. Thus software agents must agree on the use of a particular ontology before engaging in conversations.

This interpretation of the social ability property results in a number of consequences on a system level. Firstly, social ability as a mandatory property rules out the existence of single-agent systems. (Compare Jennings and Wooldridge [81]: “When adopting an agent-
oriented view of the world, it soon becomes apparent that a single agent is insufficient” and “...It can be argued that there is no such thing as a single agent system; everything involves multiple agents.”). That means that a problem space must be modelled with multiple software agents, each of them compliant with the weak notion of agency according to this interpretation.

Secondly, a multiagent system must be designed on the basis of shared domain knowledge in order to facilitate meaningful communication. The author acknowledges efforts in the area of Semantic Web research with the focus on merging different ontologies. However, this is out of the scope of this thesis. It is assumed that software agents share common understanding which is, in the simplest form, predefined for all software agents of a multiagent system.

Thirdly, based on the explicit modelling of domain knowledge, the semantics of software agents is not implicitly encoded into fixed structures or hierarchies within a multiagent system but rather loosely defined by different agent roles. An agent-oriented view considers more than functional decomposition based on the principle of separation of concerns. It adds social aspects (compare Shoham [150]: “...a programming paradigm promoting a social view of computing, where ‘agents' interact.”). System behaviour emerges through interactions among software agents, increasing flexibility and adaptability of agent-based approaches, but also increasing their complexity compared to purely functional decompositions. The complexity is determined by the knowledge representation of a problem space, in particular how concepts and properties are specified based on discrete or continuous domains (compare Section 2.3).

**Interpretation of the Reactivity Property**

Reactivity defines a software agent’s ability to perceive its environment, to reason about its percept and to react, if necessary, in a timely manner. In general, any type of software component is reactive in one way or another. Thus reactivity is no special property of agency (compare Padgham and Winikoff [130]). However, reactivity does have two particular implications:

- **Message percept**

  As seen, social ability can be interpreted so that problem spaces can only be ad-
equately modelled using an agent-oriented view with multiagent systems. Accordingly, the environment of a software agent is a multiagent system comprising various software agents. A software agent, therefore, perceives its environment solely through the messages it exchanges with other software agents. A software agent may receive additional inputs in its role as a model of a real world entity. However, this is subject to agent-internal design because a software agent is designed to encapsulate external processes and offer them as services to other software agents. In addition, a software agent cannot assume that other software agents perceive the same external stimuli and that these inputs influence their decision-making and acting unless communicated via messages.

• **Deliberation**

Based on the autonomy property, software agents do not simply react to changes in their environment, e.g. in the form of executing a particular operation. Software agents deliberate about a situation and autonomously decide what action to take with regard to their goals and local preferences. Hence a software agent can also decide not to take any action at all (compare Luck et al. [108]: “By contrast, agents have the ability to decide for themselves whether to participate in computational activity, and whether to perform the desired operation. This is the fundamental distinction that marks out agents as distinct by virtue of their autonomy.”).

There are also consequences on the system level. Firstly, software agents need to respond in a timely manner to incoming messages to successfully cope with the dynamics and non-determinism of multiagent systems (as implied by the autonomy property). If a software agent deliberately or accidentally responds too late, its response most likely will not have the desired effect. Timeliness is ensured based on message timeout information that is shared among communication partners. Secondly, the autonomy property also implies that no software agent can be forced to respond to particular changes. Consequently, reactivity does not ensure determinism on a system level.

**Interpretation of the Proactiveness Property**

Proactiveness describes software agents as software components that deliberate and take initiative for performing actions in order to satisfy their design objectives usually stated
2. PREREQUISITES

in the form of a set of goals (compare Ferber [46]: “The concept of action, which is funda-
mental for multiagent systems, is based on the fact that the agents carry out actions which
are going to modify the agents’ environment, and thus their future decision making.”). 
Proactiveness has the following implications:

- **Goal orientation**
  A software agent acts in order to satisfy its design objectives. Given its autonomy,
a software agent has a certain degree of independence to decide how to achieve its
goals. If necessary, a software agent takes initiative to proactively trigger interac-
tions with other software agents. A software agent is endowed with reasoning and
decision-making capabilities in order to support the decision process, to assess its
internal state and situation of its environment and to derive appropriate sequences
of actions.

- **Inherent flexibility**
  Software agents exhibit a high degree of flexibility. Software agents are programmed
with a set of different plans. Each of these plans can be used to achieve a design
goal or parts of it. During processing, a software agent chooses a plan according
to its current goal and its perception of its environment. A software agent may
decide to use a different plan in case changes occur in its environment or its goal
changes (compare Luck et al. [108]: “It is this autonomy that is also responsible
for providing the flexibility that is needed for open and dynamic environments.
If behavior is predetermined and is guaranteed when invoked, then the ability to
provide flexible responses in the light of changing circumstances is severely curtailed,
if not ruled out entirely.”).

- **Active components**
  Software agents are active components. Proactiveness separates the agent paradigm
from traditional programming paradigms. Software agents are not only passive
components. Proactiveness is an intrinsic property of the notion of agency implying
that all software agents have the inherent capability to get active during processing
at least at some stage. A software agent becomes active where it reasons that its
goal is still valid but the current environment does not provide the conditions for
achieving the goal. Thus the software agent aims at affecting its environment to create conditions that allow achieving its goal. A software agent also may request information from other software agents in order to help it in its decision-making.

Proactiveness implies a consequence on the system level. Time and effect of a proactively triggered action are unknown at design-time because of uncertainty and non-determinism implied by the autonomy property. Software agents are independent decision-makers with the incentive to achieve their own design objectives. Therefore, if, when and how to interact with its environment and other software agents in particular is subject to runtime decisions made by a software agent. The effect of actions based on these decisions depends on the situation in the software agent’s environment over which a software agent does not have full control. Hence actions may be successful but can fail as well.

2.4.3 Multiagent Systems and Open Environments

Wooldridge [169] notes that one reason for the enormous growth in the multiagent community is driven “at least in part by the belief that agents are an appropriate software paradigm through which to exploit the possibilities presented by massive open distributed systems – such as the Internet.” In fact, many authoritative publications on the topic of multiagent systems propose open uncertain environments as the main application domain for agent-oriented solutions, e.g. d’Inverno and Luck [36], Ferber [46], Luck et al. [108] and Padgham and Winikoff [130]. This thesis follows that argument.

The presented interpretation of the weak notion of agency imposes properties and characteristics not only for single software agents but also for multiagent systems. As highlighted above, the properties of the weak notion of agency impose partial observability, non-determinism, dynamics and high complexity on software systems built solely of software agents. Thus a multiagent system based on this interpretation of the weak notion of agency constitutes a good match with the characteristics of open environments as described in Wooldridge [169]. This matching between problem space characteristics and multiagent system properties suggests the existence of agent-based solutions that are suitable for tackling the complexity of open environments. This hypothesis forms the foundation of this thesis.

Therefore, a multiagent system in this thesis is defined as a software system comprising
of two or more independent actors in the form of software agents. The software agents comply with the weak notion of agency. Consequently, respective multiagent systems have the following properties, as outlined in Jennings et al. [83, p.11]: “A [multiagent system] MAS can be defined as a loosely coupled network of problem-solvers that work together to solve problems that are beyond the individual capabilities or knowledge of each problem-solver. These problem-solvers – agents – are autonomous and may be heterogeneous in nature. The characteristics of MAS are:

- “Each agent has incomplete information, or capabilities for solving the problem, thus each agent has a limited viewpoint;
- “There is no global system control;
- “Data is decentralized;
- “Computation is asynchronous.”

In summary, multiagent systems exhibit the characteristics of open environments. Moreover, the activities of a service composition process can be represented by interactions among software agents. As a result, multiagent systems facilitate the modelling of dynamic service composition approaches in open environments as agent-based collaborative problem-solving.

2.5 Agent-Oriented Software Engineering

One major goal of the agent community is the transition of agent-oriented methods from research topics into mainstream industry solutions. Agent-oriented software engineering constitutes according to Jennings [79] the methodology for modelling, designing and implementing software based on the essential key concepts of agent orientation: agent, interaction and organisation. A major argument in Luck et al. [108] and Jennings [79] for adopting an agent-oriented approach is based on the claim that agent orientation delineates a different and higher level of abstraction that is particularly useful when developing complex software systems. Subsequently, Odell [129], Jennings [78] and Jennings and Wooldridge [81] argue that the agent paradigm represents the natural successor of object orientation in the general evolution of programming paradigms in the last 50 years.
High expectations of agent technology as a mainstream software engineering paradigm are documented by Wooldridge and Jennings [171]. They quote Sargent’s [148] prediction from 1992 that agent technology would be the next break-through technology for software development. Similarly, Houlder [71] is cited to have estimated in 1994 that revenue from agent-oriented software would be 3.5 billion US dollars in 2000.

Even though the agent-oriented paradigm found currency in the vast research output of agent platforms, specification and verification languages for agents and interaction protocols, development methodologies, CASE tools and various standards for defining knowledge representations and interaction protocols, little of the ambitious goal seems to have been realised yet. Even without quantitative proof, a lack of wide-spread proliferation of agent-oriented methods in industry in comparison to ‘conventional’ technologies appears to be evident as Luck et al. [108] concluded in 2004: “Having been the subject of intensive research activity for more than a decade, agent technology has still not met with broad acceptance in industrial settings (despite some encouraging success stories).”.

Hendler [68] raised the question about the industrial application of intelligent agents in 2007 and concurred: “There has been much research and talk about intelligent agents, but few real-world implementations”. Hendler remarked that even though key research topics of the agent community concerning interoperability and knowledge engineering are now manifested in standards such as SOAP, WSDL and OWL-S in mainstream Web services and Semantic Web services technology, the agent paradigm itself seems not to have made its way into industry on a large scale. Reasons for failing to do so are identified by Luck et al. [108, ch.4.1]: “We believe that three characteristics of industrial development have so far prevented wider adoption of agent technology:

1. “The scope of industrial projects is much larger than typical research efforts.

2. “The skills of developers are focused on established technologies, as opposed to leading-edge methods and programming languages.

3. “The use of advanced technologies is not part of the success criteria of a project.”

The author believes Luck’s argumentation covers only one side of the issue. On the other side, there are still many open questions within the agent community preventing the agent paradigm from becoming mature enough for industry-scale software development.
2. PREREQUISITES

Hendler’s question on its own, for example, is technically difficult to answer because there is still no mutual agreement on what constitutes a software agent (compare Section 2.4).

In the author’s opinion, it is not possible to formulate a uniform software engineering methodology for the design and implementation of agent-oriented software systems if the fundamental building block of the agent paradigm cannot be specified in a uniform and universal fashion. In this context, software engineering is defined, for example, by the IEEE [76] as “the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software”, implying the existence of a clearly identified fundamental building block to be absolutely essential. The vast diversity of available approaches to agent-based software development is confusing and hinders a clear understanding of the characteristics of agent-based software components and systems and a clear separation from other software engineering paradigms. In particular, it prevents a clear disclosure of pros and cons of the agent-oriented software engineering paradigm. But that is exactly the question to be answered.

Thus the general task of agent-oriented software engineering is to develop a paradigm that delineates distinctive key features compared to other paradigms, such as object orientation, in order to establish agent orientation as a universal self-contained paradigm for the design and implementation of distributed systems, as remarked in Jennings et al. [83]: “...we therefore expect an agent-based system to be both designed and implemented in terms of agents.”

The main focus of this thesis is on the development of an agent-based approach for dynamic service composition in open environments. A second aspect is to investigate the impact of the presented interpretation of the notion of agency when applied as the basis of a uniform and universal agent-oriented paradigm: its impact on design and implementation of a concrete software system as well as the pros and cons of an agent-oriented paradigm, particularly in the absence of an agreed universal agent definition.

To those ends, the author’s understanding of the core properties of an agent-oriented software engineering paradigm is outlined below. It is not the aim to create a complete design methodology. The next two sections merely describe the way the fundamental building blocks of agent orientation are applied for building complex software systems. The properties of the notion of agency as described in Section 2.4.1 form the foundation
2.5. AGENT-ORIENTED SOFTWARE ENGINEERING

of the component and system level.

2.5.1 Component Level Definition

A software agent is defined as a composite software component. It is depicted in Figure 2.4. A software agent operates through ports in order to interact with its environment. It provides an interface \textit{IAddMail} on the \textit{InPort} for other software agents to send messages to it. This interface is basic; it provides exactly one method for adding a message to the software agent’s mailbox. A software agent also defines a required interface on the \textit{OutPort} which it uses to send out messages to other software agents.

![UML specification of a software agent component.](image)

\textbf{Figure 2.4: UML specification of a software agent component.}

Only the external communication of a software agent that is relevant for interacting with other software agents is modelled. Every additional dependency or interaction with software components other than software agents or human users are outside the scope of a multiagent system. It is assumed that a software agent encapsulates additional external relationships and interactions that are provided as services and communicated with messages to other software agents.

Internally, a software agent comprises two elements: a passive mailbox component and an active component that contains the internal state and behaviour of a software agent in a separated thread of control. The mailbox implements the provided interface \textit{IAddMail} and communicates with the two external ports according to whether an incoming or outgoing message needs to be handled. It further implements a second interface \textit{IMail}
that is used by the core agent implementation in order to process and send messages. The IMail interface provides methods for opening and closing the agent’s mailbox, for checking the mailbox status, for fetching available messages and for sending messages to other software agents.

It is assumed that every single software agent complies with this structural definition (syntactical definition). It is further asserted that every software agent complies with this interpretation of the weak notion of agency (semantic definition), in particular that software agents exhibit all four properties outlined by Jennings et al. [83]: “However, we believe that it is the presence of all the attributes in a single software entity that provides the power of the agent paradigm and which distinguishes agent systems from related software paradigms – such as object-oriented systems, distributed systems, and expert systems”.

2.5.2 System Level Definition

After the fundamental building blocks are specified, it is now defined how they are used in order to establish a uniform and universal software engineering paradigm. Uniform refers to the internal uniformity of a multiagent system with regard to the properties of the software agents used for modelling elements of a problem space. Hence it is asserted that all software agents comply with the syntactical and semantic definition of a software agent as outlined above. Universal refers to the application of the agent paradigm to model a problem space in an agent-based fashion as Jennings et al. [83] explain: “By an agent-based system, we mean one in which the key abstraction used is that of an agent.”. Accordingly, a multiagent system comprises only software agents.

This approach to an agent-oriented paradigm requires a problem space to be modelled and implemented solely with software agents. Thus the resulting multiagent system is a software system of actors. Consequently, the universality of multiagent-based approaches is limited because firstly, not every problem space comprises merely actors, and secondly, the design and implementation of low-level components and algorithms below a subsystem level are too fine-grained for agent orientation and simply cannot be modelled on the high abstraction level of the agent paradigm (see Jennings and Wooldridge [82], Wooldridge and Jennings [170]).
Issues with the design and implementation of multiagent systems according to the presented interpretation of the weak notion of agency and its subsequent application as a uniform software engineering paradigm are presented in Chapter 10.

2.5.3 Agent Orientation and Service-Oriented Paradigm

Foster et al. [54] identify synergies between agent-oriented and service-oriented paradigms. According to Foster et al., agent orientation is focussed on flexible dynamic decision-making and goal-oriented problem-solving, whereas service orientation is concerned with defining stable infrastructures and standards to support seamless interoperability and integration. Foster et al. conclude that both paradigms are complementary and their combination/integration offers great potential for developing distributed systems.

The author argues that both paradigms do not only have complementary scopes but also share fundamental characteristics. SOC revolves around the service notion, service-oriented design principles and SOA with the aim of supporting strategic goals and benefits associated with SOC (Erl [41]). Although agent orientation is centred on software agents as the key abstraction concept, it realises service-oriented design principles (compare Section 2.1):

- **Standardisation of service contracts**
  Software agents expose their contracts based on commonly agreed ontological specifications of domain knowledge of a problem domain.

- **Abstraction**
  Software agents reveal very small interfaces. Software agents exchange messages at the knowledge level – a higher level of interaction – and therefore offer each other only the opportunity to add messages to a message queue instead of syntactically and semantically defined operations. Messages must be interpreted. Software agents fully abstract their interfaces from an underlying implementation.

- **Loose coupling**
  Software agents are independent problem-solvers. The autonomy of software agents implies a high degree of loose coupling. Dependencies and relationships between
software agents emerge and dissolve dynamically at runtime based on mutual agreement. Every single software agent has full control over its involvement in any type of computation.

- **Reusability**
  Abstraction and loose coupling between software agents provide a good base for reusability. In addition, software agents exhibit goal-oriented behaviour with the intrinsic ability to adapt to changes. They are endowed with multiple plans for achieving a particular goal (design objective). As such software agents can be reused in similar problem situations for flexibly solving problems with respect to their design objectives.

- **Autonomy**
  Software agents are fully autonomous as described in Section 2.4.2.

- **Discoverability**
  Software agents reveal meta data about the services they offer in order to support an effective service discovery.

- **Composability**
  A multiagent system represents a decomposition of a problem space into multiple interacting autonomous problem-solvers. Software agents collaborate and form coalitions in order to fulfil complex tasks. Coalition formation is a form of dynamic composition brought about by aggregating services of different software agents into a composition based on local autonomous decision-making.

In fact, even though the agent paradigm is not explicitly based on the service notion, software agents model and encapsulate services, however, in a different - proactive – fashion according to organisational decomposition (Jennings [78] and Zambonelli et al. [177]). The compliance of agent orientation with service-oriented design principles and the specific characteristics of the notion of agency directly support agent-based approaches in attaining several of the goals and benefits of service orientation, such as increased interoperability, federation and business and technology alignment (compare Erl [41]).

An SOA model incorporates various technologies, products and standards according
to Erl [41]. Software agents can be seen as one specific technology to implement an SOA model, particularly for extending SOA initiatives beyond corporate boundaries for inter-organisational business process automation. An agent-oriented approach to service composition seems to be natural in this respect because software agents support building software systems in a service-oriented fashion with implicit emphasis on economical and social dependencies and relationships between interacting organisations and their service applications. Agent orientation implicitly guarantees intrinsic autonomy and self-governance of different applications as well as enabling their dynamic federation.

The component structure as illustrated in Figure 2.4 describes a platform-independent model of a software agent component. Accordingly, it is assumed that there are different concrete implementations possible, in particular with service-oriented technologies, e.g. Web services.

2.6 Summary

This chapter sets the scope of the research by introducing fundamental terms and concepts that are essential for this thesis. It provides the foundation for the reader to understand the background of SOC, its key aspect of service composition and the complexity of open environments. Furthermore, the weak notion of agency is presented and interpreted in order to derive the definition of fundamental buildings blocks of an agent-based software engineering paradigm that leverages the similarities and complementary character of agent orientation and service orientation on the one hand and multiagent systems and open environments on the other hand. It is crucial for the reader to understand the relationship between multiagent systems based on the weak notion of agency and the characteristics of open environments, because this relationship is the key to establishing a successful mapping of the problem of dynamic service composition in open environments into multiagent-based collaborative problem-solving. The next chapter presents an extensive literature review in order to analyse how existing agent-based approaches to dynamic service composition cover the mapping issue. It further presents discussion and comparison of related work based on the given definitions and interpretations.
2. PREREQUISITES
Chapter 3

Related Work

This chapter presents a review of relevant publications in the field of dynamic service composition. The literature in the area is extensive, as are exercised techniques. The foci of related contributions cover the whole composite service life-cycle from modelling to operation (compare Figure 2.3). This thesis is focused on the service composition phase. As such, neither preceding nor subsequent phases and activities – such as service modelling, deployment, re-planning/re-composition, etc. – are considered. Furthermore, the scope of this thesis is confined to automated dynamic service composition. As a result, the following review does not include literature with respect to design-time service composition (see Milanovic and Malek [120] for an overview or Agarwal et al. [2], Blanchet et al. [13] and Guo et al. [62] for exemplifying approaches) or online user advise systems (e.g. Chen et al. [29]). The goal of this chapter is to present an analysis of the applicability of existing agent-based approaches to dynamic service composition in the context of open environments. The chapter is organised based on a classification that groups related work according to scope as well as architectural and behavioural similarities. The complexity and diversity of existing approaches affects the given classification. It is by no means complete but rather representative with some categories overlapping. During the review, each category is presented with one or more typical instances and then evaluated with respect to the classes of shortcomings outlined in Section 1.2.
3. RELATED WORK

3.1 Semantic Web Approaches

The class of Semantic Web approaches has its focus on the application of Semantic Web standards and technologies to address the problem of dynamic service composition. The main aim is to enable dynamic service discovery, matchmaking and selection with semantically annotated service descriptions (e.g. based on DAML-S, OWL-S or WSMO). Semantic Web approaches can be divided into service broker and central composer approaches.

3.1.1 Service Brokers

A classical example of a service broker is described in Paolucci et al. [133] and Sycara et al. [159]. A service broker follows the classical mediator pattern described in Gamma et al. [56]. It is, therefore, also referred to in the literature as a service mediator. The use of a service broker in the context of distributed Web-service applications implies a layered system architecture separating service client, service broker and service providers from top to bottom as illustrated in Figure 3.1.

![Diagram of Semantic Web service broker approach](image)

Figure 3.1: UML component diagram of the concept of a Semantic Web service broker approach.

The basic functioning of the service broker approach can be summarised as following. The service broker has access to an internal or third party service repository in which service providers publish their semantically annotated service advertisements. If a service client has a certain task to be performed, it sends a request to the service broker. The
broker processes the request with the help of its knowledgebase and formulates a query for determining services whose capabilities match the requirements of the request. After the broker retrieves a set of service advertisements, it ranks the set and picks the most beneficial advertisement (for general functioning it is not important here on which heuristic the ranking is based). The service broker then requests the respective service provider, obtains the response and relays it back to the service client. The message exchange is also based on Semantic Web notations and technologies.

A service broker is the crucial component in Semantic Web architectures because it provides essential functionality for dynamic service discovery, matchmaking and selection. Moreover, a service broker manages all interactions between a client component and service providers during a service composition process. Service brokering provides flexibility and robustness firstly because of the loose coupling between client and provider side and secondly because of semantic annotations of service capabilities that allow dynamic binding of concrete service instances (Sycara et al. [159]). It also enables interoperability by providing a mechanism to easily support translation between different ontology languages or domains (Sycara et al. [159]).

Further semantic broker approaches are published in Chen et al. [28], Mandell and McIlraith [113] and McIlraith et al. [118]. Chen et al. [28] describes a central broker agent that provides small and resource-limited devices with computing and reasoning capabilities for managing context knowledge that can be obtained from different sources including Semantic Web services. The centralisation of the approach is motivated by the support of small devices to overcome their computational restrictions in a pervasive environment. The approach does not explicitly specify a method for dynamic service composition but describes an infrastructure that utilises the dynamic accumulation of information by linking and coordinating heterogeneous sources into a coherent model. Mandell and McIlraith [113] propose the Semantic Discovery Service that is integrated in a BPEL4WS engine in order to broker between the BPEL execution engine and Semantic Web services. McIlraith et al. [118] outline specialised Web agents that provide an application service to end users based on semantically annotated generic procedures. Web agents contact an agent broker in order to interact with semantic marked-up Web services in order to satisfy a user request and its specific constraints.
3. RELATED WORK

Sycara et al. [159] also discuss a slightly different approach called matchmaker. Matchmakers appear in different forms, as for example a UDDI registry. Semantic Web matchmakers provide functionality to acquire suitable services based on capability matchmaking between problem properties and semantically annotated service capabilities. However, a matchmaker does not manage interactions between client and provider side. It basically provides references to a set of candidate services and leaves the service client in charge of coordinating the interactions with concrete service instances. In this respect, a matchmaker approach does not exhibit centralised data and control flows compared to a service broker.

Service broker approaches have limitations in open environments. They are based on complete knowledge about currently available services that is provided in central repositories. The very nature of a service broker leads to a single point of failure because it centralises data and control flows in the system. It also manifests central decision-making in order to control the discovery, matchmaking and selection of a candidate service without the involvement of service client or service providers. Thus service broker approaches lack locality in the form of local knowledge and preferences, e.g. of service providers. The focus of a broker approach is usually on dynamic service discovery, matchmaking and selection. It is assumed that a process definition exists prior to interactions. However, a service broker approach can be combined with automated planning methods or transformed into a central composer as outlined in the next section.

Limitations also exist with respect to software engineering aspects. The notion of agency is significantly different from the interpretation presented in Section 2.4.2. It usually stands for an entity that acts on behalf of a client or that possesses reasoning capabilities but not exactly all four properties of the weak notion of agency. Service brokers violate the autonomy property (because they are strictly benevolent and always adopt the goal of a requesting client) and proactiveness (since they passively wait for requests). In contrast, the service client may be implemented as a software agent that exhibits full compliance with the weak notion of agency. The results are inconsistent degrees of compliance with the weak notion of agency for different agent types. Finally, the broker approach is not modelled using software agents as a consistent and coherent paradigm. It usually interacts with other Semantic Web component types, e.g. Web
services and UDDI registries.

3.1.2 Central Composers

Central composers extend the concept of service brokers by rudimentary planning techniques. A central composer has access to a repository of pre-defined workflow process templates (Agre and Marinova [3] and Korhonen et al. [96]) or concrete workflow process definitions (Laukkanen and Helin [102]). A central composer is a complex software agent that performs dynamic service composition directly on behalf of a human user.

Korhonen et al. [96] propose a central composer that, upon user request, instantiates a so called master workflow template from so called component workflows. A component workflow represents a DAML-S process model that specifies capabilities and interaction sequence of a particular service. Thus this approach does not compose workflows based on automated planning but rather discovers, matches and selects concrete services from a central repository and assembles their process models into a composite workflow process definition.

Laukkanen and Helin [102] present a similar approach based on DAML-S and WS-BPEL. A workflow composer agent has access to WSDL and DAML-S descriptions of services in a UDDI repository as well as a set of pre-defined workflow process definition. As such, every process definition represents a fixed set of concrete services that is ready for instantiation. Upon user request, the workflow composer agent verifies the availability of services of a chosen workflow process definition. In case of failure, the agent is able to replace the erroneous service with a semantically similar service based on capability matchmaking.

In contrast, Agre and Marinova [3] describe the Infrawebs framework for building Semantic Web Service applications. It comprises a central composer component that is able to iteratively decompose a user-defined goal until a configuration of matching services has been found. User goals are based on goal templates specified in WSMO. A goal description is matched with WSMO-based service advertisements. If no match is found, the goal is decomposed into two subgoals and so on. If one or multiple matches exists, the composer creates a listing and presents it to the user who is left with the final decision as to which services to select and invoke.
3. RELATED WORK

Central composers have the same limitations as service brokers with respect to their adaptability to open environments. This is because they are also fully centralised, are based on complete knowledge and lack the capability to consider local knowledge and the preferences of service providers. In this context, although Agre and Marinova [3] describe their service repository to be distributed in a structured peer-to-peer network, the whole approach still exhibits fully centralised decision-making and control flows. Central composers differ from brokers in their attempt to address service planning as well. Central composers in their majority employ only a single composer agent. The composer exhibits the same shortcomings regarding agent autonomy and proactiveness as outlined for broker approaches. Accordingly, they do not conform with the weak notion of agency.

3.2 Multiagent Systems

Multiagent systems tackle the problem of dynamic service composition by distributing functionality and decision-making over a set of software agents. The aim is to capitalise on the intrinsic flexibility of multiagent communication and coordination. Multiagent-based approaches can be separated into multiagent middleware and agent-based frameworks. Multiagent-based approaches are difficult to classify since many of them are tailored to specific problem domains. In addition, a number of multiagent approaches depend on particular environments such as mobile computing (e.g. Berger et al. [10]) or pervasive computing (e.g. Vallée et al. [163]) to address the problem of dynamic service composition.

3.2.1 Multiagent Middleware

Agent-based middleware approaches form a characteristic middle layer between a client layer and a services layer, as depicted in Figure 3.2. Software agents act in generic middle-agent and proxy-agent roles in order to provide flexible dynamic service discovery, matchmaking and selection. Middle agents act or mediate on behalf of clients. They initiate the formation of team structures according to a client request and based on knowledge of composite service structures. Proxy agents exhibit the functionality of one (e.g. Hassnaoui et al. [65]) or multiple (e.g. Fan et al. [44]) services. The key feature of multiagent middleware is the interactions between cooperating software agents that replace rather
rigid and pre-determined control and data flows of conventional workflow-driven solutions. Multiagent interactions are based on distributed yet centrally controlled decision-making.

Classical middleware system architectures are outlined in Xie and Huang [173] and Yoshimura et al. [175]. Particularly Yoshimura et al. describe deliberate proxy agents that participate in team actions based on local goal-oriented decisions and mediate between the ACL driven communication in the multiagent system and Web services interactions based on WSDL. The former stresses the importance of multiagent interactions to tackle the problem of dynamic service composition; the latter emphasises the role of proxy agents as integrators between the agent layer and the lower services layer.

Figure 3.2: UML component diagram of the concept of a middleware multiagent system approach.

In the two examples so far, agent roles have been fixed. In contrast, Barker and Mann [8], Charif-Djebbar and Sabouret [26, 27], Fan et al. [44], Hassnaoui et al. [65] and Künagas and Matskin [100] present decentralised approaches exhibiting dynamic role adoption amongst a set of equal-righted peers that are embodied by autonomous software agents. All five approaches do not support explicitly dedicated middle-agent roles. On the contrary, a peer that receives a user request adopts the role of a middle agent and initiates
and coordinates team collaboration only if the request is beyond its own capabilities.

Barker and Mann [8], Fan et al. [44], Hassnaoui et al. [65] and Künugas and Matskin [100] realise service discovery by establishing long-term relationships among proxy agents. Barker’s approach is based on a fixed interaction protocol in a scientific grid-computing context whereas Fan and Hassnaoui implement relationship networks (e.g. in the case of Hassnaoui inspired by the human immune system). These networks are updated dynamically after every service composition attempt based on a heuristic that evaluates the efficiency and quality of collaborations. Künugas and Matskin use a Chord-protocol-driven peer-to-peer network. Charif-Djebbar and Sabouret [27] rely on a central UDDI repository. In contrast to the four other approaches, Barker and Mann [8] implement proxy agents with each of them representing one or multiple services capabilities not services. Concrete matching services for a capability are obtained by querying a central UDDI registry.

A more fine-grained modelling of proxy agents can be found in Maamar et al. [109, 111]. This model investigates dynamic service composition based on context information that takes into account service provider resource capacities and status information (Maamar et al. [111]) as well as resource limitations of small devices in mobile and wireless networks (Maamar et al. [109]). In Maamar et al. [111] a composite service agent first interacts with so called master service agents in order to obtain contact with a service agent. Master service agents monitor the resources and statuses of services at a service provider’s site. Service agents represent a concrete service instance. Maamar et al. [109] incorporate a set of middle agents, namely user delegate, service-broker and resource broker agents. User delegate agents meet with the other software agents on a so called meeting platform in order to compose services and to allocate resources for later service execution by collaborating with service broker and resource broker agents. The use of a meeting platform and the separate handling of resource allocation reflect the limited resources available to small mobile devices.

Existing approaches exhibit a number of shortcomings with respect to the requirements of open environments. Complex knowledge about environments is encoded in explicit interaction protocols (e.g. Barker and Mann [8]), hierarchies of agent roles (e.g. Maamar et al. [109]), complex relationship structures (e.g. Fan et al. [44]) or provided with central
repositories (e.g. Charif-Djebbar and Sabouret [27]). Data and control flow is centralised following the top-down call structure: from the top client layer down to the services layer. Even in distributed approaches such as Barker and Mann [8], Charif-Djebbar and Sabouret [26, 27], Fan et al. [44] and Hassnaoui et al. [65], the agent that receives a service request adopts a leader role that imposes clear hierarchical constraints on information flow and decision-making. Proxy agents typically provide service functionality and contribute to decision-making according to the aforementioned constraints. Apart from Maamar et al. [109, 111] no approach models the local preferences of service providers. Xie and Huang [173] and Yoshimura et al. [175] outline basic goal decomposition methods. Küngas and Matskin [100] incorporate theorem-proving based on linear logic and semantic service descriptions. Despite these three examples, service planning is not included in the majority of multiagent middleware approaches. They basically focus on dynamic service discovery, matchmaking and selection based on a set of pre-defined composite service templates.

There are also a number of issues with respect to agent-based software engineering. Most approaches do not specify a definition of the notion of agency. Since proxy agents are left passive, they violate the proactiveness property of the weak notion of agency. Their autonomy is also in question in many cases due to their intrinsic benevolence (except the service master agents in Maamar et al. [111]). In contrast, middle agents, such as the end user agent in Yoshimura et al. [175], are more complex than proxy agents and exhibit more compliance with the weak notion of agency. Consequently, resulting multiagent systems are inconsistent with respect to the notion of agency. In addition, the use of non-agent components such as Web-service-based UDDI registries (e.g. Charif-Djebbar and Sabouret [27]) result in non-compliance with agent orientation as a uniform software paradigm. The strict separation between agent-based middle layers and service layers incurs costly integration efforts, e.g. for running different infrastructures or for the transformation of ACL messages into WSDL messages and vice versa, effecting implementation complexity and execution speed.

### 3.2.2 Agent-based Frameworks

The difference between multiagent middleware approaches and agent-based frameworks for dynamic service composition is effectively the introduction of additional specialised
software agents. These agents are aimed at utilising and facilitating service composition amongst multiple software agents beyond the actor-centric character (software agents represent service consumers and service providers) of previously introduced middle agents and proxy agents. Specialised software agents in general encapsulate functionality to support service discovery, service matchmaking and service selection. Existing approaches are very diverse, there are no uniform naming conventions or notions to follow.

The focus of a number of frameworks is on the interoperability and integration of Semantic Web services for realising inter-organisational distributed workflows. Blake [11] proposes to implement a team of cooperating software agents for cross-organisational workflow composition and orchestration. The team consists of different broker agents that advertise and discover applicable services from UDDI registries. It also includes coordination agents that embody workflow process patterns and exception handling agents that monitor composition progress and trigger actions to dynamically adapt a workflow process to errors and changes. A central non-agent component initialises different software agents. Blake [12] extends the approach with additional agent types, e.g. Workflow Manager Agents, Role Manager Agents, etc. Workflow composition is in both approaches performed based on pre-existing workflow process templates and optimised according to past compositions and executions.

Ermolayev et al. [42] present a similar approach in which service requester agents assign a service provider agent with a composition task. The service provider agent can negotiate the fulfilment of tasks with other service provider agents. The service provider agent is supported by matchmaker and ontology agents. Service discovery is performed via a UDDI/DAML-S registry.

Aberg et al. [1] present an agent framework for integrating compositions of Semantic Web services into a proprietary workflow engine by introducing a so called butler agent. The butler agent manages the integration of external workflow subprocesses that include external Semantic Web services. The butler interacts with a number of different software agents that all register with a central registry. Web service provider agents, for example, advertise this way their OWL-S based service descriptions. A Web service manager agent assists the butler agent in looking up Web services from a central repository. A Web service discovery agent performs matchmaking of Web service provider agents. And a set
of ontology agents provide different domain knowledge representations. The butler agent coordinates interactions with all these different software agents in order to manage the composition of external workflow subprocesses.

Shrivastava et al. [151] propose an approach that allows software agents to start and participate in workflows incorporating an arbitrary collection of different application via CORBA services. By participating in workflows, software agents (wrapped as CORBA services) add flexibility to the service-selection process according to changes in the environment, e.g. the changing availability of particular services.

Agent frameworks are also used in several cases for building end-user applications. Howard and Kerschberg [72] describe an approach for brokering Semantic Web services based on a knowledge framework. A broker agent receives a user request and initiates and manages the composition of necessary Semantic Web services to satisfy the user request. The broker agent is supported by a discovery agent for service lookup from a central UDDI registry, a classification agent that handles domain ontologies, a decomposition agent that decomposes complex services and workflow tasks, preference agents and user agents that represent accumulated long-term and current non-functional requirements respectively and an integration agent that generates a concrete workflow process definition. The framework is based on a central knowledgebase that supports decision-making for the selection of particular services with information of previous composition processes (e.g. quality and reliability of services) and user preferences.

Liao et al. [106] similarly outline a multiagent-based framework in which several different highly specialised software agent types perform and manage dynamic service composition with the aim of achieving self-configuration, self-optimisation, self-healing, etc. The approach mixes agent-services (as some kind of compound services) with atomic-domain-services that get combined into one composite service based on predefined plan templates.

Two other publications describe concrete agent-based application frameworks that perform service composition, e.g. for event organisation (Poggi et al. [138]) and for a travel agent competition (Zou et al. [180]). Both approaches include a number of problem-customised software agents, such as passive service agents, auction agents and directory service agents. They are implemented on the Agentcities FIPA-compliant agent network.
and support the composition of OWL-S annotated Semantic Web services. Because of the FIPA compatibility of Agentcities, both approaches make use of central repositories for looking up software agents and services. In addition, they are based on a strictly hierarchical overlay network which is itself based on a single main-container and a set of sub-containers.

Agent-based frameworks suffer from their central repositories, which create a central point of failure, and are difficult to maintain in open environments. Furthermore, they are based on complete knowledge that must be available in the central repositories in order to enable smooth processing. Due to the structural layering of frameworks, clear dependency hierarchies exist that materialise in central data and control flow: lower layer software agents depend on input from and control of upper layer software agents or even other software components. The frameworks are mainly focused on dynamic service discovery, matchmaking and selection, e.g. in the form of plan templates (Blake [12]) or pre-defined agent types and specifically tailored interaction sequences (Ermolayev et al. [42]).

There are additional issues with respect to software engineering aspects of agent-based frameworks. The clear functional decomposition of a framework into different complementing functional software agents reduces their conformity with the weak notion of agency on the component and the system level. Matchmaker agents and ontology agents as in Ermolayev et al. [42] or Web service manager agents and Web service discovery agents as in Aberg et al. [1] are limited in their autonomy and proactiveness. Firstly they are essential for the overall functioning of a framework (and strictly benevolent) and secondly they represent passive functional units that respond to queries in a strict client–server fashion. As such, there is no multi-threaded concurrent processing observable in most approaches. On the other hand, Aberg et al. [1] specify a butler agent that initiates and completely controls service composition. Thus it is not limited in its autonomy by other agents leading to an inconsistent use of the notion to model different agent types of the same multiagent system. Frameworks comprise not only software agents but incorporate other software components such as UDDI repositories (e.g. Blake [11]) or workflow engines (e.g. Shrivastava et al. [151]). Thus they do not apply agent orientation as a uniform and universal software paradigm.
3.3 Automated Planning and Formal Methods

A substantial amount of research has been dedicated to the application of AI planning methods for automated service composition or, more precisely, automated service planning (see Section 2.2). Klusch [91] recapitulates that service planning can be interpreted as a classical state-transition-based planning problem \((A, I, G)\) in which a client-defined request corresponds to the goal state \(G\) that is achieved through the concatenation and execution of a sequence of actions \(A\) starting from a well-defined initial system state \(I\). An action represents a valid state transition in the planning system.

Services correspond with actions in the domain of automated service planning. They specify preconditions and effects of state transitions based on semantically rich annotations such as DAML-S and OWL-S service profiles or given with BPEL process models. These service definitions need to be transformed into a declarative formal representation, such as the Planning Domain Definition Language (PDDL), in order to enable the application of formal logics.

Classical AI planning is based on assumptions such as complete observability, atomic time and full determinism that are not fully applicable in the service composition domain in general and in open environments in particular. Thus service planning approaches for dynamic service planning employ extended and augmented classical planning methods, such as a derivate STRIPS planner in Martínez and Lеспérance [116] or partial-ordered planning in Peer [136]. Besides classical planning, there are a number of additional representation formalisms in use, such as HTN planning (e.g. Jianhong et al. [85], Sirin et al. [152], Wu et al. [172]), hybrid HTN and forward-chaining planning (Klusch and Gerber [92]), linear logic (e.g. Rao and Su [143], Küngas [99]), situation calculus (e.g. McIllraith and Son [117], Nariai et al. [125], Sohrabi et al. [156]) and symbolic model-checking (e.g. Pistore et al. [137], Traverso and Pistore [161]).

The literature in the field is extensive. Comprehensive reviews can be found in Klusch [91], Dustdar and Schreiner [39] or Rao and Su [144]. Alternative artificial intelligence techniques are available to cover parts or the whole service composition process, e.g. formal methods such as Petri nets, constraint satisfaction, automated learning techniques such as case-based reasoning or procedural reasoning based on the belief–desire–intention model. Representative examples are outlined in the subsections below. It is not the aim
of this section to detail the principles of automated planning. Rather, the scope of this section is focused on the characteristics of automated planning approaches with respect to the requirements of open environments.

Automated planning approach are similar in their characteristics to central composers (Section 3.1.2), with the difference that they are focused on the transformation of semantic descriptions into logical sentences and fully automated planning instead of enriching service descriptions for automated service discovery and matchmaking. Automated planners are centralised software systems as shown in Figure 3.3. They acquire necessary information from central knowledgebases or service registries. In their majority, they require complete knowledge about the availability and type of all services in one central point, even though a number of automated planners (e.g. Martínez and Lespérance [116], Peer [136], Sirin et al. [152] or McIlraith and Son [117]) can handle uncertainty in the form of incomplete information about the initial state or non-deterministic effects of services. The application of central planning components implies central decision-making as well as centralised data and control flow (if necessary after all). Distributed planning (e.g. des-Jardins and Wolverton [35]) may be used to ease the centralisation. However, distributed planners are in general not robust against the failure of one or more planning agents due to a complementary partition of the search space. Planning is based on static semantic service advertisements. Thus service providers have no means to actively influence the

Figure 3.3: UML component diagram of the concept of an automated planner approach.
planning process with respect to their local preferences. In general, automated planning systems do not integrate service planning with service discovery, matchmaking and selection into a single framework assuming that a separate matchmaker performs it (with the exception of Sohrabi et al. [156]).

Most automated planning approaches do not refer to multiagent systems. A planning agent is situated in a Semantic Web environment. The use of the term software agent is not identical with the interpretation of the weak notion of agency as presented in Section 2.4.2. It merely denotes a software entity with planning capabilities. Accordingly, automated planners do not conform with the weak notion of agency. They must be considered non-compliant with a uniform agent-oriented software paradigm.

3.3.1 Petri Nets

Hamadi and Benatallah [63] outline a Petri net process algebra aimed at the support of automated definition and verification of composite services. Likewise Narayanan and McIlraith [124] centre on automated reasoning for specifying and validating composite services. In this approach, DAML-S service process models are formulated as actions in situation calculus. Based on these formal logic descriptions of services, a composition agent creates a Petri net for automatic instantiation and dynamic verification of correctness and performance of a composite service. The two approaches are based on the assumption of simplified atomic service process models. In contrast, Yi and Kochut [174] implement a Coloured Petri net in order to realise the creation of composite services from component services with complex service process models that employ a sequence of mandatory interactions with a single service instance.

3.3.2 Constraint Satisfaction

A different way to tackle the service composition problem is to apply constraint satisfaction techniques. Hassine et al. [64] argue that a service composition process corresponds to a constraint optimisation problem. A composite service request including client-defined preferences defines hard constraints that need to be balanced with the local constraints of particular services, e.g. input and output parameter domains and values. The approach comprises three agent types: interface agent, abstract service agent and information agent.
An interface agent handles interactions with the requesting client. The abstract service agents form a hierarchy according to a composite service layout. Each abstract service agent represents a particular service type and manages interactions with all associated concrete services. Thus the abstract service agents perform the actual constraint satisfaction algorithm and also enable constraint optimisation with the involvement of the client in order to determine the optimal set of concrete services for a request. Information agents support abstract service agents with information about concrete services from local registries upon request. Another constraint-satisfaction-based approach is outlined in Cao et al. [24]. Both, Cao et al. [24] and Hassine et al. [64] correspond in their characterisation of multiagent middleware (Section 3.2.1) in contrast to all other planning and formal methods approaches.

3.3.3 Case-based Reasoning

Case-based reasoning or other learning techniques are also attractive means for automating and improving the performance of composite services based on reasoning about aggregated historical information. Limthanmaphon and Zhang [107] propose a case-based reasoning approach for service discovery, matchmaking and selection based on a similarity relation between a current service request and previous ones. A case-based reasoning engine determines a set of component service types and feeds them to a service composer. Information about services is acquired from a central UDDI registry. The reasoner possesses a case base containing component services.

3.3.4 Procedural Reasoning

Belief–desire–intention is a particular agent-based architectural and behavioural model that specifies knowledge modelling, reasoning and action selection of a software agent based on a small set of plans and constant feedback from the software agent’s environment. Kim and Jin [90] propose a single composition agent that takes a service request as input and makes it its major goal (its desire). The composition agent reasons based on its knowledge about its environment and internal state (its beliefs) and selects a sequence of actions (its intention) in the form of a plan accordingly. A chosen intention reflects the composition agent’s deliberate course of action aimed at achieving its desire. The
composition agent may decide to pursue another plan at any time because of changes in its beliefs. This flexibility is a key feature of the belief–desire–intention framework. With respect to service composition, the composition agent’s beliefs are embodied in OWL-S based process models of available services, stored in a central repository. The composition agent’s plans comprise tasks that are interpreted as generic service type descriptions. Once the composition agent has decided on a course of action, it performs service discovery, matchmaking and selection for refilling all tasks with concrete service instances.

3.4 Other Approaches

This section presents four additional methods for dynamic service composition. Each overlaps with preceding categories but also exhibit distinct features that make it difficult to assign them to one specific category.

3.4.1 Negotiations and Auctions

Negotiation methods support multiagent-based service discovery and selection but not automated service planning. Software agents interact constrained by market mechanisms in order to negotiate or auction composite services.

Cao et al. [24] present a negotiation approach in a multiagent setup. A central composition agent instantiates a set of service agents depending on a pre-defined workflow process model. Each service agent represents a particular service type. Thus a service agent manages interactions with relevant services. The negotiation process commences with each service agent choosing an initial concrete service in the attempt to optimise client-defined preferences. After allocation of a service, every service agent multicasts a service description of its choice to all other service agents. Service agents respond to an allocation announcement in case it interferes with their local allocations. They request the sender to change according to their preferences. If a service request agent receives change requests, it discovers and selects an alternative candidate service from its local repository that better suits collaborating service agents, if available. The group negotiations continue until a valid set of services has been allocated or the combinatorial search space spanned by all available services is exhausted. The approach is implemented using constraint satisfaction problem-solving techniques.
Preist et al. [140] explore technical and economical aspects of principal composition agents that simultaneously participate in auctions in two different markets. Composition agents bid in reverse English auctions in a client market in order to secure the right to provide a composite service for an agreed price. That means that clients publish requests for composite services in an auction and the composition agent with the lowest bid wins. Simultaneously, composition agents participate in English auctions in a provider market for obtaining a necessary and sufficient set of concrete services to solve a composite service request. The selection of candidate services is based on a local plan base from which a composition agent chooses one or more candidate plans that match a composite service request. The fundamental decision-making of the composition agent is to rank candidate plans according to the status of the simultaneous auctions in both markets. Thus the system is highly dynamic and may change at any time due to bids of different composition agents. The approach’s system design resembles service brokers (Section 3.1.1). The existence of multiple composition agents partially relaxes the centralisation of the approach in comparison to original service brokers.

A combination of negotiation and auction techniques is described in a multiagent travel planning approach in Hsu et al. [73]. A travel agent publishes a travel request comprising start and end locations as well as a partially ordered set of stopover locations to all service agents that are registered with a central UDDI registry. Interested service agents return so called qualified track segments that connect any two user-defined locations. The travel agent performs a shortest path algorithm on a graph based on available track segments in order to identify one or more minimal itineraries. The itineraries form the basis of the upcoming interleaved auction and negotiation activities. The travel agent informs the service agents about the obtained itineraries and invites them to form coalitions (one per itinerary) in order to compete in a reverse English auction. Each coalition of service agents determines its next bid depending on the outcome of a coalition-internal negotiation. Service agents negotiate to optimise their local preferences. Coalitions failing to produce a new bid in a specified amount of time take no further part in the auction.

Negotiation and auction approaches display a number of similar and as many varying characteristics. The knowledge about available services, for example, must be complete and is stored in a central repository in Hsu et al. [73] and is distributed amongst several
3.4. OTHER APPROACHES

service agents in Cao et al. [24] whereas it is decentralised and incomplete in Preist et al. [140] where service providers actively advertise their services at runtime in an ad hoc fashion. Data and control flow is centralised in Hsu et al. [73] by a central travel agent but semi-decentralised around a central composition agent in Preist et al. [140] and fully decentralised with multicast messaging in Cao et al. [24]. The decision-making is decentralised amongst negotiation parties in all three approaches, with the limitation of it being fully cooperative and ignorant of local service provider preferences in Cao et al. [24]. Service agents in Preist et al. [140] and Hsu et al. [73] actively consider their local preferences during negotiation and auction activities. Service planning, based on a travel domain-specific shortest path algorithm, is only considered in Hsu et al. [73]. Hsu et al. [73] also note the formation of coalitions, which is, however, directly implied by the central shortest path algorithm of the travel agent.

In general, all negotiation-based or auction-based approaches show high compliance with the weak notion of agency on component, system and paradigm levels. Only the use of a UDDI registry in Hsu et al. [73] limits the conformity with a uniform agent-oriented software paradigm.

3.4.2 Declarative Specification Languages

Formal specification languages such as KIF, PDDL or the Z-specification can be used to describe service capabilities and composite service structures and constraints in terms of logical formulas that are fed into an automated reasoner or theorem prover in order to instantiate a composite service model with concrete service instances.

Ambroszkiewicz [4] specifies a declarative specification language called Entish with the aim of uniformly defining information exchange and the internal state of software agents and services based on logical sentences. Basically, Entish specifies preconditions and effects of actions performed by agents or services. Due to the unified approach for modelling software agents and Web services alike, Entish is split into two layers to separate the description and composition language from concrete binding and transport protocols. Ambroszkiewicz advocates a central composer approach. A single software agent acts on behalf of a user and performs service discovery, matchmaking and selection of services. It retrieves information about available services from so called infoServices
3. RELATED WORK

and attempts in a next step to obtain binding commitment of concrete services. The composition process is guided by the software agent’s intentions, being a sequence of tasks to be automatically matched with service capabilities. A description of the composition protocol is out of the scope of this thesis. It is elaborated in Ambroszkiewicz [4]. But it is necessary to note that the protocol is restricted to service publication, service discovery and workflow formation conversations exclusively between either services and infoServices, software agent and infoServices and software agent and services. Entish does not support multiagent conversations or service planning activities.

Letia et al. [104] describe the application of the Z-specification for defining composite service models and service capabilities of software agents in a multiagent system in the form of an agent-based framework. The framework incorporates a proxy agent for every participating Web service. A proxy agent encapsulates interactions with its associated Web service and also provides a description of the services’s capabilities. The user is represented by a so called Z-agent that has a set of Z-based composite service specifications at its disposal. After a user has selected a composite service, the Z-agent performs service discovery, matchmaking and selection. The Z-agent is supported by a number of infrastructure agents: a reliable agent utilises optimal selecting of concrete services, a WSDL2Z agent provides transformation services for interactions between proxy agents and their associated Web services and domain agents keep state variables and invariants of specific application domains such as travel planning.

Both approaches exhibit the specific characteristics of their associated categories as described above with respect to their limitations in open environments and agent-oriented software engineering aspects. However, Ambroszkiewicz [4] explicitly models message timeouts as an essential mechanism for ensuring correctness and efficiency of the service composition process in contrast to the majority of agent-based service composition approaches.

3.4.3 Compositional Agents

Compositional agents provide a form of integration between agent technology and Semantic Web services different from complex multiagent systems and layered approaches. Compositional agents are complex software components that model Web service function-
alility as agent-internal behaviours. Behaviours are implemented by semantically annotated software components that effectively proxy concrete external Web services (see Figure 3.4). Hence Semantic Web services are integrated into solitary software agents. Compositional agents represent composite services whereas Web services provide atomic service functionality agnostic to any particular composite service. The key feature of compositional agents is their runtime creation (Buhler and Vidal [23]) or dynamic configuration (Amor et al. [5]) to endow them dynamically with specific behavioural components for enabling the execution of a user-selected workflow process model. The dynamic nature of compositional agents increases the flexibility of the service composition process to handle changes in an environment.

![UML component diagram of the concept of a compositional agent approach.](image)

Amor et al. [5] present an agent architecture comprising a central mediator component that interconnects a knowledgebase, a coordinator and behaviour components. Behaviour components can provide internal functionality or information of external Web services in the form of DAML-S process models. The mediator acts as a service broker and dynamically discovers, matches and selects behavioural components with respect to the agent’s internal state (stored in the knowledgebase) and the coordinator’s actions. Hence the component-based approach and semantic annotations allow for dynamic selection and binding of concrete services. The compositional agents’ complexity can be easily extended by plugging in new Web service components. Similar approaches are described
3. RELATED WORK


Czajkowski et al. [32] describe an approach that incorporates automated planning and uses OWL-S/RDF ontologies for describing goals and specify service capabilities. An on-demand planning system analyses a user-defined goal and looks up relevant services from a central repository based on an input and output parameter matching. The service descriptions are then used by an automated backward-chaining planner for creating a generic workflow process model called agent blueprint. An executable software agent is finally instantiated after discovering and selecting proper services according to the service type descriptions of the blueprint.

In general, compositional agents display shortcomings due to their centralised nature such as central data and control flows and central decision-making. In addition, they rely on central repositories that provide complete knowledge about available services. Compositional agents are focused on service selection with the exception of those described by Czajkowski et al. [32]. Being centred on single-agent entities, compositional agents do not conform with the weak notion of agency.

3.4.4 Mobile Agents

Mobile agents typify a highly specialised agent paradigm for programming distributed systems by accenting mobility as an intrinsic and most important property. The foremost concern of mobile agents is the optimisation of network load by migrating code instead of moving large amounts of data. Despite the specialisation, mobile agents provide means for dynamic service discovery, matchmaking and selection in distributed environments.

Lee et al. [103] introduce a distributed peer-to-peer network that is roaming by mobile agents. Every peer in the network hosts an agent platform for mobile agents to migrate to and reside on. Agent platforms also provide access to local Web services. Service composition commences when a user configures a mobile agent with a goal and a generic workflow process model. The mobile agent starts processing by interacting with a peer-
3.4. OTHER APPROACHES

to-peer discovery service for dynamically looking up, matching and selecting a service for the first task. It then migrates to the corresponding remote host and instantiates and invokes the service locally. If the service is not available or fails, the mobile agent is able to discover and select a replacement service. Hence a mobile agent conducts reactive planning. Service composition finishes after the execution of the final service in the workflow process model.

Zahreddine and Mahmoud [176] outline a similar but extended approach that enables a mobile agent to handle more complex workflow control structures such as parallel split/join and other conditional execution patterns. In case of a split, a mobile agent multiplies itself into as many clones as necessary to cover all concurrent threads. The clones operate autonomously and simultaneously before they reunite in the join element and are destroyed by the original mobile agent.

Maamar et al. [110] propose the application of two interacting mobile agents for dynamic service discovery and selection. These mobile agents operate in a multiagent system that is populated by various stationary provider agents that proxy the access and mediate interactions with external Web services. The approach operates like the above examples with the exception that it separates service discovery and lookup from service execution into different mobile agents. A user agent acts on behalf of a user. It keeps the user-defined inputs and the selected workflow process model. For service composition, the user agent assigns a delegate agent to discover, match and select appropriate services for a given workflow task. The delegate returns a service description based on whether the user agent decides on remote invocation or migration and local execution of the corresponding service. Service composition and execution are interleaved because the delegate agent is setup to perform service composition one step ahead of the user agent. Thus if the user agent executes a task at position $n$ in the workflow, the delegate agent already composes the next task $n+1$ and so on.

Mobile agent approaches distribute knowledge and perform dynamic service selection in an on-demand manner. This enables increased adaptation to dynamic open environments. However, they suffer from central decision-making, centralised data and control flows and they exhibit a lack of automated planning. In addition, mobile agents raise severe security issues that make the technology unlikely to be adopted in industry due
3. RELATED WORK

to the intrinsic risk of malicious behaviour of both the mobile agents and the agent platforms. The outlined approaches do not fully comply with the weak notion of agency. The first two because they model single-agent systems. The latter approach is in conflict on the component and system levels due to its inconsistent use of agent types with differing fundamental properties (e.g. the differing proactiveness and autonomy of mobile agents compared with provider agents).

3.5 Summary

This chapter presents a review of related work organised in categories. Each of the categories groups approaches with similar characteristics and highlights their shortcomings with respect to the issues outlined in Section 1.2. As discussed above, the approaches are technically sound. However, the approaches of every single category expose one or more shortcomings that prevent full compliance with the complex characteristics of open environments:

- Complete knowledge contradicts partial observability of open environments
- Central repositories are hard to keep up to date due to the dynamics and non-determinism of open environments
- Central data and control flows contradict the partial observability and non-determinism of open environments
- Central decision-making is constrained by partial observability, dynamics and the non-determinism of open environments all of which introduce uncertainty
- Limited incorporation of local preferences during central decision-making amplifies the effects of partial observability and dynamics of open environments
- Partial automation of the service composition phase constrains the potential of an approach to adjust to partial observability, non-determinism and dynamics of open environments.

The aim of this thesis is to create an approach that avoids the aforementioned shortcomings, using the weak notion of agency as basis for a uniform and coherent agent-
oriented software paradigm. The focus of the approach taken in this thesis is on the automation of the entire service composition process as described in Czajkowski et al. [32] and Sohrabi et al. [156], but without limiting centralised system structures and rigid agent dependencies. The approach is outlined in the next chapter. It is aligned with several techniques and methods that are described separately from each other in different related work, such as:

- Local decision-making based on local context and preferences as in Maamar et al. [111] and Preist et al. [140]
- Decentralised market-based decision-making and interactions as in Preist et al. [140] and Cao et al. [24]
- Decentralised multiagent system architecture based on peers with equal rights as in Küngas and Matskin [100] and Charif-Djebbar and Sabouret [27]
- Partial knowledge about initial state during service planning as in Sirin et al. [152]
- On demand service composition as in Buhler and Vidal [23]
- Real-time constraints on messaging as in Ambroszkiewicz [4].
Part II

Contribution
Chapter 4

On-demand Service Composition in Open Environments

This chapter presents an approach to the problem of dynamic service composition in open environments. There are several ways of modelling open environments. The chapter, therefore, commences with an outline of a concept for modelling open environments which is described best using an open market metaphor (Section 4.1). The chapter then presents a concept of a dynamic service composition approach that matches the particular characteristics of the open market environment (Section 4.2). Finally, the key characteristics of the proposed dynamic service composition approach are discussed (Section 4.3) and then the major constraints and assumption underlying the approach are presented (Section 4.4). The dynamic service composition concept described in this chapter combines a set of methods and techniques that are applied partially in different existing service composition approaches (see Section 3.5). The aim of the presented concept is to alleviate some of the shortcomings that have been identified in related literature (compare Section 1.2 and Chapter 3).

4.1 Scope and Environment

The goal of service composition is the automation of business processes with several of the potential advantages described in Section 2.1.2. As discussed so far, agent-based approaches are aimed at increasing flexibility and adaptability of the service composition
4. ON-DEMAND SERVICE COMPOSITION IN OPEN ENVIRONMENTS

process in order to handle dynamic changes in the environment and runtime exceptions. To
do so, agent-based approaches utilise the automation of particular activities in the service
composition process, e.g. service discovery or service selection. Thereby, agent-based
approaches are applicable in general to both intra-organisational and inter-organisational
business processes.

The scope of this thesis is solely inter-organisational business processes for the fol-
lowing three reasons. Firstly, one major aim of intra-organisational business process
automation is to enforce full control over business processes in a centralised manner.
Centralised workflow-driven approaches match such a requirement exactly. Additional
automation, out of the control of human decision-makers, does not seem desirable in
that context, particularly because it potentially takes the user out of the loop and thus
jeopardises key advantages (such as full control over and predictable costs of business
processes). Secondly, the domain of inter-organisational business process automation in
contrast exhibits explicit characteristics of open environments by accommodating differ-
ent independent corporate bodies with potentially conflicting incentives, lack of central
control, at best partial control over the environment, dynamic short-term collaborations,
etc. Open environments appear to be problematic when tackled with conventional ap-
proaches because these approaches do not necessarily provide a means of abstraction,
decomposition and organisation that match the characteristics of open environments (see
Section 10.1.1). Thirdly, the current trend towards advanced networking, e.g. ad hoc,
wireless or peer-to-peer technologies, creates new challenges by adding to the complexity
of open environments in the domain of inter-organisational business process automation,
with increasing decentralisation, dynamics and uncertainty. Agent technology seems to
be particularly promising for modelling such situations while conserving service-oriented
key principles (compare Section 2.5.3).

This thesis places the service composition problem in an open market environment in
which supply and demand confront each other in the form of independent organisational
peers that provide and consume services. Service providers offer one or more services. Ser-
vice consumers state problems that typically cannot be solved by a single service provider
using a single service. Due to disparate capabilities and differing incentives and prefer-
ences, service providers actively embrace the key market concepts of collaboration and
competition in order to provide composite services that satisfy particular service requests. Service providers congregate proactively in teams in order to collaborate and provide competitive solutions together. Composite services are beneficial for both sides. They embody valid problem solutions for service consumers whereas their consumption yields measurable gains (e.g. monetary profits or resource optimisations) for service providers.

The outlined open market environment is modelled to exhibit the characteristics of open environments as defined in Section 2.3. Wooldridge [169, preface] remarks that one reason for the enormous growth in the multiagent community is that it is driven “at least in part by the belief that agents are an appropriate software paradigm through which to exploit the possibilities presented by massive open distributed systems – such as the Internet.” In that sense, the scope of this thesis on dynamic inter-organisational service composition in open environments appears to be a natural candidate for an agent-oriented problem solution. This correlation of the presented open market metaphor with Wooldridge’s statement is the motivating force driving the development of a multiagent approach to dynamic service composition.

![Conceptual model of an open market environment.](image)

The general concept of a system that models an open market environment is depicted in Figure 4.1. This will be referred to as the *open market concept* hereafter. The open market concept consequently follows the scope of the open market environment. It defines a system populated by various service provider entities representing service supply on the
one hand and service consumer entities embodying service demand on the other hand. Hence the open market concept models only the actors in the open market environment.

All entities are connected through a universal communication channel. Note that there is no particular order or hierarchy in the open market concept. All entities are peers with equal rights. The communication approach is message-oriented. Information exchange among entities is performed with unicast (between two peers), multicast (one-to-many peers) or broadcast (one-to-all peers) messages, subject to the knowledge and state of a sending peer.

Hence the open market concept does not limit the generality of the communication channel to any particular infrastructure or technology. It considers a most basic approach to networking based on an unstructured peer-to-peer overlay network on top of the transport layer. It is assumed that peers (entities) discover each other directly without the existence of central repositories by using network-flooding-based search operations. Central repositories are avoided because they impose performance bottlenecks and are difficult to keep up to date in dynamic environments, as outlined in Paolucci et al. [132].

The use of an unstructured peer-to-peer approach is further motivated by the interpretation of the notion of agency provided earlier and the fact that this thesis is aimed at applying agent orientation as a universal coherent software engineering paradigm (compare Section 2.5). Central repositories seem very likely to violate the notion of agency (e.g. autonomy, proactiveness or social interactions) in this context because they are typically implemented as purely passive data containers.

Moreover, the open market concept does not incorporate additional mediation entities in a middle layer on top of the overlay network or between entities. Middle layers (e.g. brokers, mediators, proxies and wrappers) are in general functional systems aimed at providing reliable, secure and transactional operations for higher application layers. They are implemented as passive and fully deterministic systems. This, however, is a direct contradiction of the notion of agent autonomy and also violates the proactiveness property as interpreted in Section 2.4.2.

Simplicity and lack of structure of the open market concept are drawn from the properties of the open market metaphor. One direct consequence of the lack of predefined structure is limited knowledge about other entities in the system: their availability, their
4.1. SCOPE AND ENVIRONMENT

location, their preferences, their relationships and dependencies. Hence entities obtain information about an open market environment, and infer knowledge from it, only through communication.

Extensive communication is necessary to compensate for the lack of structure and to establish necessary and sufficient relationships and dependencies among entities. By exchanging messages, entities increase their knowledge incrementally in a stepwise fashion. However, inferred knowledge is neither absolute nor complete: it is subject to ageing. It cannot be guaranteed in general that a piece of information that is valid at time $t_0$ is still valid at a later time $t_n > t_0$. Information may become obsolete or even false.

Since the goal of this thesis is to develop a multiagent approach to dynamic service composition, the actors of the open market concept – service providers and service consumers – are modelled with software agents (see Chapter 6.1). The specific software agents are referred to as service provider agents and service consumer agents respectively in the remainder of this thesis.

![Figure 4.2: Visualisation of the complexity of the open market concept with the dungeon metaphor.](image)

The complexity of the open market concept can be illustrated with a person walking a dark dungeon with a torch (Figure 4.2). The light of the torch is not bright enough to illuminate the entire dungeon nor single rooms. While the person is roaming, they collect information about items placed in the dungeon located within the cone of light of the torch. However, after moving on, a visited room falls back into darkness and the person cannot see whether the items have since changed. The person can only obtain
such information by traversing a particular room again. If two or more persons separately roam the dungeon, they have direct connections with each other only when they meet in a room. In all other cases, the only way to share information is by remote communication, e.g. shouting.

### 4.2 On-Demand Service Composition Concept

After having delineated the scope and environment of the open market concept, this section presents an outline of a corresponding dynamic service composition concept. The focus of this section is merely on introducing and highlighting the characteristics of the on-demand service composition concept. Detailed semi-formal specifications of a concrete service composition method, as well as the static and dynamic models of a multiagent system implementing the on-demand service composition concept, are provided in subsequent chapters.

The analysis of relevant agent-based approaches to service composition (compare Chapter 3) reveals that existing solutions improve adaptability and flexibility of the service composition process by shifting service discovery, matchmaking and selection from design-time (offline) into run-time (online) – or in other words, from static to dynamic. Service planning, however, is in many cases considered an offline activity. As a consequence, existing approaches are based on fixed workflow process definitions that are known a priori to a service consumer, service broker or intermediary. In contrast, if online service planning is employed, service discovery, matchmaking and selection are typically based on the assumption of completeness and timeliness of information about available concrete services and their properties (see Section 3.3). These assumptions are problematic with respect to the characteristics of open environments (compare Section 1.2).

In summary, the focus of existing approaches is typically on the instantiation of static workflow process definitions that is performed by central entities based on complete and correct knowledge. Moreover, online planning approaches to dynamic service composition are also based on central entities that have complete knowledge about the type and availability of services. The adaptability of existing approaches to decentralisation, dynamics and uncertainty in open environments is limited (compare Chapter 3).

This thesis proposes on-demand service composition that incorporates a fully online
service composition process – including service planning. The aim is to eradicate the above shortcomings with respect to the characteristics of open environments. The on-demand service composition concept is particularly focused on service planning and service selection as the key concepts for achieving flexible and adaptive service composition in open environments. It exploits dynamic planning and implements decentralised decision-making under uncertainty for service selection based on limited ad hoc information.

The description of the on-demand service composition concept is based on the activities of the service composition life-cycle stage outlined in Section 2.2. The following subsections present the activities of the on-demand service composition concept in sequential order. However, it needs to be noted that all activities but service planning are eventually performed concurrently subject to timing and the order of exchanged messages between the software agents that participate in a concrete service composition process. The concurrency of service composition activities is a direct consequence of the characteristics of the open market concept.

![Visualisation of the on-demand service composition concept with the classroom metaphor.](image)

**Figure 4.3:** Visualisation of the on-demand service composition concept with the classroom metaphor.

The on-demand service composition concept envisaged in this thesis can be illustrated with a classroom example (Figure 4.3). Consider the situation when a teacher asks pupils in class to form teams for solving a group task, e.g. to form volleyball teams or to conduct and evaluate a complex experiment. Figure 4.3 a) depicts the original loose organisational structure among pupils before the teacher’s request. The teacher’s request can be interpreted as a service request that is issued as a broadcast message to all pupils in class. Following the request, the pupils start congregating in groups according to the requirements of the request, the skills of individual pupils in class and the private
preferences of individual pupils, e.g. whom they like or dislike. The composition of teams may take a while, subject to the complexity of the request and the differences in private preferences. It is performed with a lot of communication between the pupils. Eventually, the class arrives at the situation sketched in Figure 4.3 b) in which the pupils have organised themselves into teams. The teams represent on-demand structures that are only valid as long as the group task has not been solved. They will dissolve eventually, e.g. at the end of a lesson they will fall back into the organisational structure of Figure 4.3 a). The teams also reflect the ad hoc character of on-demand service composition. The team structures are adapted to the current situation in class. They may not be reproduced for the same teacher request in similar situations, e.g. if some pupils are on sick leave or other pupils have joined the class.

4.2.1 Service Request Creation

The creation of a service request is not a fundamental activity of the service composition life-cycle stage delineated in Section 2.2. However, this activity is outlined here because the existence of a service request is imperative for triggering any type of service composition process. The creation of a service request is performed by a service consumer agent. It elicits information about a goal to achieve, including QoS requirements, from its associated user and generates a service request from the information obtained. Elicitation of the problem statement and creation of a service request depend on the characteristics of a particular application domain. In the course of this thesis, service requests need to be created for experimental evaluation and as part of a demo application. In the former, they are automatically generated based on scripts; in the latter service requests are elicited with a graphical user interface (see Chapter 8).

A service request represents a user goal as a transformation of some input into some output. Except for trivial cases, a goal is represented by a complex structure of interrelated tasks that all together specify a concrete problem to be solved. The corresponding data structures used to model service requests are presented in Section 6.3. In this thesis, a service request contains the following elements:

- **Set of tasks**

  A single task is specified with a keyword, a set of input and output parameters and
a set of preconditions and effects. The keyword is a unique identifier that denotes a specific service category with respect to the nomenclature of a concrete problem domain (e.g. specified within the UNSPSC classification system [162]). Input and output parameters are expressed with triples comprising id, key and value. The id is a unique identifier within the goal. The key refers to a parameter type. The value represents a placeholder or concrete element in the parameter type’s domain. Preconditions and effects specify constraints on input and output parameters or between them in order to describe the conditions of a proper transformation of the task’s input parameters into output parameters.

• **Control structures**
The temporal and functional dependencies between tasks are expressed with control structures such as sequence, split or join. The control structures specify tasks as initial, terminal or intermediary. Initial tasks do not have preceding tasks (do not receive inputs from other tasks). Terminal tasks are not followed by succeeding tasks (do not provide inputs to other tasks). Intermediary tasks are tasks that have dependencies to both preceding and succeeding tasks. Furthermore, the control structures specify the control and data flows between the tasks of a service request that define the semantic meaning of a goal.

• **QoS requirements**
QoS requirements are separately specified per task. They are defined with tuples in the form of key–weight pairs. The key denotes a particular QoS parameter. The weight expresses the level of importance of a QoS parameter with a value in the interval \([0, 1]\). A small value or 0 indicate little or no importance, whereas a large value or 1 indicates high importance. In addition, the weights of all QoS parameters associated with a single task must sum to 1.

• **Service request timeout**
The service request timeout indicates the maximum amount of time the user allows for the service composition process. The timeout is specified in milliseconds.

Consider an example from the travel planning domain as an illustration of a service request. In this example, a student wishes to organise a return trip from Hawthorn, a
4. ON-DEMAND SERVICE COMPOSITION IN OPEN ENVIRONMENTS

suburb of Melbourne to the city of Sydney including a one-night stay. The student’s goal can be expressed with three sequentially ordered tasks as outlined in Table 4.1.

Table 4.1: Simplified illustration of an example service request for a return trip from Hawthorn, Melbourne to Sydney city centre including a one-night stay.

<table>
<thead>
<tr>
<th>Service Request 1</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Control structure: Sequence</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timeout: 3000 ms</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task 1: outbound trip</td>
<td>Task 2: stay over</td>
<td>Task 3: inbound trip</td>
<td></td>
</tr>
<tr>
<td>Key</td>
<td>Transportation</td>
<td>Accommodation</td>
<td>Transportation</td>
</tr>
<tr>
<td>Input</td>
<td>{from, Hawthorn (Mel)}</td>
<td>{in, City (Syd)}</td>
<td>{from, City (Syd)}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{quality, budget}</td>
<td></td>
</tr>
<tr>
<td>Output</td>
<td>{to, City (Syd)}</td>
<td>{in, City (Syd)}</td>
<td>{to, Hawthorn (Mel)}</td>
</tr>
<tr>
<td>QoS</td>
<td>{costs, 0.0}</td>
<td>{costs, 1.0}</td>
<td>{costs, 0.7}</td>
</tr>
<tr>
<td></td>
<td>{duration, 1.0}</td>
<td></td>
<td>{duration, 0.3}</td>
</tr>
</tbody>
</table>

Besides key, input and output parameters, the student also specifies QoS requirements for each goal concerning the duration and costs of the trip. The semantics of the weights may be explained as following. The student is interested in the fastest connection for the outbound travel, e.g. because he needs to be on time for a workshop. The choice of accommodation depends on the price with the aim to find the cheapest available because it is just for a few hours over night. On the way back to Melbourne, the student prefers a low-budget connection with a reasonable travel time in order to reduce the total costs of the trip.

4.2.2 Service Planning

The on-demand service composition process is triggered by a new service request. After having generated a service request, a service consumer agent performs automated planning. The on-demand service composition concept does not require fixed workflow process definitions that are created only once at design-time (offline) but rather exploits dynamic planning in order to decompose the complexity of the tasks of a service request online. Hierarchical planning is envisaged that is based on a functional decomposition of a prob-
4.2. ON-DEMAND SERVICE COMPOSITION CONCEPT

In this thesis, the principle of separation of concerns is applied to the problem domain. Service planning asserts the existence of general knowledge about how complex problems in a particular problem domain can be modelled, decomposed, and solved. This knowledge is available to all participants in the open market environment in the form of a domain ontology.

A brief outline of automated planning is provided in Section 7.3. In general, the foundation of hierarchical planning is that the tasks of a service request are complex problem descriptions that can be decomposed with sets of small composable plans known as methods. The corresponding data structures used to model methods are presented in Section 6.3. In this thesis, a method contains the following elements:

- **Keyword**
  A method is associated with a unique identifier that denotes a specific family of service categories with respect to the nomenclature of a concrete problem domain (e.g. specified within the UNSPSC classification system [162]).

- **Input/output parameters**
  A method defines the decomposition of a compound task into a functionally and semantically equivalent network of component tasks based on the information transformation between input and output parameters of component tasks. Input and output parameters are expressed with triples comprising id, key and value. The id is a unique identifier within the method. The key refers to a parameter type. The value represents a placeholder or concrete element in the parameter type’s domain. The input parameters link to the input parameters of component tasks whereas the output parameters correspond to output parameters of component tasks.

- **Preconditions/effects**
  A method also defines constraints for the state change implied by the execution of its tasks. Preconditions and effects specify constraints on input and output parameters, or between them, in order to describe the conditions of a proper transformation of the method’s input parameters into output parameters.
• Set of tasks

A single task is specified with a keyword, a set of input and output parameters and a set of preconditions and effects. The keyword is a unique identifier that denotes a specific service category with respect to the nomenclature of a concrete problem domain (e.g. specified within the UNSPSC classification system [162]). Input and output parameters are expressed with triples comprising id, key and value. The id is a unique identifier within the goal. The key refers to a parameter type. The value represents a placeholder or concrete element in the parameter type’s domain. Preconditions and effects specify constraints on input and output parameters or between them in order to describe the conditions of a proper transformation of the task’s input parameters into output parameters.

• Control structures

The temporal and functional dependencies between tasks are expressed with control structures such as sequence, split or join. The control structures specify tasks as initial, terminal or intermediary. Initial tasks do not have preceding tasks (do not receive inputs from other tasks). Terminal tasks are not followed by succeeding tasks (do not provide inputs to other tasks). Intermediary tasks are tasks that have dependencies to both preceding and succeeding tasks. Furthermore, the control structures specify the control and data flows between the tasks of a service request that define the semantic meaning of a goal.

A method represents abstract procedural knowledge about how to decompose a problem into a network of two or more interlinked subproblems called tasks. Tasks are generic. They do not refer to specific services, service instances or service providers. They model subproblems in an abstract fashion and may be subject to further decomposition depending on the type of task. Tasks that can be further decomposed are known as compound tasks whereas atomic tasks represent basic activities of a problem domain that are not decomposable any further.

Concerning the travel example, a service consumer agent may have a method Method 1 that decomposes Task 1 – the outbound travel from Hawthorn, Melbourne to Sydney city centre – into three sequential tasks as shown in Table 4.2. Method 1 defines the
4.2. ON-DEMAND SERVICE COMPOSITION CONCEPT

Table 4.2: Simplified illustration of an example method Method 1 decomposing Task 1: outbound trip from Hawthorn, Melbourne to Sydney city centre.

<table>
<thead>
<tr>
<th>Method 1</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Key:</strong></td>
<td>Transportation</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Input:</strong></td>
<td>{from, $L_A$}</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Output:</strong></td>
<td>{to, $L_B$}</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Constraint:</strong></td>
<td>$L_A$.$city \neq L_B$.$city$</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Dependency:</strong></td>
<td>Task11.$Output_{to} = Task12.$Input_{from}$</td>
<td>Task12.$Output_{to} = Task13.$Input_{from}$</td>
<td></td>
</tr>
<tr>
<td><strong>Control structure:</strong></td>
<td>Sequence</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Task 11</th>
<th>Task 12</th>
<th>Task 13</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Key</strong></td>
<td>Local Transportation</td>
<td>Intercity Transportation</td>
</tr>
<tr>
<td><strong>Input</strong></td>
<td>{from, $L_A$}</td>
<td>{from, $IH_A$}</td>
</tr>
<tr>
<td><strong>Output</strong></td>
<td>{to, $IH_A$}</td>
<td>{to, $IH_B$}</td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>$L_A$.city = $IH_A$.city</td>
<td>$IH_A$.city $\neq IH_B$.city</td>
</tr>
<tr>
<td></td>
<td>$L_A$.suburb $\neq IH_A$.suburb</td>
<td></td>
</tr>
</tbody>
</table>

Generic transformation of an arbitrary location $L_A$ into another location $L_B$ with the postcondition that $L_B$ is situated in a different city than $L_A$. The tasks of Method 1 specify subproblems, e.g. Task 11 defines local transportation as a generic transformation from location $L_A$ to an intercity travel hub $IH_A$ (a landmark at which intercity means of transport depart/arrive) under the postcondition that both locations remain in the same city but different suburbs. Figure 4.4 depicts an example decomposition tree of the outbound travel task for a journey starting in Melbourne’s suburb Hawthorn. Here, Task 11 is further decomposed by Method 3. Figure 4.4 also illustrates an alternative decomposition of Task 1 in the form of Method 2. Method 1 and Method 2 are semantically equivalent. Both ensure input and output as well as preconditions and effects of the original Task 1. However, they comprise different networks of component tasks.

Automated planning is performed based on the matching of input and output parameters as well as preconditions and effects of compound tasks on the one hand and methods on the other hand. Methods preserve the characteristics of the tasks they decompose.
in order to ensure semantic and syntactical equivalence. The service planning of the on-demand service composition concept terminates if there is no further decomposition of compound tasks possible. The outcome of the service planning activity is a decomposition tree for the task network of the original service request. The leaves of the decomposition tree are atomic tasks.

Hence the creation of a decomposition tree is firstly possible because the application of hierarchical planning techniques reduces the combinatorial complexity of the planning space. Secondly, the creation of a decomposition tree is necessary because, in contrast to classic planning approaches, the automated planning here is not based on the assumption of complete knowledge about the open market environment. Thus no interleaved planning and plan execution (so called online planning) is possible, because operators (concrete services) that can be bound to the generic atomic tasks are not known yet.

The decomposition tree may yield multiple valid decompositions of the original service request (compare the different sub-trees spanned by Method 1 and Method 2 in Figure 4.4). A valid decomposition is called a generic plan. It contains a set of atomic tasks and the control structures implied by the methods that have been used to decompose relevant compound tasks. A composite service request is an enhanced service request in which the original task network (set of tasks and control structures) has been replaced with a
4.2. ON-DEMAND SERVICE COMPOSITION CONCEPT

generic plan. The corresponding data structures used to model composite service requests are presented in Section 6.3. The service consumer agent selects one of the generic plans in order to create a composite service request. The decision-making for selecting generic plans is out of the scope of this thesis. There are various strategies possible that may be investigated in future work.

Table 4.3 provides a simplified illustration of a composite service request for the given travel planning problem. For the sake of simplicity and space, the table does not depict a composite service request for the complete service request given in Table 4.1 but only for Task 1: the outbound trip from Hawthorn, Melbourne to Sydney city centre.

Table 4.3: Simplified illustration of an example composite service request Composite Service Request 1 for Task 1: outbound trip from Hawthorn, Melbourne to Sydney city centre.

<table>
<thead>
<tr>
<th>Composite Service Request 1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Key:</strong> Transportation</td>
</tr>
<tr>
<td><strong>Input:</strong> {from, Hawthorn (Mel)}</td>
</tr>
<tr>
<td><strong>Output:</strong> {to, City (Syd)}</td>
</tr>
<tr>
<td><strong>QoS:</strong> {costs, 0.0}</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Dependency:</strong> Task11.Output = Task12.Input_from</td>
</tr>
<tr>
<td><strong>Control structure:</strong> Sequence</td>
</tr>
<tr>
<td><strong>Timeout:</strong> 3000 ms</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Task 11</th>
<th>Task 12</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Key</strong></td>
<td>Local Transportation</td>
</tr>
<tr>
<td><strong>Input</strong></td>
<td>{from, (L_A)}</td>
</tr>
<tr>
<td><strong>Output</strong></td>
<td>{to, (IH_A)}</td>
</tr>
<tr>
<td><strong>Constraint</strong></td>
<td>(L_A).city = (IH_A).city)</td>
</tr>
<tr>
<td></td>
<td>(L_A).suburb (\neq) (IH_A).suburb</td>
</tr>
</tbody>
</table>

The service planning of the on-demand service composition concept exhibits interesting properties in comparison to pre-defined workflow-driven approaches. The use of automated planning offers:

- Greater flexibility and expressiveness for creating composite services from user-
defined goals in comparison to static pre-defined workflow process definitions

- A decoupling of problem definition from problem solution that increases the adaptability of the service consumer agent to changes in the environment simply by potentially providing alternative solutions to one problem

- Increased extensibility of service composition capabilities by supporting the incorporation of additional methods for particular new subproblems.

### 4.2.3 Service Discovery and Matchmaking

Service discovery and matchmaking of the on-demand service composition concept significantly differs from related work (compare Chapter 3). A service consumer agent triggers the service discovery and service matchmaking activities by disseminating a composite service request. Advertising a need for a composite service rather than advertising services can offer advantages in dynamic environments (as noted by Martin et al. [114]), especially where particular atomic services are frequently demanded. Hence service provider agents can decide autonomously when to proceed with a next request as soon as they are idle again, resulting in a more effective use of their local resources.

Active service request dissemination in this thesis is necessary because of the stochastic character and limited observability of the open system that is spanned by the open market concept. Hence the open market concept does not incorporate central or distributed repositories because it only models actors but not passive data sinks. A service consumer agent, therefore, does not have access to reliable information about available service provider agents due to the system’s intrinsic dynamic and uncertain character.

Due to this lack of knowledge and explicit structures, both of which are imperative for facilitating information exchange between software agents, a service consumer agent cannot publish a composite service request in a public repository nor can it look up and contact service provider agents directly, e.g. with unicast interactions. Thus it sends out a broadcast message that is received by all service provider agents that actively listen on the communication channel. Broadcasting causes potentially high communication costs. However, it is an essential means of multiagent communication amongst autonomous agents as described in d’Inverno and Luck [36, p.126] in the context of the contract net
4.2. ON-DEMAND SERVICE COMPOSITION CONCEPT

Accordingly, service discovery in the on-demand service composition concept is better characterised as active on-demand service advertising. Service advertisements are disseminated by service provider agents also via broadcast messages. Broadcasting again is necessary, because service provider agents also do not possess reliable information about other available service provider agents due to the dynamics and uncertainty of open environments and the lack of structure in the multiagent system at this stage. Using broadcast messaging, a service provider agent advertises its service amongst all service provider agents in a problem domain, reaching both potential partners and competitors. See Section 6.4.2 for details about the corresponding interaction protocol.

A service advertisement is a declarative specification of a service interface and behaviour. The corresponding data structures used to model service advertisements are presented in Section 6.3. In general, it includes the following information:

- **General provider information**
  Information about the real business entity, contact details, etc. as defined, for example, in the OWL-S Service Profile [114, 115].

- **Service type information**
  The service type of a service is defined with a unique identifier that denotes a specific service category with respect to the nomenclature of a concrete problem domain (e.g. specified within the UNSPSC classification system [162]).

- **Input/output parameters**
  A service advertisement is specified by the information transformation between input and output parameters. Input and output parameters are expressed with triples comprising id, key and value. The id is a unique identifier within the advertisement. The key refers to a parameter type. The value represents a concrete element in the parameter type’s domain.

- **Preconditions/effects**
  A service advertisement also defines constraints for the state change implied by its execution. Preconditions and effects specify constraints on input and output param-
eters, or between them, in order to describe the conditions of a proper transformation of the service’s input parameters into output parameters.

- **QoS constraints**
  Services also have non-functional properties. QoS constraints are specified with tuples in the form of key–value pairs. The key denotes a particular QoS parameter. The value represents a concrete element of the corresponding domain of the parameter’s type. The number and type of QoS parameters depend on the QoS preferences specified with a composite service request. In order to provide the foundation for successful matching, a service advertisement must define at least the QoS parameter types specified with the original task (the one being decomposed) of a composite service request. However, a service advertisement may optionally comprise additional QoS parameters, e.g. for distinguishing a service offer from competition.

- **Reference to composite service request**
  The reference ensures the unambiguous association of a service advertisement with the original composite service request.

- **Expiration deadline**
  The deadline defines for how long an advertisement is valid. After the deadline has passed, the advertised service may not be offered any more in the advertised form by its associated service provider agent.

Every service provider agent triggers the dissemination of its service advertisement only after it detects a matching of its own service with an atomic task of a composite service request. The matching between an atomic task and a service is based on capability matchmaking of input and output parameters as well as preconditions and effects between a task definition and a service advertisement. Capability matchmaking is outlined in detail in Section 7.4. If a service provider agent does not detect such a matching, it does not pursue any further actions and may return into some idle state waiting for new composite service requests to arrive. In case of a positive matching, a service provider agent engages in further activities in order to attempt the composition of a composite service.

Table 4.4 presents an example service advertisement of the given travel planning example that matches Task 12 of Composite Service Request 1. The advertisement is published.
Table 4.4: Simplified illustration of an example service advertisement for Task 12: outbound trip from Melbourne city centre to Sydney city centre.

<table>
<thead>
<tr>
<th></th>
<th>Service Advertisement 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provider</td>
<td>ABC Coach Service</td>
</tr>
<tr>
<td>Service type</td>
<td>Intercity Transportation</td>
</tr>
<tr>
<td>Input</td>
<td>{from, City (Mel)}</td>
</tr>
<tr>
<td>Output</td>
<td>{to, City (Syd)}</td>
</tr>
<tr>
<td>Constraint</td>
<td>Input.city ≠ Output.city</td>
</tr>
<tr>
<td>QoS</td>
<td>{costs, 65 AUD}</td>
</tr>
<tr>
<td></td>
<td>{duration, 720 min}</td>
</tr>
<tr>
<td>Reference</td>
<td>Composite Service Request 1</td>
</tr>
<tr>
<td>Expiration deadline</td>
<td>12 pm on 25/01/2009</td>
</tr>
</tbody>
</table>

by ABC Coach Service. It advertises an intercity coach connection that takes 12 hours from Melbourne to Sydney and costs 65 AUD. There may be further service advertisements that match Task 12 such as from other intercity coach services or intercity train services.

The second step for every service provider agent is the discovery and matchmaking of service advertisements of other service provider agents. In general, it is assumed that every non-trivial composite service request contains multiple atomic tasks. Consequently, multiple services are required for creating a composite service. Thus service provider agents need to discover a necessary and sufficient number of other service provider agents and their respective services.

Service discovery in this thesis is an active service advertising method. Service discovery is reduced to fetching incoming service advertisement messages from other service provider agents. A service provider agent matches newly received service advertisements with all the atomic tasks of the composite service request. If a match is found, a service advertisement is linked to the respective task (see Figure 4.5 for an example).

A sufficient number of service advertisements is crucial for further processing. A service provider agent engages in the subsequent service selection activity only if it has detected at least one service advertisement for each task of a composite service request. From
this point, service discovery and matchmaking on the one hand and subsequent service selection activities on the other hand are performed in parallel by a service provider agent, depending on the type and timing of newly arriving messages. Hence service advertisements may drop in at any time in between the original composite service request and the service request timeout.

![Figure 4.5: Visualisation of the linking of service advertisements to the tasks of Composite Service Request 1 for Task 1: outbound trip from Hawthorn, Melbourne to Sydney city centre.](image)

Service discovery and matchmaking result in two implications that are important for further processing. Based on the exchange of service advertisements, service provider agents obtain information about the availability and location of other service provider agents and their respective services. This information triggers the transformation of the multiagent system from a completely unstructured to a partially structured system of service provider agents by reducing the anonymity between service provider agents working the same composite service request. Service provider agents are now able to engage in future interactions in a point-to-point fashion using unicast messages. This way, the communication overhead accumulated with broadcast messaging is cut down.

Moreover, every service advertisement embodies an expression of interest: the invitation to get into contact and to discover opportunities to collaborate. Service advertisements form the basis of any future decision-making. They are assumed to be binding with respect to their content throughout a single service composition process. Hence service
advertisements cannot be altered after being sent. They are valid until their expiration
deadline has passed.

4.2.4 Service Selection

In general, service selection is concerned with assigning an optimal set of services that
match the tasks of a composite service request. One key aspect of this thesis is the
development of a dynamic service selection method that is designed to comply with the
characteristics of the open market concept. Hence the on-demand service composition
concept does not incorporate central data and control flows nor central decision-making.
There is no single central instance such as a broker or mediator agent controlling the
service selection process.

Service selection is performed on-demand. The service selection process can be char-
acterised as a bottom-up approach. Due to the lack of any central control it is the service
provider agents that proactively negotiate with each other in order to propose compos-
ite services to a requesting service consumer agent. Service provider agents engage in a
dynamic coalition formation process in which they self-organise in coalitions according to:

- Client-defined QoS requirements
- Available service advertisements of potential coalition partners and
- Local private preferences.

The congregation of service provider agents in coalitions is facilitated by direct bi-
directional interactions based on unicast messaging and fully decentralised autonomous
decision-making. Thus service provider agents have direct and full influence over the
selection of their coalition partners and retain full autonomy.

Every complete coalition contains a necessary and sufficient set of service provider
agents that together are able to provide a requested composite service. There may be
multiple coalitions that satisfy the requirements of a single composite service request,
depending on the number and nature of available service provider agents. All coalitions
concurrently and proactively propose their competing composite service proposals to the
requesting service consumer agent which eventually decides on a single winning coalition
according to client-defined QoS preferences. The coalition formation model underlying the on-demand service composition concept is proposed in Section 5.3.

The service composition process is composed of three distinct phases. They are outlined below in sequential order. However, these phases occur simultaneously for different service provider agents during execution due to the dynamic and stochastic character of an open market environment.

Coalition Initialisation

A service provider agent only engages in the service selection activity if it has obtained at least one service advertisement for every task of a composite service request. The aim of coalition initiation is to establish the basis of a structure that supports the formation of coalitions. Hence the open market system is initially unstructured with no explicit dependencies and relationships among service provider agents.

A service provider agent that has obtained a sufficient set of service advertisements converts into the role of a *coalition candidate*. Before engaging in interactions for service selection, a service provider agent needs to determine the most likely coalition partners. Given the precondition of the existence of at least one service advertisement for every task of a composite service request prior to service selection, it is always possible for a service provider agent to determine potential partners. This is achieved with local decision-making under consideration of the client-defined QoS requirements of a composite service request, the set of service advertisements that are at a service provider agent’s disposal and its private preferences.

Local autonomous decision-making is performed concurrently and independently by all participating service provider agents. It is performed under uncertainty due to the dynamics and decentralisation of an open market environment. The full decentralisation of decision-making and the autonomy of each service provider agent establish a high degree of freedom for decision-makers and imposes little control over their actions. In order to foster goal-directed behaviour of autonomous service provider agents – behaviour directed at the solution of composite service requests – the local decision-making is constrained by market-based control. This means that service provider agents can only gain benefits if they best satisfy the client-defined QoS requirements of a composite service request.
Hence the final decision of the winning coalition remains with the service consumer agent.

Coalition initialisation commences for a service provider agent (hereinafter referred to as the sender) if it requests another service provider agent (hereinafter referred to as the receiver) to form a coalition. If the receiver accepts to join, both service provider agents have initiated a base coalition and change their roles. The sender adopts the \textit{coalition leader} role. The receiver transfers into the \textit{coalition member} role. See Section 6.4.3 for details of the corresponding interaction protocol.

Hence coalition initialisation yields an implicit leader election mechanism. If a receiver accepts to join a coalition, it also accepts the sender as the leader of that coalition. In case a receiver does not accept the sender as leader, it simply rejects the request. The proposed approach does not support an explicit leader election because election is potentially undecidable amongst autonomous self-interested software agents that may not be able to reach agreement due to conflicting local preferences (particularly in case of two software agents).

Coalition initialisation is typically triggered concurrently by different service provider agents. The receiver also may launch its own request to the sender or any other prospective service provider agent.

\textbf{Coalition Extension}

With the exception of trivial cases, a base coalition is typically not sufficient to satisfy a composite service request. This is especially the case where a service provider agent provides only one service (an assumption of this thesis). Thus a base coalition needs to be extended until it is complete. Coalition extension is facilitated by repeated coordination and communication activities between a coalition leader and coalition members until a complete coalition has been achieved or the composite service request timeout is passed. A coalition is complete if every task in a composite service request has been assigned a service/service provider agent respectively.

The structure of a base coalition supports coalition-extension activities. It establishes an ad hoc organisation of coalition members around a central coalition leader. A coalition leader acts as a \textit{"primus inter pares"} (first among equals), triggering proactively actions for the extension and eventually completion of a coalition. Coalition members of
a coalition are not actively involved in coalition extension and completion activities but remain autonomous and may proactively leave coalitions. This implicit structure between coalition leader and coalition members does not exist by default but emerges based on interaction and agreement amongst service provider agents. This emerging structure proves beneficial by simplifying coordination efforts and reducing communication costs amongst coalition participants.

Coalition extension is triggered by the coalition leader. It iteratively determines the next prospective partners and engages in interactions with these prospective service provider agents. As with coalition initialisation, the coalition leader requests a prospective service provider agent to join the coalition. If the requested service provider agent accepts, the coalition leader informs all original coalition members about the new member to indicate that the coalition has been extended. See Section 6.4.4 for details of the corresponding interaction protocol.

Coalition extension does not incorporate a voting protocol for eliciting the preferences about prospective partners from coalition members. As a matter of fact, the coalition leader is the sole decision-maker in this regard. Voting or any other form of eliciting coalition member preferences may have a severe impact on the performance of the service selection activity due to potentially conflicting private preferences of service provider agents. Moreover, the dynamics and partial observability of an open market system may invalidate a voting outcome at any time.

Service provider agents may decide to leave a coalition because they received a request to join a more prospective service provider agent or coalition or because they disagree with a new coalition member that just joined the coalition. The departure of a coalition member is typically of no consequence for the structure of a coalition. However, if the coalition leader leaves, the coalition disbands and the coalition members returns into the role of coalition candidate. No new coalition leader is elected. As stressed before, the proposed approach avoids the risk of elections among autonomous self-interested decision-makers. If a leaving member agent leaves only the coalition leader behind, the coalition ceases to exist and the leader returns into the role of coalition candidate. Coalition extension accommodates interaction protocols for supporting member leave (see Section 6.4.6) and leader leave (see Section 6.4.5) situations.
Coalition Registration

Coalition registration is triggered by the coalition leader of a coalition. If the coalition leader detects that the coalition is complete – a necessary and sufficient set of service provider agents and their respective services have congregated – it requests all coalition members to confirm their commitment to the coalition in order to prepare and submit a composite service proposal to the waiting service consumer agent. The data model of a composite service proposal is specified in Section 6.3. In general, a composite service proposal contains the following elements:

- **Reference to the original composite service request**
  The reference is necessary in order to unambiguously identify the composite service request that a composite service proposal refers to. Moreover, it is not necessary that a composite service proposal contains the complete structure of a composite service request.

- **Set of service advertisements**
  The service advertisements represent the coalition partners that congregated to provide the composite service proposal. See the outline of service advertisements in Section 4.2.3 for more details.

- **Expiration deadline**
  The provision of a composite service proposal binds the resources of service provider agents. The validity of a composite service proposal is, therefore, limited to a specific period of time. The coalition partners do not guarantee a proposed composite service after the deadline has passed.

If not all coalition members respond with a positive confirmation, the coalition leader determines the reason (e.g. failure or leave of a coalition member) and triggers actions to solve the problem in a timely fashion. Thus the coalition leader returns to engaging in coalition extension activities. See Section 6.4.7 for details of the corresponding interaction protocol.

Service provider agents remain active and pursue the goal of creating a complete coalition until they succeed and register a composite service proposal with the service consumer
agent or the timeout of a composite service request is passed. Local autonomous decision-making and temporal constraints, however, impose risks with respect to the completeness of this approach. Conflicting local preferences may delay the entire service selection process, resulting in non-optimal or even no feasible solutions at all. These drawbacks are the result of the adaptation of the proposed concept for on-demand service composition to the characteristics of open market environments.

In the given travel planning example, two potential coalitions can be identified with respect to the example advertisements introduced in Figure 4.5. Assume that the respective service provider agents formed coalitions as summarised in Table 4.5. The service of every service provider agent is specified with the name of the corresponding provider and the QoS constraints imposed for costs and duration, e.g. the tram service is advertised with costs of 3 dollars and a duration of 15 minutes. The coalition value is the total costs and duration of the coalition.

<table>
<thead>
<tr>
<th>Participant name + costs/duration</th>
<th>Coalition 1</th>
<th>Coalition 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train, $3/15min</td>
<td>Coach Service, $65/720 min</td>
<td>Taxi, $20/10 min</td>
</tr>
<tr>
<td>$68/735 min</td>
<td>$115/670 min</td>
<td></td>
</tr>
</tbody>
</table>

### Table 4.5: Simplified illustration of possible complete example coalitions for Task 1: outbound travel from Hawthorn, Melbourne to Sydney city centre.

#### 4.2.5 Service Contracting

The last activity in the composite service life-cycle is service contracting. The precondition for doing so is the timely submission of at least one composite service proposal. The requesting service consumer agent makes the final decision as to which, if any, coalition’s composite service proposal wins and is contracted for execution. If the service consumer agent has received two or more composite service proposals it ranks them according to the client-defined QoS preferences. The highest ranked composite service proposal wins because it best fulfils the client-defined QoS requirements.

After a service consumer agent decides on a winning proposal, it informs all coalition
4.2. ON-DEMAND SERVICE COMPOSITION CONCEPT

leaders of the outcome of its deliberation by sending an acceptance message to the winner and rejection notes to the losers. The service provider agents of the winning coalition and the service consumer agent then form and fix a binding SLA that represents the base for later execution of a composite service as proposed by the coalitions. See Section 6.4.7 for details of the corresponding interaction protocol.

If no composite service request has been submitted on time, a service composition process terminates without result. It also terminates without result if the service consumer agent rejects all submitted composite service requests.

All coalitions dissolve after completion or termination of a service composition process because they are no longer needed. Every service provider agent returns to some idle state in which it is ready to engage in further service composition processes. The proposed service composition approach is truly on-demand. It avoids keeping track of structures and information as a full adjustment to the dynamics and uncertainty of open environments: on the one hand, the properties and availability of services can change at any time, and on the other hand, functionally similar composite service requests of different service consumer agents most likely differ with respect to their QoS requirements that have a direct influence on the congregation of service provider agents. Accordingly, long-term structures and dependencies are not suitable.

Table 4.6: Simplified unnormalised ranking of two coalitions for three scenarios of client-defined QoS preferences for Task 1: outbound travel from Hawthorn, Melbourne to Sydney city centre.

<table>
<thead>
<tr>
<th>Scenario 1</th>
<th>Scenario 2</th>
<th>Scenario 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weight$_{costs}$ = 0.0</td>
<td>Weight$_{costs}$ = 0.7</td>
<td>Weight$_{costs}$ = 1.0</td>
</tr>
<tr>
<td>Weight$_{duration}$ = 1.0</td>
<td>Weight$_{duration}$ = 0.3</td>
<td>Weight$_{duration}$ = 0.0</td>
</tr>
<tr>
<td>Coalition 1</td>
<td>735.0</td>
<td>268.1 (winner)</td>
</tr>
<tr>
<td>Coalition 2</td>
<td>670.0 (winner)</td>
<td>281.5</td>
</tr>
</tbody>
</table>

With respect to the travel planning example, the task of the service consumer agent is to rank the two coalitions depicted in Table 4.5 according to the client-defined QoS requirements presented in Table 4.1. Since the weight for duration is set to 1.0, the coalition with the lowest duration is the winner: Coalition 2. In this context, note the strong impact of the client-defined QoS requirements. For example, if the weights for
4. ON-DEMAND SERVICE COMPOSITION IN OPEN ENVIRONMENTS

cost and duration were set to 0.7 and 0.3 as with Task 3 of the original service request (compare Table 4.1), Coalition 1 would become the winner. If the weights were set to 1.0 for costs and 0.0 for duration, Coalition 1 also wins. Table 4.6 presents a comparison of these three scenarios. For demonstration purpose only, the ranking of the coalitions is based on a simplified calculation without considering normalisation of the QoS parameter domains. Costs and duration values of the coalitions are as shown in Table 4.5. The significant influence of client-defined QoS requirements in the travel planning example illustrates the market-based mechanism implied by the open market concept, as discussed in the following section.

4.3 Discussion of Key Characteristics

The focus of the service composition approach in this thesis is on its consequent tailoring to the key characteristics of open environments (see Section 2.3). The conceptual model of the open market concept presented is distinguished from related work (compare Section 3) by harnessing features that alleviate the shortcomings of existing agent-based approaches to dynamic service composition as identified in Section 1.2:

- **No assertion of complete knowledge**
  The concept presented does not assert the existence of complete knowledge about the service composition environment. Firstly it specifies the location, type and availability of service provider agents and their respective services. Service provider agents join and leave the open market environment at will and without notice. A second aspect covers the local knowledge as well as private preferences and incentives of service provider agents that are not or only partially revealed. Finally, relationships between service provider agents are dynamic. They change according to local decision-making without notifying external third parties. Consequently, no single service provider agent in an open market system exhibits full knowledge. The concept presented satisfies partial observability of open environments.

- **Avoidance of central repositories**
  Information about service provider agents and their associated services in an open market system is not maintained in central service repositories. There is no pre-
4.3. DISCUSSION OF KEY CHARACTERISTICS

accumulation and maintenance of potentially variable or volatile information. Service provider agents actively advertise their services in an ad hoc fashion. The concept presented conforms to the dynamics property of open environments.

- **Decentralised data and control flow**
  
  Decision-making and subsequent interactions among service provider agents in an open market system are fully decentralised. Every service provider agent has full control only over its own processes. This is an intrinsic feature of open markets. Decentralisation is a stronger notion than distribution. Decentralisation implies both distribution and the lack of any central coordination or control.

- **Autonomous local decision-making**
  
  Service provider agents perform actions and trigger interactions in an autonomous fashion based on local knowledge and concurrent decision-making. Decentralisation and autonomous local decision-making imply non-determinism in an open market system. The concept presented complies with the stochastic character of open environments.

- **Incorporation of private information**
  
  Service provider agents conduct local decision-making based on private information and local preferences that reflect their design objectives as well as their actual knowledge and state in an open market system at decision-making time. Decision-making under consideration of private information is a natural means for modelling independent actors in an open market environment.

- **Focus on service planning and service selection**
  
  The software agents in an open market system model and implement all stages of the service composition process as dynamic online activities. The majority of existing agent-based service composition approaches usually target either dynamic service planning or dynamic service discovery, matchmaking and selection. In contrast, the concept presented encompasses dynamic service planning (executed by service consumer agents) as well as dynamic service discovery, matchmaking and selection (performed by service provider agents).
4.3.1 Key Concepts and Principles

The design decisions for enforcing the presented open market concept to adopt the characteristics of open environments (as discussed in Section 2.3) are inspired or based on the following set of principles and concepts.

**Bottom-up Decentralisation**

As outlined in Section 3, the majority of existing agent-based service composition approaches rely on centralised top-down system architectures. The term top-down refers in this context to directed data and control flows between software components of different hierarchically ordered layers. In general, a service requester agent (consumer, mediator or broker) of an upper layer triggers and controls the activities of the service composition process (e.g. service discovery and service selection) in a centralised fashion by invoking the functionality of passive service provider agents (e.g. proxy) of a lower layer (see Figure 4.6 a). These service provider agents are typically passive by nature and have limited influence on the composition process (e.g. the selection of partner service provider agents involved in the provision of a composite service) because the service requester has full control over the decision-making.

![Figure 4.6: Comparison of data and control flows in a) central top-down and b) decentralised bottom-up approaches.](image)

The characteristics of open environments restrict the effectiveness of such centralised top-down approaches due to their partial observability, non-determinism and the distribution of domain knowledge amongst multiple entities. Hence no single entity can obtain complete knowledge and exhibit full control over its environment. Therefore, functionality and control in a corresponding software system must be distributed in a decentralised
4.3. DISCUSSION OF KEY CHARACTERISTICS

fashion by dealing with local problems locally. The result is increased robustness of the resulting software system.

The open market concept offers a fully decentralised system. Composite services are provided by fully autonomous service provider agents that engage proactively in the service composition process in order to form structures that satisfy composite service requests. Hence service provider agents have direct control over their local problem-solving which is a natural model of actors in an open market environment. These service provider agents inform the requesting service consumer agent only after a valid proposition for a composite service request has been constructed and agreed on (see Figure 4.6 b).

In general, the open market concept does not a priori imply layers since all software agents in the system are modelled as peers with equal rights that interact via a shared communication channel (compare Figure 4.1). However, if the different layers for service requesters and service providers of existing approaches are assumed, there is a directed bottom-up flow of information from service provider agents to an upper layer service consumer agent in addition to decentralised data and control flows between the service provider agents. The bottom-up metaphor emphasises the decentralisation of the open market concept.

**On-demand Structural Organisation**

The term *on-demand* in this thesis does not refer to on-demand computing, which is a paradigm concerned with flexible provision of computing and communication infrastructure and aimed at facilitating smooth delivery of business services, especially in peak situations. Rather, the term *on-demand* is used to refer to enabling the construction of ad hoc solutions which are created for a special purpose only and which can neither be generalised nor adapted to serve purposes other than their original purpose.

The open market concept presented incorporates an on-demand mechanism with respect to the structural organisation of the software agents in an open market system. Relationships and dependencies between software agents do not persist long-term. In fact, an open market system may be completely disorganised if no service requests are active. That means no relationships between software agents and, thus no predefined compositional structures exist in an open market system a priori.
Service composition is solely performed on-demand and triggered by newly disseminated composite service requests. Hence service providers interact and create ad hoc structures to construct and provide proposals for the provision of a composite service following a composite service request. Composite service proposals are customised to the QoS preferences included in a composite service request. They are only valid in this specific context. After completion (a SLA has been reached) or termination (the service request timeout is passed) of the service composition process for a specific composite service request, all structures dissolve. Service provider agents discard interactions and organisational roles and return into idle states.

Dynamics and partial observability of open environments pose risks for long-term structures. An on-demand approach represents a better match with these characteristics because it does not require maintenance. Open environments may change rapidly and frequently and thus impose costs (e.g. communication costs) for reflecting these changes in the system’s structural organisation. Tracking of changes may be difficult (e.g. wireless ad hoc networks), impossible without programmer intervention or the amortisation of the costs they incur may exceed the actual costs of the additional communication of an on-demand approach.

Self-Organisation

Self-organisation is the ability of a system to establish a complex organisation of its internal structure without external guidance (Heylighen and Gershenson [70]). Partial observability and uncertainty about the behaviour of single entities of the open market concept result in non-determinism. As such, an open market system is difficult to control from the outside. It avails of self-organising structures derived from independent software agents that are able to adapt to varying situations.

The open market system of this thesis exhibits little organisation in an idle state. However, after a composite service request has been issued into the system, service provider agents self-organise through communication into group structures (coalitions) that are able to yield one or more solutions for a composite service request. Service provider agents engage in interactions as peers with equal rights based on their local decision-making and preferences guided by client-defined QoS requirements. Hence neither a single software
agent nor any external entity controls the service composition process. It is solely based on mutual agreement between interacting service provider agents.

Heylighen and Gershenson [70] emphasise the intrinsic robustness of self-organising systems as their central benefit: self-organisation allows the adaptation to a variety of exceptions with the system performance degrading gradually depending on an increasing severity of the accumulation of errors. There is a margin for error in an open market system. Failure of particular service provider agents does not prevent a feasible solution as long as other service provider agents exist that provide similar services or are able to create a solution for a problem in a different way.

**Emergence**

Self-organisation often creates emergent system behaviour. Emergence is observed and studied in inherently complex adaptive systems such as economical markets, the immune system and complex software systems. Goldstein [61] characterises emergence as the phenomenon of “...the arising of novel and coherent structures, patterns and properties during the process of self-organisation in complex systems.” Emergent phenomena appear in different forms in different systems but share, according to Goldstein [61, p.50], the following properties:

- “Radical novelty: emergents have features that are not previously observed in the complex system under observation. . . .
- “Coherence or correlation: emergents appear as integrated wholes that tend to maintain some sense of identity over time. . . .
- “Global or macro level: the locus of emergent phenomena occurs at a global or macro level, in contrast to the micro-level locus of their components. . . .
- “Dynamical: emergent phenomena are not pre-given wholes but arise as a complex system evolves over time. . . .
- “Ostensive: emergents are recognised by showing themselves. . . .”

An open market system shows emergence by gaining complex system behaviour from a population of interacting autonomous software agents that embrace simple sets of decision-
4. ON-DEMAND SERVICE COMPOSITION IN OPEN ENVIRONMENTS

making rules and thus are clearly distinguished from algorithmic, centralised and deterministic solutions. In the proposed approach, no single service provider agent is in general able to provide the functionality to satisfy a complete composite service request. The provision of a composite service becomes possible only through the organisation of service provider agents in coalitions. Every coalition exhibits the properties of an emergent. The associated composite service proposal of a coalition attains a complex problem solution by aggregating complementing capabilities and divergent local incentives of participating service provider agents. In summary, emergence in an open market system is an effect caused by self-organisation that gains beneficial complex system behaviour from dynamic relationships between relatively simple software agents.

Coalition Formation

In general, the main focus of multiagent systems is on effective distributed problem-solving. Coalition formation represents a cooperation technique in which multiple software agents team up in order to gain the benefits of solving problems collectively that could not be solved otherwise. Coalitions form as a result of binding agreements amongst a set of software agents with regard to the distribution of the benefits. Coalition formation algorithms are mostly based on game theory with the main primitive to achieve optimal and stable coalition structures and fair benefit distribution. However, Klusch and Gerber [93] show that traditional coalition formation approaches are static. They do not model the dynamics and uncertainty in open environments.

This thesis follows Klusch and Gerber [93] in proposing dynamic coalition formation as a research area with the aim to “clarify which kinds of dynamic settings, and to what extent available algorithms for the static formation of stable coalitions, should be adopted.” The on-demand service composition concept exhibits the characteristics of a dynamic coalition formation problem. According to Kraus et al. [98] it can be placed in the request-for-proposal domain in which service consumers disseminate requests in the form of complex tasks and multiple service providers join together to address a request.

Coalition formation supports the flexible and adaptive composition of complementary atomic services provided by autonomous service provider agents. The decision-making of goal-oriented service provider agents is fully decentralised and considers the global require-
ments for a composite service as well as local preferences. Thus service provider agents compete and collaborate in coalitions based on the rationale of market-based mechanisms for gaining local benefits by satisfying global composite service requests.

**Market-based Control**

Clearwater [30, preface] defines a market-based mechanism or control as “...a paradigm for controlling complex systems that would otherwise be very difficult to control, maintain, or expand.” Market-based mechanisms regulate how trading partners reach agreement in a market. In the context of multiagent systems, market-based control imposes guidelines for software agents to act within certain boundaries in a non-malicious and rational fashion. Newell [127] defines a software agent to be rational if it is goal-oriented and selects only those actions it knows will eventually lead it to achieving a specific goal. As such, software agents do not select actions that harm themselves or other software agents. Rationality is crucial for effective problem-solving, particularly in an open environment without central control and guidance.

The fundamental rationale underlying the on-demand service composition concept is that service provider agents gain profits (e.g. measured in monetary units) from participating in the provision of composite services. The market-based mechanism guiding service provider agents is client-centric. Service consumer agents issue composite service requests in the form of requests for proposals. Service provider agents attempt to congregate in coalitions to satisfy such a request for a composite service.

The selection of a winning coalition merely depends on the benefit (total utility) its associated composite service proposal yields to the service consumer agent. The coalition with the highest utility wins. As a result, the rationality of service provider agents is bound by the QoS requirements stated in a composite service request because the total utility of a coalition is calculated based on its concrete QoS offering. Thus a service provider agent’s priority is to choose partners and form a coalition that optimises the imposed QoS constraints, and this may overrule local preferences.

An issue to be considered is the restrictiveness of the market-based mechanisms of the on-demand service composition concept. The lack of central control in an open market system opens loopholes for single or groups of software agents to take advantage of others.
4. ON-DEMAND SERVICE COMPOSITION IN OPEN ENVIRONMENTS

There are basically three cases to be considered:

1. Coalitions vs. service consumer agent

2. Coalitions vs. coalitions

3. Service provider agent vs. service provider agent within the same coalition.

In the first case, service consumer agents have no influence on the coalition formation process. Thus they cannot interfere with the interests of service provider agents. Since non-malicious behaviour is assumed, service provider agents merely fix SLAs for forming coalitions based on the propositions received from their coalition partners without tampering with the propositions. On the other hand, all service provider agents in an open market system could agree on price-fixing since a service consumer agent does not have knowledge about and influence over the optimality of coalition proposals. However, since only the coalition with the highest utility wins, price-fixing results in a problem similar to the game-theoretical Prisoner’s Dilemma problem (Axelrod [7]). If a coalition sticks to the price-fixing agreement, it shares the same chance of winning as all other coalitions. Hence all coalitions aim for winning and will, therefore, offer in such a situation very similar proposals in order not to minimise their own chances and not to thwart other coalitions. Similar offers and sharing the same chance of winning, however, take away the strategic advantage of a competitive offer. Such behaviour is against the inherent incentive of accumulating profit in a competitive market-based environment. If a coalition defects from the price-fixing agreement, it increases its chances of winning by simply under-pricing competing coalitions. Thus all coalitions would defect and price-fixing does not take place.

With respect to the second case, service provider agents might attempt to join multiple coalitions to increase their chance of winning or to spy. Spying is out of the question, since all service advertisements are known a priori to all service provider agents because they are sent via broadcast messages. However, an agent may join multiple coalitions in order to improve its chances, particularly in cases in which several coalitions with very similar proposals emerge. Due to the autonomy of service provider agents, it is unknown to a service provider agent if such a situation will occur. Thus it would need to join every possible coalition, resulting in a trade-off between the benefit of increasing chances and
4.3. DISCUSSION OF KEY CHARACTERISTICS

the costs incurred from additional communication and coordination. However, it cannot be ruled out. It is assumed in this thesis that service provider agents always only join one coalition at a time.

Thirdly, since non-malicious agent behaviour is asserted, there is no way service provider agents can take advantage of others within a coalition. Composite service proposals are created based on the coalition member’s service advertisements. Every coalition member has expressed its demands with its service advertisement prior to coalition formation. These service advertisements are fixed throughout an entire service composition process for a single composite service request. There are no coalition internal negotiations for the distribution of benefits that may create opportunities to take advantage.

Belief–Desire–Intention

The foundation of the BDI concept was laid with philosophical studies aimed at explaining practical reasoning phenomena with human attitudes such as beliefs, desires and intentions (Bratman [17]). Beliefs describe the knowledge of an agent (human or software) about its environment. Desires denote higher level goals to be achieved, and intentions represent partial plans of action that an agent commits to execute in order to fulfil a goal (Rao and Georgeff [141]). The BDI concept can be used to specify software agents as rational deliberate goal-oriented decision-makers. They pursue a goal and commit to actions accordingly until either the goal has been achieved or the goal cannot be achieved any more. While pursuing a goal, the beliefs of a software agent may change and thus affect the commitment to specific intentions. A software agent, therefore, may drop an intention in favour of an alternative more prospective solution path.

The on-demand service composition concept does not imply the implementation of a classical BDI architecture as presented for example in Rao and Georgeff [142] and Wooldridge [168]. It is rather inspired by the BDI concept in the way service consumer agents choose between alternative methods for forming plans and service provider agents choose between different potential partners in order to form coalitions. The beliefs of a service consumer agent are the user-defined service request, a set of methods for automated planning and the domain ontology of a specific problem domain (e.g. travel planning). The desire of a service consumer agent can be interpreted as the goal to provide its associated
user with a composite service instance for an active service request. Its intentions are actions a service consumer agent performs for decomposing the compound tasks of a service request into atomic tasks that can be executed with concrete service according to the user-defined requirements. The beliefs of a service provider agent are its local preferences, a composite service request and the service advertisements of other service provider agents. The desire of a service provider agent can be interpreted as the goal to join a winning coalition. Its intentions are sequences of actions for determining and contacting the most beneficial coalition partners with respect to the QoS preferences of a composite service request.

The BDI concept offers flexibility for software agents in open systems. Goal orientation and deliberation about internal states and the situation in the multiagent system allow software agents to adapt to changes. They are endowed with a number of alternative intentions that support the fulfilment of a goal under differing conditions. Therefore, the BDI concept implemented in single software agents contributes to the increased robustness of a whole multiagent system.

4.3.2 Advantages

The implications of the open market concept and the on-demand service composition concept for the design of a multiagent system and the behaviour of incorporated software agents accord with the approach taken in this thesis to characterising open environments. They are enumerated in the following list. In addition, the presented approach fosters beneficial service-oriented design principles as discussed in Section 2.5.3.

- **Flexibility**
  Since the approach in this thesis executes all activities of the service composition life-cycle (see Section 2.2) online, it offers great potential for effective adaptation to runtime changes in open environments, in particular to the availability of service providers and their associated services as well as changing user-defined functional and QoS preferences. The former is achieved with decentralised on-demand coalition formation to support the dynamic selection of appropriate services. The latter is achieved with automated service planning. Service consumer agents have sets of small composable plans (methods) at their disposal that can be flexibly combined
4.3. DISCUSSION OF KEY CHARACTERISTICS

at runtime to match the specific requirements of service requests.

- **Extensibility and reusability**
  The existing set of available methods can be dynamically extended if the requirements or characteristics of a problem domain change. Additional methods can be interleaved with existing methods in order to either further decompose existing methods or to add higher level alternatives that are decomposed by existing methods. The reuse of existing methods supports an iterative extension of the planning capacity of service consumer agents. Reuse increments the number of valid and proven task decompositions. It simplifies refinement and composition of composite service requests. It also reduces necessary effort for extending the service composition capacities of service consumer agents.

- **Robustness**
  Robustness is achieved in two ways. Firstly, with automated planning. The decomposition tree for a service request typically yields multiple generic plans. If there are no composite service proposals available for a composite service request based on one generic plan, an alternative generic plan may exist that can be used for creating a further composite service request. Secondly, local autonomous decision-making introduces redundancy that enables the multiagent system to perform even though a number of service provider agents are not available or even fail during a service composition process. The robustness of plan decomposition and the redundancy of decentralised decision-making reduce the risks of single points of failure.

- **Scalability**
  The decentralisation of data and control flows as well as decision-making, together with the absence of any central components or repositories, creates potentially high scalability. There are no central bottlenecks in an open market system. The on-demand service composition is, therefore, based on direct unilateral and bilateral communication between software agents without the incorporation of mediators or brokers. As such, the system’s scalability depends on the number of messages exchanged and is limited only by the capacities of the network that implements the communication channel of the open market concept.
4.3.3 Shortcomings

The advantages of the open market and on-demand service composition concepts are opposed by a number of shortcomings. These stem from the complexity of open environments and the lightweight approach of this thesis for modelling a multiagent system for dynamic service composition in open environments. The limitations are as follows:

- **Limited control**
  Open environments in general do not allow for central control. The lack of structure of the open market concept and the emergent non-deterministic behaviour of the on-demand service composition concept emphasise this fact. Thus a single software agent cannot determine for certain the properties of other software agents or the resulting multiagent system as a whole. The control of processes and their costs, described as two of the most important features of business process automation, cannot be guaranteed.

- **Incomplete outcomes**
  The termination problem of a service composition process is undecidable because of partial observability, dynamics and the stochastic character of the proposed approach to dynamic service composition. Therefore, a service consumer agent specifies a timeout with a composite service request that defines the latest time a service consumer agent will accept composite service proposals. The use of hard temporal constraints, however, affects the final results. Hence composite services are formed based on local decision-making and interactions between software agents. Both activities take time that cannot be unambiguously predetermined.

- **No guarantee of optimal outcomes**
  The system behaviour of an open market system is non-deterministic, limited by the availability and autonomy of service provider agents. The local preferences of different service provider agents can be conflicting and may prevent the creation of optimal outcomes. Optimality here refers to the compliance of a composite service proposal with respect to the QoS requirements of a composite service request. Note that the autonomy of software agents and the characteristics implied by an open
4.4 CONSTRAINTS AND ASSUMPTIONS

Environment constrain the optimality of computation in general. Open environments exhibit risk and chance.

- **High communication costs**
The wide use of broadcast messaging amongst software agents incurs high communication costs which pose a particular problem in networks with low bandwidths, e.g. wireless ad hoc networks. Broadcast communication is necessary to compensate for the lack of structure and control delineated above. A different concept for the dissemination of composite service requests and service advertisements may reduce communication costs, but it would also compromise the adaptation of the on-demand service composition concept to partial observability and dynamics in an open market system. The early accumulated communication overhead is later contained by the exchange of unicast messages for most interactions between service provider agents (see Chapter 9).

4.4 CONSTRAINTS AND ASSUMPTIONS

The concepts of the open market system and on-demand service composition are based on or introduce the following limiting constraints and assumptions for the design of software agents and a resulting multiagent system.

- This thesis is focused on activities centred on dynamic service composition only. It does not consider any further stages of the composite service life-cycle such as service enactment, monitoring or re-planning. The service composition process in this thesis terminates after the service contracting activity.

- It is imperative that all software agents share the same domain knowledge of a particular problem domain in order to ensure interoperability. A domain ontology is created a priori to specify that knowledge. The author acknowledges the problem of ontology merging. However, this is out of the scope of this thesis. All information exchanged during a service composition process is based on a single uniform domain ontology.

- Software agents obtain knowledge about their environment only through interaction
with each other. The knowledge is direct; it does not include information about third parties. The knowledge is time-limited and may become obsolete or wrong.

- All software agents are rational, truthful and non-malicious. This thesis does not address security and trust issues.

- A service consumer agent acts on behalf of exactly one client.

- A service provider agent provides exactly one service on behalf of exactly one service provider organisation.

- Service provider agents offer simple stateless services. Complex stateful services with complex message-exchange patterns are out of the scope of this thesis.

- Service provider agents are cooperative problem-solvers. If a composite service request exceeds the capabilities of single service provider agents, they interact to form coalitions in order to achieve a problem solution.

- Service provider agents are self-interested. They do not consider the payoff of a successful composite service proposal for other service provider agents. As such, service provider agents are not benevolent. They do not always adopt the goals of other software agents straight away but only if it is beneficial to themselves.

- A service provider agent joins exactly one coalition at a time. There are no overlapping coalitions.

- The decision-making of service provider agents is limited by the market-based mechanisms as outlined above.

- Service provider agents do not incorporate learning for long-term optimisations. The on-demand service composition concept of this thesis is focused on one-shot service composition processes without considering history.

- Service advertisements are fixed and binding throughout a service composition process with respect to a single concrete composite service request.
4.5 Summary

This chapter presents a concept for modelling open environments and a concept for on-demand service composition in open environments. These concepts together constitute the foundation of the approach to dynamic service composition in open environments of this thesis that will now be applied to the characteristics of open environments. The chapter also provides a discussion of techniques and principles that have inspired and guided the creation of the two concepts as well as expected advantages, drawbacks, constraints and assumptions. The focus of this chapter is merely on introducing and highlighting the characteristics of the open market and on-demand service composition concepts. Detailed semi-formal specifications of a concrete service composition method (based on dynamic coalition formation) as well as the static and dynamic models of a multiagent system implementing the on-demand service composition concept (including software agent design, data models, decision-making methods and interaction protocols) are presented in the next three chapters.
4. ON-DEMAND SERVICE COMPOSITION IN OPEN ENVIRONMENTS
Chapter 5

On-demand Coalition Formation in Open Environments

This chapter specifies a novel method for dynamic on-demand service composition in open market environments that is based on the concept of multiagent coalition formation. The chapter firstly provides an introduction to the game-theoretical concept of coalition formation in order to establish fundamental terms and notions (Section 5.1). Secondly, the chapter outlines relevant dynamic coalition formation approaches that are aimed at similarly dynamic and open environments (Section 5.2). Finally, the chapter defines the novel dynamic multiagent coalition formation model of this thesis. The model provides the theoretical foundation for the implementation of the previously defined service discovery, matchmaking and selection stages of the concept of on-demand service composition in an open market system.

5.1 Concept of Coalition Formation

Creating software systems that target the problem of service composition in open environments among autonomous self-interested peers is a challenging problem due to the disparate ownership of the software agents that act on behalf of the peers. These software agents perform actions according to their local goals and preferences triggered by fully autonomous decision-making with the aim at increasing local welfare.

Moreover, software agents do not possess full knowledge about each other due to
dynamics and partial observability of open environments. Firstly, software agents do not disclose private information about their internal state and perform decision-making based on potentially differing techniques. Secondly, the information flow between software agents may be interrupted by unforeseen network or communication errors. Thirdly, information itself may become obsolete. Thus a service composition approach aimed at service composition in open environments must be designed to address the intrinsic dynamics, uncertainty and non-determinism of the problem space.

If it is not possible to enforce the behaviour of software agents, the incentive for them to collaborate must be provided in a different way. This thesis incorporates market-based mechanisms that on the one hand entice software agents to collaborate for gaining profits and on the other hand guide agent collaboration to achieve efficiency with respect to global client-defined QoS requirements. Collaborations emerge in a bottom-up fashion solely through interactions among service provider agents. Such a bottom-up approach to service composition in an open market environment is equivalent to what is known in game theory as forming coalitions in cooperative games among a set of self-interested software agents.

Klusch and Gerber [93] define a cooperative game \((A, v)\) as a set \(A\) of software agents and a characteristic function \(v\) that assigns every subset of \(A\) its maximum payoff. Each subset is denoted a coalition \(C\) of software agents. The outcome of its characteristic function \(v(C)\) is also called its coalition value and this solely depends on the coalition members and not on coalition-external software agents or their actions. Coalitions are formed based on binding agreements on how to share the coalition value amongst the coalition members. The solution of a cooperative game is defined as a coalition configuration \((S, u)\) comprising a coalition structure \(S\), a valid disjoint partition of \(A\) into coalitions and a utility function \(u\) for payoff division that assigns every software agent in \(A\) its utility \(u(a)\) with respect to the coalition value of a software agent’s coalition in a concrete coalition structure.

The aim of a self-interested software agent in a cooperative game is to optimise its payoff. This is achieved by determining the most beneficial coalition structure \((S, u_{\text{max}})\) yielding the highest utility for a software agent \(a\). Without limiting generality it can be defined as \(\exists(S, u_{\text{max}})\forall(S, u) : u_{\text{max}}(a) \geq u(a)\). To calculate the most beneficial coalition structure, all software agents simultaneously execute a coalition algorithm \(CA\) in a
5.1. CONCEPT OF COALITION FORMATION

decentralised manner and negotiate with each other accordingly by exchanging messages.

According to Sandholm [147] a coalition algorithm typically comprises the following three steps:

- Generation of valid coalition structures
- Solution of the optimisation problem of each coalition
- Calculation of the payoff division.

A coalition algorithm is designed to provide stable coalition configurations for any given cooperative game. A coalition configuration is called stable if no software agent intends to leave a coalition because of its assigned utility. A coalition configuration is called Pareto-optimal if no software agent can gain higher utility with any different valid utility function for a particular coalition structure and coalition algorithm. Klusch and Gerber [93] finally outline two other relevant terms. A coalition formation environment denotes the assumptions and constraints that are invariably valid during any coalition formation activity in a given cooperative game, including problem domain related propositions about the functionality of software agents and particular methods for performing the three steps of a coalition algorithm. A coalition formation model denotes a coalition formation environment and a coalition algorithm for that environment.

The majority of existing coalition formation models are based on game-theory concepts for stable payoff division, such as the Core, the Kernel or the Shapley value (see Kahan and Rapoport [88]). Although they provide strong theoretical models they also inherit a number of limiting constraints from the game theory domain. Firstly, respective coalition formation environments are designed as closed systems which means that the number of software agents does not change over time. As such, these models are static and do not allow any dynamic changes during the coalition formation process. Secondly, it is assumed that all software agents in a cooperative game have complete knowledge about the capabilities and possible actions of other software agents, which implies a fully observable coalition formation environment. Thirdly, coalition formation environments are homogeneous. Thus all software agents operate according to the same coalition algorithm fully revealing the basis of their local utility calculation and without the option to object to join or to leave coalitions. This implies a deterministic system with the foremost
task for every software agent being to compute the optimal coalition structure based on a search in the space of all valid coalition structures of a given cooperative game.

The computational complexity of the search operation, however, is under certain limiting conditions polynomial (e.g. Shehory and Kraus [149]) but in most cases exponential (e.g Blankenburg et al. [14]) because there are $|A|^{|A|/2}$ possible coalition structures without limiting the generality (Sandholm [147]). Coalition formation based on the Core for example is aimed at optimising social welfare, which means the maximisation of the sum of coalition values of all coalitions of a coalition structure. In this respect, Sandholm [147] shows that at least $2^{|A|-1}$ valid coalition structures need to be evaluated in a cooperative game of a set $A$ of software agents in order to establish the worst case boundary for the optimum. Similarly expensive are coalition algorithms based on the Shapley value that derive the utility of a software agent from the value it contributes to a coalition averaged over all possible joining orders. Also the Kernel exhibits exponential computational complexity unless the size of coalitions is limited by a small constant value. This is because each software agent needs to compare its gain with the gains of every other software agent to ensure that no single software agent could get a higher payoff in a different coalition.

Klusch and Gerber [93] conclude that classical coalition formation models are not suitable for developing coalition formation approaches in open environments. Instead they explore the field of dynamic coalition formation (DCF) in order to investigate coalition formation methods that match the characteristics of open environments. The major constraints to be met by a dynamic coalition formation method are specified by Klusch and Gerber as the following:

- Dynamic creation of coalitions
- Software agents may enter or leave
- Set of tasks to be solved changes
- Lack of global control
- Incomplete knowledge
- Uncertainty about the preferences and decisions of software agents
5.2 RELEVANT COALITION FORMATION APPROACHES

- Temporal constraints.

The main aim of a dynamic coalition formation model is to provide a method for software agents to adapt to changes and uncertainty in a particular coalition formation environment without restarting coalition formation negotiations under the premise to enable beneficial outcomes for all coalition members. A complete restart of a coalition formation process is not only undesirable; it is in many cases even impossible, due to hard real-time constraints that limit the processing time for software agents to accomplish a given goal. Consequently, Klusch and Gerber [93] reason that each software agent must gradually adapt its decision-making to changes in its environment, e.g. by incorporating probabilistic reasoning, multi-criteria decision-making, reputation and trust measures or reinforcement learning. In the light of the complexity of dynamic coalition formation environments Klusch and Gerber suggest that it is also necessary to reconsider classical concepts of optimality and stability in the dynamic domain, since in most cases it is more important to quickly determine any type of coalition structure on time instead of searching for the optimal one, which may be computationally expensive.

Klusch and Gerber [93] note that developing coalition algorithms for dynamic coalition formation environments is a challenging task. No coalition algorithm has been proposed so far that is specifically developed for dynamic coalition formation. However, several coalition formation models are available in the literature that adapt classical coalition algorithms to dynamic environments. The most relevant of them are discussed below.

5.2 Relevant Coalition Formation Approaches

Klusch and Gerber [93] propose a dynamic coalition formation approach called the DCF-S scheme that supports dynamic coalition formation in environments in which:

- Software agents continuously receive a set of goals to be accomplished
- Software agents leave or join the environment at any time
- A set of so-called world-utility agents is available that maintain a record of registered agents, their capabilities, and the QoS ratings, provided by other software agents.
Klusch and Gerber [93] define a goal-oriented cooperative game \((A, v)|G\) of a set \(A\) of software agents and a characteristic function \(v\) which assigns every coalition its total expected outcome with respect to a single goal \(G\). Coalitions are represented by a coalition leader that acts on behalf of all coalition members, e.g. during coalition negotiation and payoff distribution. Since the number of members of a coalition is not predetermined, the smallest possible coalition is a single-agent coalition. In this case, which represents the initial setup of a game, every single agent is also a coalition with the only agent being the coalition leader. Hence a single agent forms a stable coalition with the aim of accomplishing one of its goals as a solution in the context of a given game.

Klusch and Gerber [93] extend the original three steps of a coalition algorithm with a fourth in an attempt to model uncertainty in open environments. Accordingly, every coalition leader performs the following four steps:

1. **Preparation**
   The preparation step is concerned with updating the local knowledge of a coalition leader, including the set of goals to be accomplished in collaboration with other software agents and information about the capabilities and QoS rankings of other software agents.

2. **Simulation**
   This step deals with the generation of valid coalition structures. A coalition leader simulates all feasible coalitions based on its local knowledge and estimates the trade-off between the expected payoff and the potential risk of forming a coalition. The coalition with the highest trade-off per goal is kept.

3. **Negotiation**
   During negotiation, a coalition leader solves the optimisation problem by engaging in multiple bilateral negotiations with all potential candidate agents for the most beneficial coalition per goal that has been successfully simulated before. The outcome of a negotiation is a binding agreement on the payoff division within a coalition. If a coalition fails or an event changes a coalition’s value or structure, the coalition leader stops the negotiation process for that coalition and restarts the processing at the simulation step in order to determine a new valid coalition. By doing so, the
coalition leader keeps software agents in the coalition for which an agreed contract already exists. A restart of the negotiation step is performed in order to complete a coalition.

4. Evaluation

Finally, a coalition leader evaluates the previous negotiations and updates the nearest world-utility agent as well as manages the payoff division amongst the coalition members according to the agreed contract. Klusch and Gerber propose the adoption of the bilateral Shapley value with equal or proportional shares for stable payoff distribution in super additive environments.

The outcomes of the DCF-S scheme for a given game cannot be guaranteed to be optimal which, of course, is expected from an approach to dynamic coalition formation in open environments. Klusch and Gerber [93] do not provide an evaluation of the optimality and stability of their approach for a specific coalition algorithm and cooperative game.

With regard to the coalition formation model proposed in this thesis, three features of the DCF-S scheme need to be discussed. First of all, the author believes that the assumption of the existence of world-utility agents is problematic in open environments for three reasons. Firstly, it seems to be very difficult to keep world agents up to date as with any central repository in open environments (as outlined before). Secondly, it is unclear how to prevent malicious manipulation of QoS ratings because this information is collected and distributed indirectly through a third party. The absence of any central control during bilateral negotiations makes it impossible, for example, to verify whether a negotiation outcome that has been reported belongs to an actual negotiation that has taken place before. Thirdly, world agents are outlined by Klusch and Gerber as passive data sinks. Thus they are very likely to violate the autonomy and proactiveness properties of the weak notion of agency. The coalition formation model of this thesis avoids the use of central repositories at all.

A second aspect is the assumption that a coalition leader negotiates with multiple potential candidates for a given coalition concurrently. The implication that these negotiations are independent does not hold in general in open environments in which different negotiation partners have local private preferences (e.g. concerning trust and reputation)
that may express strong objections against other specific software agents. Thus if a negotiation fails and the coalition leader attempts to negotiate a replacement, coalition members that have already agreed to join a coalition are bound by the agreement to remain in the coalition even if the change contradicts their local preferences. With the given model, there is no mean for the coalition leader to consider the preferences of coalition members, last but not least because the disclosure of private information is out of the question. The coalition formation model of this thesis forces coalition leaders to negotiate with only one negotiation partner at a time. This way, potential members have the choice to decide whether to join a coalition based on information about current coalition members. Moreover, coalition members and also the coalition leader are free to leave a coalition at any time as long as they do not explicitly confirm their coalition membership.

Thirdly, if different coalition leaders perform coalition negotiations concurrently, two coalition leaders may attempt to engage independently into negotiations with each other at almost the same time for the same goal. Klusch and Gerber [93] define the negotiation step only as proactive behaviour leaving out information about the reactivity of software agents; in particular how they coordinate to avoid concurrency phenomena such as life locks or dead locks or whether a coalition leader abandons its own agenda for joining the proposed coalition of a negotiation partner. Their approach is only valid under the assumption that different coalition leaders do not share the same goals. The coalition formation model of this thesis is designed for a coordinated execution of proactive and reactive behaviours while maintaining a consistent knowledgebase within a software agent.

Another high risk scheme for dynamic coalition formation is proposed in Soh and Tsatsoulis [154]. The approach is designed for coalition formation in multi-sensor target tracking networks in which software agents collaborate to perform triangulation operations for tracking target objects in their coverage area. Every software agent represents a sensor in such a network that is restricted by a number of constraints such as:

- Limited local computational resources
- Partial or outdated information about other software agents
- Network inconsistencies
- Hard real-time constraints.
5.2. RELEVANT COALITION FORMATION APPROACHES

Given the complexity of the environment, agents cannot perform complex computations in order to determine optimal coalitions. Rather, the focus of the scheme is on increasing the chance of coalition formation in a timely, robust and flexible manner that adapts to dynamic changes in the environment and the usage of limited time-bound resources. Coalition formation is performed dynamically in response to an event signalling the appearance of a target object and ceases if a complete coalition has been formed or the target object has left the coverage area.

All software agents are independent peers that autonomously perceive the environment, make decisions and form coalitions. They are organised in a neighbourhood structure in which direct neighbours communicate with each other. A software agent may belong to multiple neighbourhoods at the same time. If a software agent perceives an event, it prepares for a new coalition formation process. To do so, it relies on information about its neighbourhood as well as knowledge on how to decompose a problem into several tasks and what coalition algorithm is best-suited for a particular event.

The coalition formation model structures a coalition algorithm into two steps:

1. **Coalition initialisation**
   The initiator of a coalition prepares a ranking of potential candidate agents based on past and current relationships with neighbours as well as their effective capabilities. Past relationships express the likelihood with which neighbours accepted to join a coalition for a particular task in the past. Current relationships model currently active collaborations of a software agent with its neighbours in order to fulfil the tasks of previously arranged coalitions. The capabilities specify the ability of neighbours to perform specific tasks. This step results in a ranking of candidate agents according to their potential utility given the outlined criteria which forms the base for a tentative allocation of tasks of the coalition to candidate agents.

2. **Coalition finalisation**
   The initiator engages in negotiations with all candidate agents according to a case-based reasoning method. The software agents negotiate bilaterally whether a candidate is willing to perform a specific task of the coalition. Coalitions are successfully formed if all member agents have confirmed a coalition. Negotiations may fail due
to network shortages or because candidate agents reject coalition proposals. If the initiator fails to accomplish a successful coalition and therefore is not able to respond to an event properly, it returns to step 1 and re-evaluates its coalition with the aim of re-starting negotiations and determining a different successful coalition until the event is timed out. In addition, the outcomes of negotiations are used to update information about the relationships of an initiator with its neighbours. This is a simple form of reinforcement learning because the updates affect the ranking of candidate agents over time expressing which neighbours are more beneficial and should be contacted first in order to improve the rate of successfully formed coalitions.

Soh and Tsatsoulis [154] stress that their approach is merely concerned with forming any possible coalition because it is impossible to search for optimal solutions given the hard real-time and resource constraints of the given problem domain. Therefore, Soh et al. [155] present an evaluation of the scheme that fully neglects the optimality issue and discusses the rate of successfully created coalitions instead. The approach supports successful coalitions in only 20% of all coalition formation attempts. Failure is caused by broken negotiations in about 56% with the other 24% being associated with network volatility and resource shortages. However, the evaluation outcomes are difficult to estimate because they are not put into the context of how long events last. Soh and Tsatsoulis also do not explicitly discuss stability.

In general Soh and Tsatsoulis’s scheme has similarities with the coalition formation model of this thesis by being settled in a time-constrained, uncertain and dynamic environment. However, the two models differ in four ways. Firstly, this thesis does not consider resource constraints on software agents. Secondly, the coalition formation model presented in this thesis is not aimed at increasing the social welfare of the system (in contrast to the scheme by Soh and Tsatsoulis in which software agents strive to maximise the overall resource allocation in the network in order to track as many target objects as possible). The coalition formation model in this thesis incorporates self-interested software agents that act according to local private preferences in order to maximise local profits. Thirdly, the initiator agent adopts the role of a coalition leader comparable to Klusch and Gerber [93]. This coalition leader is in full control of the coalition formation process. It creates coalitions and decides on the payoff distribution autocratically. Hence
coalition members have no influence on the choice of coalition partners according to their local preferences. However, this is not an issue in the given scheme in which software agents are organised in a fixed network structure. Fourthly, Soh and Tsatsoulis’s scheme is based on the assumption that all software agents perform the same decision-making and reasoning. The coalition formation model in this thesis is not grounded on the assumption of agent homogeneity.

Kraus et al. [98] propose a coalition formation model for forming coalitions with uncertain information that is relevant for this thesis because it models coalition formation in a market-oriented fashion. Software agents in need of solving a complex problem adopt the role of requester business agents and publish requests for proposal which in turn trigger the formation of coalitions of service provider agents aimed at satisfying the request with a proper solution. The coalition formation process is constrained by two major restrictions. Firstly, the value of a task a single software agent can provide and its associated costs are private information and therefore inaccessible to other software agents. Secondly, the time for responding to a request for proposal is limited.

The focus of Kraus et al. is on the development of a coalition formation model that aligns with the given constraints but still enables local gains based on optimal, stable and fair coalitions. Even though local gains are the main drive for businesses in a market scenario, they must be compromised for two reasons: to arrive at a computationally feasible model and to guarantee stability and fairness. Hence stability in particular is essential in time-limited domains to foster the creation of successful coalitions on time in the first place.

The approach by Kraus et al. is centred on a trusted central manager agent that coordinates a Dutch-auction-like mechanism which is extended for coalition formation negotiations. The coalition formation process commences with the manager agent publishing available requests for proposals to all registered service provider agents. Each request is assigned a price – the payoff that can be gained from accomplishing a composite service. The auction is organised in rounds. The price of a request decreases by a well-known constant factor with every single round. To form coalitions, service provider agents need to engage in negotiations. These negotiations are also organised in rounds and are centrally controlled by the manager agent. There is exactly one negotiation round per auction
round. Every service provider agent has exactly one turn in one negotiation round that is randomly allocated to it by the manager agent.

During its turn, a service provider agent either reacts to a coalition proposal received in the previous round or it proposes its own coalition to potential partner agents. Service provider agents calculate potential coalitions based on their knowledge about the capabilities of other software agents as well as estimates of their expected costs in order to rank potential candidates and subsequently coalitions. The ranking is performed with the help of different simple heuristics, such as picking the coalition with the highest expected net payoff or picking coalitions for tasks with a low number of competitors.

If a coalition has been agreed upon, the initiator of the coalition submits a proposal to the manager agent. The manager agent then verifies the proposal (whether it can accomplish the requested composite service) and awards the winning coalition. If multiple valid proposals for the same composite service are received in one round, the manager selects the winner randomly. A proposal for a composite service is a binding agreement; breaches are heavily penalised. The payoff distribution is also performed centrally by the manager agent after the composite service has been successfully executed.

The approach by Kraus et al. supports stability due to its timing constraints. Software agents know the factor by which the coalition value decreases and thus are able to forecast and compare potential gains of a current round with the ones of future rounds. Consequently, software agents compromise and aim at stable coalitions early. Fairness is ensured with the central manager agent that controls the entire coalition formation process including payoff distribution. The reported results of an experimental evaluation show that the approach achieves local gains for service provider agents that are up to 70% of the gains of an optimal scenario with complete information and no timing constraints (depending on the chosen heuristic).

However, the central character is also a major drawback of this approach because it introduces a single point of failure. A coalition formation process might fail even though the manager agent is only temporarily not available (e.g. due to network outages). Despite the time constraint of the auction process, the approach does not support real-time coalition formation because time steps are discretised in rounds. Accordingly, it is not known a priori how long a coalition formation process takes because it depends on the
processing time per participating service provider agent and negotiation round. Kraus et al. [98] do not describe temporal constraints on the responsiveness of service provider agents. Therefore, it is unclear whether the approach can handle dynamically changing sets of service provider agents; e.g. what happens if a service provider agent fails during a calculation (thus is not able to de-register properly from the manager agent during an active auction/negotiation process)?

There are other related publications available that do not specifically refer to dynamic coalition formation. They do not fulfill the requirements for coalition formation in open environments but discuss other aspects of the coalition formation model relevant to this thesis. Cornforth et al. [31] for example propose a coalition formation approach that is centred on the concepts of market orientation and emergence. Software agents form coalitions by performing simple price-based positional bargaining in a decentralised bottom-up fashion until a coalition is formed. Every software agent possesses a start capital which is multiplied or exhausted depending on the performed actions over a number of iterations. The idea of investigating long-term effects of coalition formation in a market environment is attractive and listed as future work in this thesis. However, the given approach eliminates software agents with negative capital and creates new software agents randomly, which is unrealistic in a market of independent service providers. The results cannot be applied seamlessly. Furthermore, the approach is not modelled to deal with real-time and concurrency issues present in open environments. In addition, the approach by Cornforth et al. is solely concentrated on monetary aspects for forming coalitions and does not consider QoS requirements.

Finally the author would like to refer to two approaches that are similar to the coalition formation approach presented in this thesis because they refer to multiagent systems of autonomous decision-makers. These are provided by Brooks and Durfee [21] and Breban and Vassileva [20]. Both works describe scenarios in which a software agent has its own local utility and is free to join or leave coalitions. Both approaches aim at maximising the utility of single or groups of software agents and the reduction of coordination costs for long-term coalitions (e.g. in the context of e-commerce in Breban and Vassileva [20]). The coalition formation model of this thesis focuses in contrast on short-term on-demand coalition formation processes. Coalitions in this thesis are not a means to
5. ON-DEMAND COALITION FORMATION IN OPEN ENVIRONMENTS

reduce coordination costs or to improve the utility of groups of agents. Coalitions satisfy only single composite service requests and the proposed coalition formation model allows software agents to find short-term partners according to their local private preferences. The coalition formation process starts all over again for a new composite service request. Hence coalition partners do not remain in long-term structures. Coalitions may vary from iteration to iteration depending on the requested composite service and changing local preferences of participating software agents.

5.3 Model for Dynamic Coalition Formation

The coalition formation model of this thesis describes the rationale behind how service consumer agents and service provider agents interact and what actions they perform in order to transform a client-defined composite service request into a composite service instance. Hence on-demand service composition in an open market system is defined as a game \((A, v)\{CSR_d\}\) of a set \(A = \{a_0 \ldots a_n\}\) of service provider agents that engage in a coalition formation process in order to form coalitions that satisfy a composite service request \(CSR\) in a given time until deadline \(d\).

A service composition process is initiated with a new client-defined service request. A service consumer agent creates such a service request from client input and performs automated planning (see Section 7.3) in order to obtain one or multiple valid generic plans that match the user-defined functional and QoS requirements. The service consumer agent chooses a generic plan as the base for a composite service request. The subsequent dissemination of a composite service request triggers an on-demand coalition formation process among service provider agents.

Let \(P = \{t_0 \ldots t_m\}\) be a valid generic plan that is defined as a set of tasks \(t_i\). Hence the specification of a generic plan as a set of tasks is sufficient for coalition formation purposes here, and for two reasons. Firstly, because it is assumed that every task \(t_i\) corresponds to a specific service type in the shared domain knowledge of a particular application domain. Secondly, it is also assumed that every single task \(t_i\) has a unique identifier that unambiguously identifies a task with regard to the control structures of the generic plan of a composite service request. In simple words, coalition formation in this thesis is concerned with replacing tasks of a generic plan with concrete services.
5.3. MODEL FOR DYNAMIC COALITION FORMATION

The set $A$ of service provider agents for a composite service request forms dynamically. It is variable and may change over time with service provider agents joining and leaving the game. Given the characteristics of an open market environment, it is possible that there is an empty or insufficient set of service provider agents for a particular composite service request. Consequently, no composite service can be created in such case.

The set $A$ of all service provider agents can be partitioned according to the tasks $t_i \in P$ of a composite service request into $m$ subsets $A_i; A_i \subseteq A$. Each subset $A_i$ contains service provider agents providing a service that matches the service type of task $t_i \in P$. The partition of $A$ is complete ($\bigcup_{i=0}^{m} A_i = A$) because only those service provider agents join the game that can contribute to a requested composite service (and, therefore, have a chance to gain profits from it). A service provider agent $a_i$, even though providing only a single service, may belong to multiple subsets $A_j$ depending on the service types of tasks $t_i \in P$.

The aim of service provider agents in a given coalition formation process is to form coalitions $C = \{a_i : a_i \in A\}$ in which each of them provides a service for at least one specific task $t_i \in P$. Hence a coalition $C$ is complete if, and only if, it comprises exactly one service provider agent $a_i$ for every subset $A_j$ of the game ($\forall_{j=0}^{m} A_j \exists a_i : a_i \in C \land a_i \in A_j$). Since there may be service provider agents $a_i$ that provide a service for multiple subsets $A_j$, the cardinality of a coalition $C$ is equal to or lesser than the number of subsets $A_j$ ($|C| \leq |\{A_j\}| = |P| = m$).

A valid coalition structure $S = \{C_0 \ldots C_l\}$ for a given plan $P$ is a partition of the set $A$ of all service provider agents into a set of disjunct coalitions $C_i (\forall C_i \in S : C_i \subseteq A \land \bigcap C_i = \emptyset)$. Hence no service provider agent is a member of two or more complete coalitions. A coalition structure $S$ does not necessarily represent a complete partition of $A$ depending on the availability of service provider agents. The partition is complete ($S = \bigcup C_j = A$) only if all subsets $A_i \subseteq A$ are non-empty and of equal size $x$ ($\forall A_i : A_i \neq \emptyset \land |A_i| = x$). In all other cases there exists at least one service provider agent $a$ that is a member of a subset $A_i$ but not a member of any coalition $C_j \in S$. This fact states rather a property of the proposed coalition formation model than a problem, since only one coalition will eventually be selected as the winner by the service consumer agent.

Uncertainty and the dynamics of open environments impose harsh conditions on the
proposed coalition formation model and prevent a priori knowledge about the set of available service provider agents and possible coalition structures. Consequently, the proposed model cannot guarantee optimality for both requesting service consumer agent and participating service provider agents. It cannot even guarantee successful completion of a composite service request in general. The set $A$ of available service provider agents changes dynamically, with service provider agents joining and leaving a game. If the cardinality of $A$ or the partitioning of $A$ into subsets $A_i$ is insufficient, no valid coalition structures exist. In addition, the deadline $d$ limits the time available for forming complete coalitions. Time may be too short to allow a sufficient number of coalition negotiations or the participating service provider agents may not agree on coalitions due to their potentially conflicting local preferences. The latter issue constrains the proposed coalition formation model not only in ensuring completeness but also in guaranteeing optimal outcomes. Formed coalitions must be interpreted as an agreement on the best (not optimal) possible outcome for a composite service request under the impact of heterogeneous local decision-making and the given temporal constraints. In this context the proposed approach is similarly opportunistic like that of Klusch and Gerber [93] and Soh and Tsatsoulis [154]. The termination of the proposed approach is always ensured due to the specification of a deadline $d$ with every single composite service request.

The next subsections present the three steps of the coalition algorithm proposed in this thesis. The proposed coalition algorithm (as schematically depicted in Figure 5.1) does not specify concrete decision-making methods. Rather, it constitutes a generic procedure that guides reasoning and decision-making of service provider agents in an open market system. The reason for the generic character of the coalition algorithm is the characteristics of an open market environment in which participating parties cannot be forced to use particular decision-making methods. Thus the software agents of a resulting multiagent system are homogeneous. They have different means and ways to arrive at a decision.

5.3.1 Coalition Structure Generation

After a service provider agent receives a composite service request containing a generic plan and a sufficient set of service advertisements, it attempts to generate possible coalition structures. In general there are $|A|^{|A|/2}$ valid coalition structures in a cooperative game.
over all possible combinations over a set $A$ of software agents. This hyper-exponential search space is hard to handle. A service provider agent would need to try at least $2^{|A|-1}$ different coalition structures in order to establish a lower bound from the optimum (Sandholm [147]).

However, the automated generation of a generic plan from a user-defined service request by a service consumer agent yields beneficial implications on the proposed coalition algorithm. It can take advantage of the fixed number of tasks $t_i$ of a generic plan $P$ and their associated service types in order to reduce the complexity of the search space. In this respect, a service provider agent (hereinafter referred to as decision-maker) is only interested in generating coalition structures that contain coalitions of proper size $m = |P|$. These coalitions must be complete sets of service provider agents. Each of them provides a sufficient and necessary set of services to cover all tasks $t_i \in P$ $(\forall j=0^m \exists A_j \in C \land a_i \in A_j)$. Moreover, a decision-maker does not possess knowledge about the local preferences of other service provider agents (hereinafter referred to as candidates). The local preferences of candidates are variable and may change frequently. As a result, it is impossible for a decision-maker to evaluate the optimality of coalition configurations $(S,u)$ simply because utility $u(a_i)$ is unknown for any candidate $a_i$. Thus a decision-maker does not create a coalition structure search space but only

---

**Figure 5.1**: *UML state diagram illustrating the steps of the proposed coalition algorithm.*

...
checks all valid coalitions it can participate in, regardless of what other coalitions may be available. The reduction in complexity is significant.

In general, the number of possible coalitions \( c \) for a given plan \( P \) of size \( m \) is basically the product of the cardinalities of all subsets \( A_i \) as depicted in Equation 5.1. If the cardinality of a single subset is empty, \( A_i = \emptyset \), then intuitively there are no valid coalitions possible and the product is 0. The number of all possible coalitions \( c(a) \) for a single decision-maker \( a \) depends on the product of the cardinalities of all subsets \( A_i \) of which this service provider agent is not element of (Equation 5.2).

\[
c = \prod_{i=0}^{m} |A_i| \tag{5.1}
\]

\[
c(a) = \prod_{i=0}^{m} |A_i|, \forall A_i : a \notin A_i \tag{5.2}
\]

Hence the number of possible coalitions depends on the cardinalities of the subsets \( A_i \) as well as the number of tasks \( m \) of a generic plan \( P \). In order to compare the search space for a single decision-maker \( a \) of the proposed coalition algorithm and general cooperative games, the minimum \( c_{\text{min}}(a) \) and maximum \( c_{\text{max}}(a) \) number of possible coalitions are calculated according to Equations 5.3 and 5.4.

\[
c_{\text{min}}(a) = 1 \text{ if } \exists A_i \forall A_j : a \in A_i \land A_i \neq A_j \land |A_i| > 1 \land |A_j| = 1 \tag{5.3}
\]

\[
c_{\text{max}}(a) \leq \left\lfloor \left( \frac{|A| - 1}{m - 1} \right)^{m-1} \right\rfloor \text{ if } \exists A_i \forall A_j : a \in A_i \land A_i \neq A_j \land |A_i| = 1 \land |A_j| \approx \frac{|A| - 1}{m - 1} \tag{5.4}
\]

The minimum number of possible coalitions for a decision-maker \( a \) depends on the distribution of candidates over the subsets \( A_j \). The smallest number of possible coalitions for a decision-maker \( c_{\text{min}}(a) \) equals 1 if:

1. All but one subset \( A_i \in A \) contain only 1 element

2. The decision-maker \( a \) is a member of that subset \( A_i \).
The maximum depends not only on the distribution of candidates over the subsets $A_j$ but also on the number $m$ of tasks $t_i \in P$ (compare Table 5.1). The maximum number of possible coalitions for a decision-maker $c_{max}(a)$ is greatest if:

1. All but one subset $A_i$ have more than 1 element
2. The decision-maker $a$ is a member of that subset $A_i$
3. The set of candidates is nearly equally distributed over the remaining subsets $A_j$.

| Number of software agents $|A|$ | Maximal number of possible coalitions |
|-------------------------------|--------------------------------------|
|                               | Number of tasks $|P|$ | 2 | 3 | 4 | 5 | 6 |
| 3                             | 2 | 1 | - | - | - |
| 4                             | 3 | 2 | 1 | - | - |
| 5                             | 4 | 4 | 2 | 1 | - |
| 6                             | 5 | 6 | 4 | 2 | 1 |

The maximum number of possible coalitions for a decision-maker reveals exponential complexity of the search space. This complexity of the proposed coalition algorithm is, however, significantly smaller in comparison to general cooperative games as illustrated in Table 5.2. The reduced complexity of the coalition structure generation step is important in the context of open market environments. Service provider agents gradually join and may also leave a coalition formation game. Every such event affects the coalition structure, potentially causing a restart of a coalition algorithm.

### 5.3.2 Solution of the Optimisation Problem

After a decision-maker has obtained possible coalition structures, or more specifically possible coalitions, it attempts to solve the optimisation problem of finding the coalition yielding maximum utility for it. Information for utility calculation is drawn from the QoS requirements of the composite service request and the service advertisements
Table 5.2: Comparison of the complexity of the search space of possible coalition structures for a single service provider agent between a general cooperative game and a game based on this thesis’s coalition formation model for games of different size.

| Number of software agents $|A|$ | General cooperative game | Model of this thesis |
|---------------------------------|---------------------------|----------------------|
|                                 | Maximal number of coalition structures $|A||A|/2$ | Minimum number to be checked $2^{|A|-1}$ | Maximum number to be checked $\left\lceil \left(\frac{|A|-1}{m-1}\right)^{m-1} \right\rceil$ |
| 2                               | 2                         | 2                    | 1                     |
| 4                               | 16                        | 8                    | 3                     |
| 8                               | 4096                      | 128                  | 12                    |
| 10                              | 1.00e+05                  | 512                  | 27                    |
| 50                              | 2.98e+42                  | 5.63e+14             | 6.74e+07              |
| 100                             | 1.00e+100                 | 6.34e+29             | 6.55e+15              |

a decision-maker receives from candidates during a coalition formation game. A service advertisement contains information about the service a candidate provides and the QoS attributes and values it is willing to guarantee.

The dynamics of the open market environment imposes a problem on the calculation of coalition values. Different candidates may join and leave a coalition formation process any time. Every such action affects the utility of possible coalitions and thus forces a decision-maker to restart its coalition algorithm. Such restarts may occur frequently. Unfortunately, the complexity of the search space of possible coalitions is still exponentially large. Thus it is crucial to avoid a full restart of the coalition algorithm. Or, if this is not possible, it is vital to develop a method that makes a restart computationally affordable.

To sort out this issue, the well-defined structure of a given generic plan $P$ is of help again. Instead of searching the space of all possible coalitions, a decision-maker uses the plan-induced partition of the set $A$ of all candidates into subsets $A_i$ in order to only search (re-calculate) the one subset of candidates that is affected by an event. This is possible because the utility of a coalition is maximal, if the aggregated utility of its members is maximal. This means that, a decision-maker determines a most beneficial candidate per subset $A_i$. Accordingly, a decision-maker calculates and ranks the utility of all candidates
(based on their advertisements) separately for each subset $A_i$.

The utility calculation for a candidate is driven by a market-based mechanism. The foremost incentive of any service provider agent is to maximise the QoS requirements of a composite service request because only this way it increases its chance to be member of a winning coalition (which is decided by the requesting service consumer agent). The QoS requirements are given per task and specified with weights that express the level of importance of each particular quality. Since there are no total values, the utility a candidate yields must be calculated relative to all other candidates of a specific subset $A_i$ in order to make different candidates comparable based on their respective service advertisements. Consequently, if there is a subset $A_i$ containing only one candidate, the utility this candidate yields equals 1 (due to the lack of alternatives).

In addition to these global requirements, every decision-maker is free to calculate the utility of candidate agents by incorporating further measures, such as local preferences (e.g., for expressing trust and reputation), interaction behaviour (expressing the probability of forming successful collaborations), etc. Section 7.5 proposes a possible method for utility calculation based on multi-attribute expected utility theory.

Ranking only the candidates per subset $A_i$ instead of directly calculating the utility of all possible coalitions results in a significant reduction in the complexity of the calculation of the optimal coalition. The maximum number $n_i$ of candidates in a subset $A_i$ can be effectively approximated with $n_i \approx n$ according to $n_i = |A_i| \leq |A| - 1$ when considering the worst case scenario. The effort of a decision-maker for the relative calculation of the utilities of all candidates in a subset $A_i$ can be estimated as the sum of:

- A maximum of $2 \times l \times n$ comparisons for determining the minimum and maximum values for each of the $l$ QoS attributes over $n$ candidates
- A constant cost factor $e \times n$ for calculating the utility of every single candidate
- And $n \times \log_2 n$ comparisons for calculating the ranking of all candidates with a divide-and-conquer sort algorithm (e.g., merge-sort).

The first two summands can be ignored because the factors $l$ and $e$ are fixed constants for a given composite service request. They do not increase the computational costs beyond linear complexity. Consequently, the worst case upper bound for determining the
most beneficial coalition of a decision-maker for a given generic plan $P = \{t_0 \ldots t_m\}$ of size $m$ depends only on the accumulated costs for ranking the candidates of every subset $A_i$ and the number of tasks $m$ without limiting the generality, as shown in Equation 5.5.

$$\sum_{i=0}^{m} n_i \times \log_2 n_i \leq m \times n \times \log_2 n$$ (5.5)

Moreover, runtime events such as service provider agents leaving or joining a game affect not all but only one of the subsets $A_i$ of candidates. Hence an event does not trigger the complete restart of the coalition algorithm, but only the re-calculation of the ranking of the affected subset $A_i$. The result is simple logarithmic complexity for a service composition game with $n$ service provider agents, as indicated by Equation 5.6.

$$n \times \log_2 n$$ (5.6)

The adjustment of the proposed model to the characteristics of an open market environment results in a drastic reduction of the complexity of the optimisation problem. The presented approach requires only logarithmic complexity in comparison to the exponential complexity $2^{n-1}$ of a general cooperative game.

Necessary partial restarts of step 2 of the coalition algorithm are significantly less expensive, even if events frequently interrupt a coalition formation game. This is a very important prerequisite to enable service provider agents to react to composite service requests in a timely fashion given the hard real-time constraints set by deadline $d$. Furthermore, the proposed approach achieves a higher scalability in comparison to general coalition formation models. Due to the reduced complexity, it is possible to accommodate larger sets of service provider agents (compare Table 5.2). The advantage of the specialisation of the proposed coalition formation model to match open market environments results, of course, in a loss of generality.

The proposed coalition formation model does not support the concept of stability in its classical meaning because a decision-maker lacks sufficient information to calculate the utility a given coalition structure $S$ yields to other service provider agents. In addition, a decision-maker cannot determine a priori whether a coalition partner has the incentive to remain or leave a coalition, because the partner’s incentive can change at any point in time (local preferences of service provider agents are private information and are potentially
5.3. MODEL FOR DYNAMIC COALITION FORMATION

pair-wise contradictory). The commitment of coalition members to remain in a coalition is explicitly determined through interactions with the coalition leader. In this context, a coalition is stable if the deadline for a composite service request has not yet passed and the coalition is complete (all coalition members explicitly committed to the coalition). This is stability by agreement.

The explicit commitment of a coalition partner can be interpreted as following. Firstly, a partner expresses that it has found the currently most beneficial partners available for it. Secondly, it signals its willingness to form a binding agreement, if the coalition leader manages to send out a composite service proposal to the requesting service consumer agent on time. If the composite service proposal arrives on time, all members and the leader of a coalition are bound by it. At this point, no agent can leave without incurring severe punishment, such as negative trust and reputation ratings or monetary fines. If the composite service proposal is sent after the deadline, a coalition disintegrates because the coalition formation process terminates.

5.3.3 Payoff Distribution Calculation

In general, service provider agents join coalitions because they expect to gain profits from them. The gain or payoff of a service provider agent depends on the coalition value $v(C)$ of the coalition it joined. Therefore, software agents strive to join the coalition structure yielding the highest coalition value for themselves (local welfare) or for all software agents (social welfare). The coalition value is usually modelled with monetary units.

In the proposed coalition formation model, the coalition value $v(C)$ depends on a multi-attribute utility function $f_{qos}$ over all QoS requirements of a composite service request. The strive to maximise $f_{qos}$ can be interpreted as the attempt to gain maximal profit by maximising the given client-defined QoS constraints. The optimal solution is accomplished in an ideal case in which all service provider agents share the same coalition algorithm without considering any local preferences. In reality, service provider agents are self-interested entities that have local preferences towards their coalition partners or QoS constraints for example. Local preferences overlie the global client-defined QoS constraints and thus may prevent an optimal solution.

On the other hand, service provider agents gain profits only if they are a member of the
5. ON-DEMAND COALITION FORMATION IN OPEN ENVIRONMENTS

winning coalition for a given composite service request. The winning coalition is decided by the requesting service consumer agent based on a rational decision according to the specified QoS requirements of a composite service request. Service provider agents that did not make the winning coalition do not gain anything because they don’t get contracted by the service consumer agent to provide the requested composite service. This fact states a dilemma for service provider agents that need to determine and join the optimal solution (from a service consumer agent’s point of view) to ensure the gain of some sort of profit beyond doubt. However, a service provider agent cannot gain necessary information to do so, due to the partial observability and dynamics of an open market environment.

The actual payoff distribution of the proposed coalition algorithm is very simple. Every service provider agent $a$ of a winning coalition receives the payoff $p(a)$ based on the demand $\text{costs}(a)$ it has stated with its service advertisement. All other service provider agents receive no payoff (see Equation 5.7). Since service advertisements are fixed for a particular service composition process, the payoff $p(a)$ of a service provider agent is the same for any coalition structure at any time during a single coalition formation process and consequently, no calculation for payoff division is necessary.

\[
p(a) = \begin{cases} 
\text{costs}(a), & \text{if agent } a \text{ in winning coalition} \\
0, & \text{otherwise}
\end{cases} \quad (5.7)
\]

Payoff distribution is relative. Every service provider agent of a coalition receives a payoff equivalent to what it contributes to a coalition because all other coalition partners apparently accepted that the service provider agent would provide a given task according to the QoS attributes (including costs) of its service advertisement. In other words, a service provider agent utilises an acceptable utility and payoff demand relative to the local preferences of its coalition partners and the global QoS requirements of a composite service request. Payoff distribution is fair because every participant of the winning coalition receives exactly the payoff it expected, which is fixed in a binding contract between service consumer agent and the coalition partners.

Given the proposed payoff division scheme, service advertisements are very important. A service provider agent needs to choose a set of QoS attribute values that it believes are competitive enough to enable its participation in the winning coalition but still satisfies
a payoff that provides a reasonable return on investment (given the communication costs and computational resources used during coalition formation and perhaps later service execution). A service provider agent is likely to consider its current working load as well as historical data and perhaps reinforcement learning methods to determine a proper advertisement for a given composite service request. The handling of these issues is out of the scope of this thesis and considered future work.

5.4 Summary

This chapter presents a novel method for on-demand service composition in open market environments that is based on the concept of dynamic coalition formation. The proposed coalition formation model is strongly adapted to the complex characteristics of open market environments. It takes advantage of pre-defined structures of the on-demand service composition concept in order to reduce the combinatorial complexity of the search space, to reduce the computational complexity of the coalition algorithm and to avoid full restarts of the coalition algorithm. The aim of these adaptations is to allow dynamic service composition under the uncertainty, dynamics and real-time constraints of open market environments. The specialisation of the proposed coalition formation model reflects its adaptation to the characteristics of open environments:

- Preserves strong agent autonomy (no central control)
- Models uncertainty about the local preferences of decision-makers
- Handles incomplete knowledge
- Supports dynamic sets of decision-makers
- Guarantees termination of a coalition formation process
- Guarantees a fair payoff division (market-based control)
- Provides low computational complexity (incorporates real-time constraints)
- No support of classic stability (stability by agreement)
- No guarantee of optimal outcomes
5. ON-DEMAND COALITION FORMATION IN OPEN ENVIRONMENTS

- No guarantee of completeness (that a valid solution can be found).

The proposed coalition formation model embodies a generic framework that guides the interactions and decision-making of service provider agents. The interactions during a coalition formation process are based on a set of interaction protocols that are outlined in Section 6.4. A set of concrete decision-making methods is specified in Chapter 7. Those can be integrated into the framework for implementing different decision-making activities, e.g. automated planning, automated matchmaking, expected utility calculation. Klusch and Gerber [93] postulate that the complexity of a coalition algorithm for dynamic coalition formation depends on the implementation of concrete interaction protocols and decision-making methods and their complexities. Accordingly, Chapter 9 investigates the properties of the proposed coalition formation model for dynamic service composition in open environments with an empirical evaluation.
Chapter 6

Multiagent System Design

This chapter outlines the static and dynamic design of a multiagent system that implements the approach to on-demand service composition in open environments set out in this thesis. The chapter opens with the presentation of the overall architecture of the multiagent system including incorporated software agent types and roles, the relationships between software agent types and general constraints. The next part defines a common vocabulary in the form of a set of fundamental data structures that software agents utilise for processing and exchanging information. The final part of this chapter specifies a set of well-defined interaction protocols that embody the foundation for inter-agent communication. The multiagent system presented is a light-weight approach to the design and implementation of a proof-of-concept prototype of the concept of on-demand service composition in open environments outlined in Chapter 4.

6.1 Multiagent System Architecture

The concept of on-demand service composition in open environments (see Chapter 4) is transformed into a multiagent system as depicted in Figure 6.1. The derived multiagent system architecture is surprisingly simple in contrast to the complexity of open environments in general and the concrete problem of dynamic service composition in open environments in particular. The light-weight character of the multiagent system architecture stems from the attempt to design a minimalist software system that incorporates only the minimal set of necessary and sufficient structural elements for enabling the imple-
mentation of the concept of on-demand service composition in open environments. Any limiting constraints on the underlying infrastructure and the multiagent system structure itself are avoided in order to meet the characteristics of open environments.

Figure 6.1: *Generic architecture of a multiagent system that models on-demand service composition in an open market environment.*

Therefore, the multiagent system only incorporates software agent components that represent the actors of the open market concept (see Section 4.1). Moreover, the multiagent system does not incorporate rigid pre-defined structural dependencies between these software agents in order to reflect the intrinsic decentralisation, uncertainty and dynamics of open environments. Dependencies and relationships between software agents are time-bound. They are established and disbanded only at runtime, in an on-demand and agent-based fashion through communication between interacting software agents.

The multiagent system comprises two software agent types, each with a different persona, for representing either the supply or the demand in an open market environment. On the one hand, demand is represented by service consumers that are modelled with service consumer agents. A service consumer agent is a software agent that acts on behalf of exactly one service consumer (person or organisation). Service consumer agents support
their associated actors in stating service requests and possess capabilities for automated planning and decision-making in order to generate composite service requests and evaluate composite service proposals. On the other hand, supply is represented by service providers that are modelled with service provider agents. A service provider agent is a software agent acting on behalf of exactly one service provider (person or organisation). Service provider agents actively engage in the service composition process by promoting electronic services and collaborating with each other, with the aim to satisfy service requests and to gain local profits. One service provider agent is associated with exactly one service entity.

The overall multiagent system architecture is generic. It does not specify concrete technologies neither for the implementation of software agents nor for their interaction protocols. The architecture solely defines the communication among software agents to be based on a high-level agent communication language (ACL), e.g. FIPA-ACL [51], KQML [97] or KIF [89].

Besides inter-agent communication, the multiagent system also defines interactions between external actors and external service entities on the one hand, and associated software agents on the other hand. The interaction between actors – service consumers and service providers – and their associated software agents may be established with any form of programming interface (from console to graphical user interface) and can be implemented with mechanisms for the direct manipulation of software agent components or via dedicated client applications. The communication between a service provider agent and its associated service instance may be established with different technologies (e.g. Web-services technology, CORBA or native programming language remote-method invocations) depending on the characteristics of existing concrete service infrastructures.

The design decision to use bidirectional unary associations between service entities and service provider agents does not limit the generality of the presented approach. It is merely a simplification in order to focus this thesis on aspects of inter-agent communication rather than agent-internal design issues. (As Henderson-Sellers and Gorton [67, p3. footnote 2] note: “The interactions between autonomous components are the essence of multi-agent technology”.) In fact, any extension to associate more than one service entity with a single service provider agent is possible without implications on the overall architecture of
6. MULTIAGENT SYSTEM DESIGN

the multiagent system and the processing of the on-demand service composition concept of this thesis.

The multiagent system architecture presented can be understood as a core part of the composition layer of the extended SOA model (see Section 2.1.1) and hence as an integral part of an SOA initiative. However, this thesis is centred on aspects of dynamic service composition following a multiagent-based approach based on the good matching of the characteristics of multiagent systems and open environments (see Section 2.4.3). Hence a purely agent-based perspective on the problem space is provided. Relevant service-oriented design elements are incorporated into the multiagent system in the form of software agents only in accordance with the interpretation of the weak notion of agency presented in Section 2.4.2. If such a direct incorporation is not possible, software agents are designed to act as autonomous and proactive representatives of these design elements. As such, service consumer agents represent service consumers and service provider agents act on behalf of service providers or service components. These software agents possess intrinsic knowledge about the roles, properties and capabilities of their associated actors and design elements.

The multiagent system is not intended to model more than the presented design elements of a potentially embedding service-oriented system. Moreover, the multiagent system architecture is designed according to the constraints and assumptions outlined in Section 4.4.

6.2 Software Agent Roles

The proposed multiagent system incorporates two agent types: service consumer agents and service provider agents. A set of roles for each software agent type is defined in the following. Concrete instances of both types adopt these roles during the on-demand service composition process. In this context, the property of agent autonomy (see Section 2.4.1) insists that a software agent only changes its role based on autonomous decision-making according to its local knowledge and preferences and with respect to its beliefs about its relationships to other software agents. A software agent can adopt only one role at a time but may adopt different roles autonomously at any time.
6.2. SOFTWARE AGENT ROLES

6.2.1 Service Consumer Agent Roles

Service consumer agents are always in one of the four roles depicted in Figure 6.2. Their roles can be modelled as states of an agent-internal finite state machine (see Section 7.2.1).

![UML class diagram of the class hierarchy of service consumer agent roles.]

AnySCARole Role

*AnySCARole* is an abstract superclass that does not model or exhibit any particular behaviour. It is the superclass of all other service consumer agent roles.

Listener Role

The *listener* role is adopted by a service consumer agent for interacting with its associated actor during the creation of a new service request. The listener role is a direct subclass of anySCARole.

Planner Role

After receiving a complete service request from its associated actor, the service consumer agent changes to the *planner* role. It then performs automated planning in order to generate a valid composite service request. The planner role is a direct subclass of anySCARole.

Requester Role

A service consumer agent acts in the *requester* role for all interactions with service provider agents during the on-demand coalition formation-based service composition process for a given actor-defined service request. The requester role is a direct subclass of anySCARole.
6. MULTIAGENT SYSTEM DESIGN

Contractor Role

A service consumer agent acts in the contractor role if it has received a sufficient number of composite service proposals. In this role, a service consumer agent mediates between its associated actor and the involved service provider agents in the leader role to leverage the formation of a mutual agreement between the actor and one winning coalition (composite service proposal). The contractor role is a direct subclass of anySCARole.

6.2.2 Service Provider Agent Roles

Service provider agents always adopt one of four roles depending on their relationships to other service provider agents. The roles are organised in a role hierarchy as presented in Figure 6.3. They can be modelled as states of an agent-internal finite state machine (see Section 7.2.2).

AnySPARole is an abstract superclass that does not model or exhibit any particular behaviour. It is the superclass of all other roles.

Role Idle

The idle role models an inactive service provider agent. A service provider agent in this role merely waits for new composite service requests. It does not have any active
6.2. SOFTWARE AGENT ROLES

relationships or dependencies with other service provider agents. The idle role is a direct subclass of anySPARole.

**Engaged Role**

The *engaged* role denotes an abstract role defining a service provider agent actively engaged in an on-demand coalition formation-based service composition process. This means that a service provider agent has already received a composite service request and it intends to engage or is engaging in decision-making and communication activities aimed at forming a successful coalition. The engaged role is a direct subclass of anySPARole. Since it is abstract, it needs to be instantiated with one of the following subclasses.

**Candidate Role**

A service provider agent acts in the *candidate* role if it has successfully matched its own service offer with a composite service request and has already disseminated its service advertisement in the multiagent system. As such, the candidate role expresses the intention of a service provider agent to actively engage in interactions with other service provider agents to form successful coalitions. However, a service provider agent in the candidate role has not joined any coalition yet and thus does not have direct and active relationships or dependencies with other service provider agents. The candidate role is a direct subclass of the engaged role.

**Leader Role**

The *leader* role is adopted by a service provider agent when it has successfully initiated its coalition and taken the lead position. The leader role implies that a service provider agent is responsible for extending its coalition and for controlling the interactions among existing coalition members. A service provider agent ceases being in the leader role if it disbands its coalition (e.g. in order to join another coalition) or if it believes that all coalition members have cancelled their participation in its coalition. A service provider agent in the leader role cannot change its role if it is part of a complete coalition that has submitted a composite service proposal to the requesting service consumer agent. In this case it must wait for feedback from this service consumer agent before considering a
change of its role. The leader role is a direct subclass of the engaged role.

**Member Role**

A service provider agent acts in the *member* role if it has joined a coalition and accepts the leader of the coalition and all existing (if any) participating coalition members. A service provider agent can cease being in the member role at many stages during an on-demand service composition process, e.g. when joining another coalition, when becoming leader of its own coalition, etc. However, if a service provider agent in the member role has confirmed its participation in a coalition for completing a composite service proposal, it must wait for feedback from the leader of that coalition before deliberating about changing its role. The member role is a direct subclass of the engaged role.

### 6.3 Data Structures

This section specifies the fundamental data structures that service consumer agents and service provider agents utilise to exchange information during on-demand service composition processes. The data structures focus solely on ensuring interoperability among the software agents of the multiagent system. Hence autonomous software agents are free in how to model and process data locally, e.g. in accordance with the requirements of their specific decision-making techniques and concepts. Thus the following data structures are not aimed at and do not constrain the agent-internal data representation.

The defined fundamental data structures are concerned with the representation of information about

- Composite service requests
- Service advertisements
- Composite service proposals.

There are various ways of defining data structures to represent the above information. Since the multiagent system architecture in this thesis is generic, the specification of data structures should also be generic. This way, the generality of the presented approach is not limited.
6.3. DATA STRUCTURES

The modelling of data structures in this thesis is inspired by Semantic Web technology (e.g. see Daconta et al. [33]), in particular, by the concepts underlying the OWL-S specification (Martin et al. [115]). OWL-S is a concrete ontology which defines the structured description of information about the syntax and semantics of Web services. It enables automated reasoning about the functionality, structure and interoperability of Web services. OWL-S is aimed at fostering the dynamic publication, discovery, composition and execution of Web services. It is defined on top of the Web Ontology Language (OWL). OWL in general is a semantic markup language developed for describing and sharing ontologies on the Web. OWL has gradually evolved as an extension of earlier efforts for knowledge representation and knowledge sharing, such as RDF, KIF and DAML-OIL.

This thesis does not include an extensive introduction to OWL and OWL-S. The reader is referred to Lacy [101] for a comprehensive introduction into OWL as well as to Dean et al. [34] and to Martin et al. [114] for W3C recommendations of the OWL and OWL-S standards respectively.

The aim is to keep the fundamental concepts of this thesis agnostic to any particular technology. Therefore, only those elements of OWL-S that are of relevance for modelling data structures in this thesis are discussed below. Moreover, the data structures in the following are specified in a generic fashion using the Extended BNF (EBNF) [43] notation.

6.3.1 OWL-S Process Model

The foundation of OWL-S is a process model. This process model represents an abstract specification of the signature and the behaviour of a process that transforms some input into some output. Figure 6.4 depicts the OWL-S process model as specified in Martin et al. [114]. The signature of a process is defined with input parameters and output parameters. The behaviour of a process is specified with preconditions and effects which express constraints on the

- Input and output parameters of a process
- Transformation of the internal state of a process
- Impact of the transformation of a process on the external world state.
Preconditions are denoted in as *conditions*. Effects are denoted as *results* in order to emphasise the conditional dependency of effects on concrete outputs.

**Figure 6.4:** OWL-S Process Model as specified in Martin et al. [114].

OWL-S introduces three types of processes. An *atomic process* embodies a single atomic stateless action of a service. That means that an atomic process expects only a single incoming request message (its input) which is transformed in a single service invocation to produce the full output. Atomic processes have no subprocesses. In contrast, a *composite process* defines a more complex behaviour of a service that can be understood as a stateful and ordered processing of multiple inputs according to a complex message exchange pattern. Composite processes have subprocesses that are linked according to *control constructs*. Finally, a *simple process* represents an abstraction of either an atomic or composite process that is useful when either a specialised view of an atomic process or a simplified representation of a composite process is required, e.g. for automated service discovery or automated planning and service composition.

The OWL-S process model contains further elements that are out of the scope of this thesis. This thesis departs from the original upper ontology of OWL-S that places
the process model of a service in a set with two further components: the service profile and the service grounding. The service profile describes ‘what a service does’. It provides information that is required for humans to discover Semantic Web services and that is used in service advertisements. This thesis does not consider the case of humans publishing and discovering services. Thus service advertisements are expressed using the notion of a simple process and some additional information instead. As mentioned before, a simple process can encapsulate information about both an atomic and a composite service in an abstract and generic fashion. Service grounding provides concrete details about how to connect and interact with a Semantic Web service, including message formats, port types, message bindings, etc. This thesis focuses on the service composition stage only. Therefore, a service grounding is not required per se. It is assumed that service grounding information is available in one form or another. It is also assumed that service provider agents encapsulate and hide the potentially complex interactions with their associated service entities during on-demand service composition processes because they act as their proxies.

Finally, the original OWL-S process model does not require processes to have input parameters. To the author, that is a contradiction of the general definition of a process as a transformation of some input into some output. Therefore, processes in this thesis must always have at least one input parameter and one output parameter.

6.3.2 Atomic Processes

The next step is to define an atomic process in EBNF as shown in Listing 6.1. Atomic processes are used in this thesis to model atomic services. An atomic service has a simple stateless process model without any further subprocesses. Atomic processes may also appear as subprocesses in composite processes. An atomic process is defined with a mandatory unique identifier (id), at least one input parameter and at least one output parameter. An atomic process may also have one or many QoS parameters, one or many preconditions and one or many effects. The input and output parameters are derived from the superclass parameter which is defined with a mandatory unique identifier, a reference to the parameter’s class type and a placeholder for a concrete instance (value) of the given class type. A qos parameter is also derived from the superclass parameter. Moreover, it
6. MULTIAGENT SYSTEM DESIGN

has an additional weight attribute that expresses its level of importance with respect to client-defined preferences. It has a value in the interval $[0, 1]$. Preconditions and effects are functional and/or logical expressions. They will be defined later.

Listing 6.1: Extended-BNF specification of the structural definition of an atomic process based on the OWL-S process model.

```plaintext
1 atomic-process = id, input, output, qos, preconditions, effects;
2 id = ?any unique identifier in the context of a parent element?;
3 input = {parameter}+;
4 output = {parameter}+;
5 parameter = id, class, value;
6 class = ?class type referring to a concept in a domain-specific ontology?;
7 value = ?placeholder for a concrete instance of the class type?;
8 qos = {qos-parameter}*;
9 qos-parameter = parameter, weight;
10 weight = ?real number in the interval [0, 1]?;
11 preconditions = {sentence}*;
12 effects = {sentence}*;
```

6.3.3 Composite Processes

The definition of a composite process extends the notion of an atomic process with a mandatory control construct and a set of bindings, as depicted in Listing 6.2. A control construct defines how the immediate subprocesses of a composite process are organised. According to the Structured Program Theorem [15], there are in general only three elementary structures, namely sequence, selection and loop, necessary for implementing any computable function. The definition of a composite process according to the OWL-S process model supports additional control structures beyond the minimum. However, the list of available control constructs is not extensive. It may be extended in the future.

It is important to note that a composite process always contains only one control construct. This control construct includes further composite processes and/or atomic processes according to the specific layout of its associated control structure (see Listing 6.2). The actual nature of specific subprocesses is hidden from a composite process. Accordingly, composite processes can be used to create hierarchical structures for modelling
6.3. DATA STRUCTURES

any kind of computable function. Composite processes support split and any control constructs in addition to the aforementioned elementary structures. The split construct provides a model for concurrency between different subprocesses whereas the any construct enables the execution of subprocesses in any arbitrary but strictly sequential order.

Listing 6.2: Extended-BNF specification of the structural definition of a composite process based on the OWL-S process model.

```
1 composite-process = atomic-process, control-construct, {binding}+;
2 control-construct = sequence | split | any | choice | if-then-else | repeat-while | repeat-until;
3 sequence = (simple-process)*;
4 selection = condition, simple-process, simple-process;
5 choice = {(condition, simple-process)}*;
6 repeat-while = condition, simple-process;
7 repeat-until = condition, simple-process;
8 split = (simple-process)*;
9 any = (simple-process)*;
10 condition = sentence;
11 binding = sink, source;
12 sink = reference;
13 source = constant | reference | function-term | relation;
14 reference = process-id, parameter-id;
15 process-id = ?id in the meaning of a pointer to a process with this id?;
16 parameter-id = ?id in the meaning of a pointer to a parameter with this id?;
17 constant = ?concrete instance of a class or subclass of the sink’s parameter class type?;
```

Control constructs define the control flow of the subprocesses of a composite process. The data flow between subprocesses is specified with the bindings. A binding is a link between a single data sink and one or multiple data sources of a different or the same
process. A sink is a reference to a single parameter (input, output) of a process (the composite process itself or any one of its subprocesses). A source is specified according to the OWL-S process model as one of four options. Firstly, it may be a constant value of an instance of a class type subsumed by the sink’s class type. Secondly, a source may also be a reference for establishing direct one-to-one links between two single parameters (e.g. an output parameter of a subprocess and an input parameter of a successive subprocess). Thirdly, a source may be based on a function that calculates the value of a data sink based on the values of one or multiple data sources. Fourthly, a source may be defined by the truth value of a relation of one or multiple data sources (their values and/or class types). Both a function and relation can be used in a nested manner that allows the incorporation of all three other options, thus providing a powerful mechanism for describing data dependencies between processes.

Bindings are not only used to define the data flow between the subprocesses of a composite process. They also link, where required, the input and output parameters of the composite service to parameters of its subprocesses. This way, a complete data flow inside a composite process is guaranteed. Bindings in this thesis are defined to occur solely between a composite process and its direct subprocesses but not, in contrast to the OWL-S process model with possible further subprocesses of a subprocess. This restriction does not limit the expressiveness of the process model given the Structured Program Theorem. The advantage of the restriction is that composite processes do not reveal or require internal details beyond their scope. The international structure of subprocesses is completely hidden (revealing good information hiding and strong modularity).

Given the way bindings are defined in this thesis there is no guarantee of their correctness (e.g. the avoidance of circles and self-references). It is assumed that tools exist to assist developers verify the correctness of the control flow and data flow of a composite service. However, this is out of the scope of this thesis.

In several places above complex functional or logical expressions for defining preconditions and effects of processes, conditions of control constructs or sources of data flow bindings were described. The Knowledge Interchange Format (KIF) has been proposed and developed by Genesereth [57] for specifying such logical expressions in a declarative fashion. Listing 6.3 contains the specification of the Standard Upper Ontology of KIF
(SUO-KIF) for expressing declarative semantics, including elements of propositional and first-order logic as well as means for defining relation terms and function terms that can be incorporated in any logical proposition or statement.

Listing 6.3 contains one minor extension of the original specification. The notion of a reference has been added in this thesis as a special variable type in order to emphasise the use of pointers to instances of class types (e.g. a subprocess, an input parameter, etc.).

Listing 6.3: Extended-BNF specification based on the Standard Upper Ontology Knowledge Interchange Format (SUO-KIF) language, as proposed by Pease [135] and Niles and Pease [128], for the declarative description of the semantics of preconditions, effects, conditions and bindings.

```
1 sentence   = word | equation | relation | logical −sentence | qualified −sentence;
2 word       = ?any identifier ?;
3 equation   = '(' operator, term, term ')';
4 operator   = '=' | '̸= | '<' | '≤' | '>' | '≥';
5 relation   = '(' relation −name, {term}+ ')';
6 relation −name = ?unique identifier of a relation ?;
7 logical −sentence = '(' not sentence ')'
8       | '( and {sentence}+ ')'
9       | '( or {sentence}+ ')'
10      | '(' ⇒, sentence, sentence ')'
11      | '(' ⇔, sentence, sentence ')';
12 qualified −sentence = '(' forall '(' {variable}+ ')') sentence ')' |
13      | '(' exists '(' {variable}+ ')') sentence ')';
14 term      = variable | word | string | function −term | number | sentence;
15 variable  = '?' word | '@' word | reference ;
16 string    = ?any string?;
17 function −term = '(' function −name, {term}+ ')';
18      | '(' function −name, {sentence}+ ')';
19 function −name = ?unique identifier of a function ?;
20 number    = ?any natural or real number?;
```

6.3.4 On-demand Service Composition Elements

Based on the previously outlined fundamental data structures, this section defines the data structures that ensure interoperability in inter-agent interactions during on-demand service composition processes. Listing 6.4 shows their formal definition using the EBNF
notation. As stressed before, the notion of a \textit{simple process} is aimed at providing a generalised view of both atomic and composite processes.

\textbf{Listing 6.4:} Extended-BNF specification of the structural definition of key data structures of the on-demand service composition process.

\begin{verbatim}
1 simple-process = atomic-process | composite-process;
2 composite-service-request = composite-process; ?using generic subprocess definitions ?
3 advertisement = simple-process, service-name, service-type, provider-info;
4 service-name = ?uuid of the service ?;
5 service-type = ?unique id with regard to a service type classification such as UNSPSC?;
6 provider-info = ?textual information about the real business entity that provides a service and its contact details ?;
7 composite-service-proposal = composite-process; ?using concrete subprocess definitions based on advertisements ?;
\end{verbatim}

A \textit{composite service request} describes the problem that triggers an on-demand service composition process in open environments. It is defined as a composite process that describes a required or requested composite service using generic subprocess descriptions. That means the definition of subprocesses is not specific enough (e.g. using process-types instead of concrete processes with generic input, output, precondition and effect parameters) to be enacted. Information about concrete services that realise the atomic subprocesses of a composite service request is missing.

An \textit{advertisement} of a concrete service that realises a process definition is defined with a simple process and additional information about \textit{service name}, \textit{service type} and \textit{provider information}. The service name represents a universally unique identifier (UUID). The service type is a non-ambiguous reference to a standardised service classification system such as UNSPSC [162]. The provider information contains information about the business entity that provides the advertised service, including contact information.

A \textit{composite service proposal} is also defined as a composite process. But in contrast to a composite service request, its atomic subprocesses have been replaced or enriched with concrete information from the service advertisements of the service provider agents that together propose a composite service proposal. A composite service proposal is specific enough to be enacted. It can be understood as the refilled version of a previous composite
service request data structure.

Besides these complex data structures, software agents use additional data structures for exchanging information during on-demand service composition processes. The additional data structures are functional and logical expressions that can be expressed with the declarative semantics of the SUO-KIF as specified in Listing 6.3.

The defined data structures are also of use for modelling and implementing the automated planning and various decision-making algorithms within this thesis. The use of the data structures will be described where appropriate in the respective sections of Chapter 7. Note that this thesis does not limit data representation to the described data structures (which would contradict the property of agent autonomy). However, the defined data structures have been applied for the sake of simplicity for the development of a prototype implementation of the concept of on-demand service composition in open environments. The chosen data structures can, of course, be converted and replaced by any other data representation.

6.4 Interaction Protocols

As Henderson-Sellers and Gorton [67, p.3 footnote 2] remark: “The interactions between autonomous components are the essence of multi-agent technology”. These interactions are established following well-defined interaction protocols. Accordingly, a set of interaction protocols forms the core of the concept of on-demand service composition in open environments presented in this thesis.

An interaction protocol defines the correct exchange of information between 2 or more participants for establishing meaningful conversations in the context of a given application or problem domain. It describes temporal and logical constraints on single messages and between different messages, the syntax, semantics and contents of these messages, and the implications of these messages on the participants: sender and receivers. The participants of an interaction protocol are typically modelled with generic roles which they may also change throughout the course of an interaction protocol.

This section presents a semi-formal specification of the interaction protocols of the approach of this thesis based on their chronological order: starting from a new composite service request and ending with contracting a winning composite service proposal. Inter-
action protocols are defined with UML sequence diagrams of a special flavour based on Agent-UML (AUML) as defined by Huget et al. [74]. (Refer to Appendix A for a brief overview of AUML sequence diagram elements and notations.) The sequence diagrams are supplemented with textual descriptions that add further information, such as about the context of the interaction protocol, its participants, the message flow in general and specific features of interest. The messages of the interaction protocols are specified with tables based on a structure derived from the FIPA ACL standards [50, 52]. For every message the following information is presented:

- **Message id**, the unique name of a message
- **ACL performative**, the speech act (e.g. assertive, directive or declarative) of the message that expresses a goal, an action or the intention of an action the sender or receiver is meant to perform
- **Sender**, the agent type and role of the message sender
- **Receiver**, the agent type and role of the message receiver
- **Type**, the message type, such as unicast, multicast, broadcast
- **Payload**, the contents of the message (partially in EBNF)
- **Expected response**, the expected response message, if any
- **Constraints**, temporal and logical constraints imposed on the message or its sender
- **Preconditions**, conditions that are in place before sending the message
- **Rational effects**, conditions that are in place after sending the message.

The interaction protocols of the approach of this thesis are aligned with each other according to the temporal and causal relationships between them. The resulting structure is illustrated in Figure 6.5. It comprises three parts. Firstly, the overarching Composite Service Request interaction protocol models the interactions between a service consumer agent that lodges a composite service request and a set of service provider agents in the multiagent system. Secondly, a set of coalition formation interaction protocols leverage
the interactions among service provider agents during the on-demand coalition formation process outlined in Chapter 5. Thirdly, the Coalition Completion and Composite Service Contracting interaction protocol facilitates the contracting of winning coalitions.

![Diagram of interaction protocols](image)

**Figure 6.5:** Schematic overview of the interaction protocols and their relationships.

The interaction protocols are in general loosely coupled. They are typically not nested such that one interaction protocol must directly trigger another one. The enactment of an interaction protocol may happen independently of previous interactions. It depends on initial conditions that must be met by a message sender before engaging in a conversation based on a particular protocol. The loose coupling of the interaction protocols is caused by the degree of freedom that software agents exhibit based on local and autonomous decision-making. It is also required by the uncertainty and non-determinism of the open market environment (see Section 4.1). There are only a few situations in which interaction protocols are directly embedded such that it is imperative to recursively enact them in order to ensure correctness of processing.

### 6.4.1 Composite Service Request Interaction Protocol

The Composite Service Request interaction protocol is the overarching interaction protocol that models the interactions between a service consumer agent on the one hand and a set of service provider agents on the other hand. It can be characterised best as a basic Contract Net Protocol (Smith [153]) in which the service consumer agent calls for proposals for a
6. MULTIAGENT SYSTEM DESIGN

specific composite service and then negotiates the best (in terms of some utility) composite service proposal with a number of responding service provider agents.

Figure 6.6 illustrates the Composite Service Request interaction protocol. The initial condition of this interaction protocol is that the service consumer agent has a generic plan of a composite service request including client-defined QoS preferences. The protocol commences with the service consumer agent in the requester role sending out a call for proposal in the form of the composite-service-request message. Due to the lack of knowledge (uncertainty of the environment), the service provider agent broadcasts the composite-service-request message to all service provider agents \( n \) in the multiagent system. However, only \( m \) service provider agents \( (0 \leq m \leq n) \) might receive the message due to the partial observability of open environments.

Service provider agents which receive the message reason about it and may decide to engage in further coalition formation interaction protocols as described below. Some of them \( (p < m) \) may adopt the leader role and reply with a proposal in the form of a composite-service-proposal message. This message indicates that a leader was able to determine a complete coalition of service provider agents that together provide a composite service that matches the original composite service request.

If the service consumer agent receives composite-service-proposal messages before the initially set timeout has elapsed, it changes to the contractor role and ranks and selects the received proposal according to the client-defined preferences of the original composite service request. If the service consumer agent has received a satisfactory proposal, it sends one composite-service-accept message to the leader of the winning coalition and composite-service-reject messages to the leaders of all other coalitions, if any.

In order for the leader of the winning coalition to confirm the composite service, the composite-service-accept message must arrive before a previously defined timeout. If so, the leader sends the composite-service-confirm message to the service consumer agent signaling that a contract has been successfully formed between the service consumer agent and the participants of the winning coalition (the leader and all members). Again, to ensure the autonomy of software agents, this message must arrive before a previously defined timeout in order to have effect. After all communication is finalised, all participants of the interaction return into some idle state.
Figure 6.6: AUML sequence diagram of the composite service request interaction protocol.
The interaction protocol also defines alternatives. All participants return to some idle state straight away, in case the service consumer agent did not receive proposals on time. Moreover, composite-service-reject messages may also be sent to all leaders in case the received proposals are not acceptable to the service consumer agent.

In Figure 6.6, the terminal symbol $X$ must be interpreted as the end of interactions and the return of the respective participants to some idle state. This arrangement applies only to this figure for the sake of clarity and brevity of the corresponding diagram.

| Table 6.1: Semi-formal description of the composite-service-request message type. |
|---------------------------------|---------------------------------------------------------------|
| **Message id**                  | composite-service-request                                    |
| **ACL Performative**            | call for proposal                                             |
| **Sender**                      | service consumer agent in requester role                     |
| **Receiver**                    | service provider agent in anySPARole role                    |
| **Type**                        | broadcast message: 1 to $n$, $n \geq 0$                     |
| **Payload**                     | {action, precondition on action}                             |
|                                 | - action: a generic plan being a symbolic description of the activity of providing a composite service (EBNF: composite-service-request;) |
|                                 | - precondition on action: a referential expression specifying QoS preferences in the form of sets of key-weight pairs (EBNF: {qos-parameter}*); |
| **Expected response**           | composite-service-proposal                                   |
| **Constraints**                 | - message timeout: sender waits in blocking mode until a specified deadline |
| **Preconditions**               | - sender has a generic plan and QoS preferences               |
| (for sender)                    | - sender did not send a composite-service-request message for this plan |
|                                 | - sender has no information about available service providers or services and their conditions of service provision |
| **Rational effects**            | - receiver identifies its potential for contribution and prospective profit and, if positive, engages in various coalition formation activities to attempt to produce a composite service proposal |
| (for receiver)                  |                                                               |
| (for sender)                    | - sender passively waits for responses until deadline arrives |
### 6.4. Interaction Protocols

Table 6.2: Semi-formal description of the composite-service-proposal message type.

<table>
<thead>
<tr>
<th>Message id</th>
<th>composite-service-proposal</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL Performative</td>
<td>propose</td>
</tr>
<tr>
<td>Sender</td>
<td>service provider agent in leader role</td>
</tr>
<tr>
<td>Receiver</td>
<td>service consumer agent in requester role</td>
</tr>
<tr>
<td>Type</td>
<td>unicast message: 1 to 1</td>
</tr>
<tr>
<td>Payload</td>
<td>{action, precondition on action}</td>
</tr>
<tr>
<td></td>
<td>- action: a refilled plan being a symbolic description of the activity of providing a specific composite service (EBNF: <code>composite-service-proposal</code>;)</td>
</tr>
<tr>
<td></td>
<td>- precondition on action: a referential expression specifying QoS properties for every task of refilled plan in the form of sets of key-value pairs (EBNF: <code>{parameter}</code>*)</td>
</tr>
<tr>
<td>Expected response</td>
<td>composite-service-accept or composite-service-reject</td>
</tr>
<tr>
<td>Constraints</td>
<td>- message timeout: sender waits in blocking mode until a specified deadline</td>
</tr>
<tr>
<td></td>
<td>- message must arrive before deadline of original composite-service-request</td>
</tr>
<tr>
<td>Preconditions</td>
<td>- sender is in the leader role and its coalition is complete</td>
</tr>
<tr>
<td>(for sender)</td>
<td>- sender believes its coalition comprises a sufficient set of members to provide the requested composite service under the condition the coalition’s QoS properties are accepted</td>
</tr>
<tr>
<td></td>
<td>- sender believes that receiver has no knowledge of the proposition of its coalition and associated QoS properties</td>
</tr>
<tr>
<td>Rational effects</td>
<td>- receiver believes sender’s coalition can provide requested composite service based on the specified QoS properties</td>
</tr>
<tr>
<td>(for receiver)</td>
<td>- receiver validates proposition and compares it with others</td>
</tr>
<tr>
<td></td>
<td>- receiver intends to reply for expressing its acceptance or rejection of proposal after the deadline of previous composite-service-request is due proposal on time:</td>
</tr>
<tr>
<td></td>
<td>- sender intends to engage in activities with coalition members to inform them about timely registration</td>
</tr>
<tr>
<td></td>
<td>- sender passively waits for accept or reject decision of receiver proposal late:</td>
</tr>
<tr>
<td></td>
<td>- sender intends to update coalition members, finalises coalition formation process and returns to the idle role</td>
</tr>
<tr>
<td></td>
<td>- sender plans to update coalition members, finalises coalition formation process and returns to the idle role</td>
</tr>
</tbody>
</table>
### Table 6.3: Semi-formal description of the composite-service-accept message type.

<table>
<thead>
<tr>
<th>Message id</th>
<th>composite-service-accept</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL Performative</td>
<td>accept-proposal</td>
</tr>
<tr>
<td>Sender</td>
<td>service consumer agent in contractor role</td>
</tr>
<tr>
<td>Receiver</td>
<td>service provider agent in leader role</td>
</tr>
<tr>
<td>Type</td>
<td>unicast message: 1 to 1</td>
</tr>
<tr>
<td>Payload</td>
<td>{action, precondition on action}</td>
</tr>
<tr>
<td></td>
<td>- action: proposed plan being a symbolic description of the activity of providing a specific composite service (EBNF: <code>composite-service-proposal:</code>)</td>
</tr>
<tr>
<td></td>
<td>- precondition on action: a referential expression specifying QoS properties for every task of proposed plan in the form of sets of key-value pairs (EBNF: <code>\{parameter\}^*</code>)</td>
</tr>
<tr>
<td>Expected response</td>
<td>composite-service-confirm</td>
</tr>
<tr>
<td>Constraints</td>
<td>- message timeout: sender waits in blocking mode until a specified deadline</td>
</tr>
<tr>
<td></td>
<td>- message must arrive before deadline of previous composite-service-proposal</td>
</tr>
<tr>
<td>Preconditions</td>
<td>- sender has selected proposition of receiver’s coalition as winner</td>
</tr>
<tr>
<td>(for sender)</td>
<td>- sender intends selected composite service to be performed</td>
</tr>
<tr>
<td></td>
<td>- sender believes receiver intends to perform composite service according to given conditions</td>
</tr>
<tr>
<td></td>
<td>- sender did not send this message to another coalition that subsequently confirmed its proposition</td>
</tr>
<tr>
<td>Rational effects</td>
<td>- receiver believes sender intends its coalition to perform the composite service</td>
</tr>
<tr>
<td>(for receiver)</td>
<td>- receiver processes message if coalition internal registration confirmation was successful before</td>
</tr>
<tr>
<td></td>
<td>- receiver intends to reply to sender for confirming the proposition</td>
</tr>
<tr>
<td></td>
<td>- receiver intends to inform coalition members about acceptance of their proposals</td>
</tr>
<tr>
<td></td>
<td>accept on time:</td>
</tr>
<tr>
<td>(for sender)</td>
<td>- sender waits passively for confirmation until deadline is due</td>
</tr>
<tr>
<td></td>
<td>- if no confirmation is received, sender deliberates to select another winner (if possible)</td>
</tr>
<tr>
<td></td>
<td>accept late:</td>
</tr>
<tr>
<td></td>
<td>- sender deliberates to select another winner</td>
</tr>
</tbody>
</table>
### Table 6.4: Semi-formal description of the composite-service-confirm message type.

<table>
<thead>
<tr>
<th>Message id</th>
<th>composite-service-confirm</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL Performative</td>
<td>confirm</td>
</tr>
<tr>
<td>Sender</td>
<td>service provider agent in leader role</td>
</tr>
<tr>
<td>Receiver</td>
<td>service consumer agent in contractor role</td>
</tr>
<tr>
<td>Type</td>
<td>unicast message: 1 to 1</td>
</tr>
<tr>
<td>Payload</td>
<td>{\text{proposition}}</td>
</tr>
<tr>
<td></td>
<td>- proposition: confirming successful agreement on performing the proposed composite service (EBNF: ( (\text{true}) \mid (\text{not true});) )</td>
</tr>
<tr>
<td>Expected response</td>
<td>none</td>
</tr>
<tr>
<td>Constraints</td>
<td>- message must arrive before deadline of previous composite-service-accept</td>
</tr>
<tr>
<td>Preconditions</td>
<td>(for sender)</td>
</tr>
<tr>
<td></td>
<td>- sender has received service-accept message before</td>
</tr>
<tr>
<td></td>
<td>- sender believes that its coalition can provide requested composite service</td>
</tr>
<tr>
<td></td>
<td>- sender believes that receiver is uncertain about final decision</td>
</tr>
<tr>
<td>Rational effects</td>
<td>(for receiver)</td>
</tr>
<tr>
<td></td>
<td>- receiver believes a contract has been fixed because coalition intends to perform composite service as agreed</td>
</tr>
<tr>
<td></td>
<td>- receiver intends to finalise composite service request process and to inform its associated actor confirm on time:</td>
</tr>
<tr>
<td></td>
<td>- sender believes a contract has been fixed</td>
</tr>
<tr>
<td></td>
<td>- sender intends to inform members of its coalition confirm late:</td>
</tr>
<tr>
<td></td>
<td>- sender intends to update coalition members, finalises coalition formation process and returns to the idle role</td>
</tr>
<tr>
<td></td>
<td>(for sender)</td>
</tr>
<tr>
<td></td>
<td>- sender believes a contract has been fixed</td>
</tr>
<tr>
<td></td>
<td>- sender intends to inform members of its coalition confirm late:</td>
</tr>
<tr>
<td></td>
<td>- sender intends to update coalition members, finalises coalition formation process and returns to the idle role</td>
</tr>
<tr>
<td><strong>Table 6.5:</strong> Semi-formal description of the composite-service-reject message type.</td>
<td></td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td><strong>Message id</strong></td>
<td>composite-service-reject</td>
</tr>
<tr>
<td><strong>ACL Performative</strong></td>
<td>reject-proposal</td>
</tr>
<tr>
<td><strong>Sender</strong></td>
<td>service consumer agent in contractor role</td>
</tr>
<tr>
<td><strong>Receiver</strong></td>
<td>service provider agent in leader role</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>unicast message: 1 to 1</td>
</tr>
</tbody>
</table>
| **Payload** | \{(action, precondition on action), proposition\}
- action: a refilled plan being a symbolic description of providing a specific composite service (EBNF: `composite-service-proposal`);
- precondition on action: a referential expression specifying QoS properties for every task of composite service in the form of sets of key-value pairs (EBNF: `\{parameter\}`);
- proposition: empty, implying dissatisfaction of the sender with QoS properties of the associated composite service proposal |
| **Expected response** | none |
| **Constraints** | none |
| **Preconditions** (for sender) | - sender believes it does not intend receiver’s coalition to perform requested composite service
- sender believes receiver does not have knowledge about its belief |
| **Rational effects** (for receiver) | - receiver believes sender does not intend its coalition to perform requested composite service
- receiver intends to update coalition members, finalises coalition formation process and returns to the idle role |

### 6.4.2 Service Advertisement Dissemination Interaction Protocol

The Service Advertisement Dissemination interaction protocol enables service provider agents to disseminate their service proposals in the multiagent system in an ad hoc manner. Figure 6.7 shows the sequence diagram of the interaction protocol. The initial conditions of the protocol are the following. Firstly, the sender has received a composite service request while acting in the idle role. Secondly, the service provider agent performed a successful matchmaking of its own service with the generic service types of the composite service request; that is, it detected that it can contribute to a solution. If both conditions are met, the service provider agent engages in the interaction protocol by sending out
6.4. INTERACTION PROTOCOLS

A service-advertisement message. This message is broadcast into the multiagent system because of the lack of knowledge of the sender about the availability and capabilities of the other service provider agents \( (n) \). However, only \( m \) service provider agents \( (0 \leq m \leq n) \) might receive the message due to the partial observability of open environments.

![Diagram of service advertisement dissemination interaction protocol](image)

**Figure 6.7:** AUML sequence diagram of the service advertisement dissemination interaction protocol.

The receivers of the service-advertisement message may be in any role. If they are aware of the original composite service request, they reason about the message and may decide to perform service matchmaking in order to detect whether the advertisement is beneficial for them. In the positive case, they update their knowledgebases to account for the new information they have received about a potential coalition partner or even competitor.

By sending out a service-advertisement message a service provider agent expresses its interest in participating in a subsequent coalition formation process. Service advertisements form the base for finding prospective coalition partners. The sender changes to the candidate role. It is now available for coalition formation interactions with other service provider agents. Service advertisements are also important for reducing the communication overhead of broadcasting further interactions during an on-demand service composition process. They allow the establishment of unicast connections between just two communication partners.
### Table 6.6: Semi-formal description of the service-advertisement message type.

<table>
<thead>
<tr>
<th>Message id</th>
<th>service-advertisement</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL Performative</td>
<td>inform</td>
</tr>
<tr>
<td>Sender</td>
<td>service provider agent in idle role</td>
</tr>
<tr>
<td>Receiver</td>
<td>service provider agent in anySPARole role</td>
</tr>
<tr>
<td>Type</td>
<td>broadcast message: 1 to $n$, $n \geq 0$</td>
</tr>
<tr>
<td>Payload</td>
<td>{proposition}</td>
</tr>
<tr>
<td></td>
<td>- proposition: a service advertisement of a service including associated QoS properties (EBNF: advertisement;)</td>
</tr>
<tr>
<td>Expected response</td>
<td>none</td>
</tr>
<tr>
<td>Constraints</td>
<td>none</td>
</tr>
<tr>
<td>Preconditions (for sender)</td>
<td>- sender received a composite service request and is in the idle role</td>
</tr>
<tr>
<td></td>
<td>- sender detected matching of its own capabilities with task(s) of the plan of the requested composite service</td>
</tr>
<tr>
<td></td>
<td>- sender adopted composite service request as a belief</td>
</tr>
<tr>
<td></td>
<td>- sender’s goal is set to intend to form successful coalition to provide a composite service matching the requested plan</td>
</tr>
<tr>
<td></td>
<td>- sender creates service advertisement including QoS properties it believes it can provide</td>
</tr>
<tr>
<td></td>
<td>- sender did not disseminate this service advertisement</td>
</tr>
<tr>
<td></td>
<td>- sender believes receiver does not have knowledge about its service advertisement</td>
</tr>
<tr>
<td>Rational effects (for receiver)</td>
<td>- receiver performs matchmaking</td>
</tr>
<tr>
<td></td>
<td>- receiver believes proposition (stores service advertisement in its knowledge-base) if matching is detected</td>
</tr>
<tr>
<td>Rational effects (for sender)</td>
<td>- sender believes potential partners have knowledge about its service advertisement</td>
</tr>
<tr>
<td></td>
<td>- sender changes to the candidate role</td>
</tr>
</tbody>
</table>
6.4.3 Coalition Initiation Interaction Protocol

The Coalition Initiation interaction protocol supports a service provider agent in requesting another service provider agent to join its coalition. It facilitates the formation of initial coalition structures of service provider agents with complementary services that together satisfy a composite service request. Figure 6.8 shows the sequence diagram of the interaction protocol. The initial conditions of the protocol are the following. Firstly, the sender is in the candidate role. Secondly, it has a sufficient number of service advertisements from prospective coalition partners. Sufficient means that the number of advertisements is sufficient and necessary for forming a coalition that can provide the requested composite service. Thirdly, the sender has performed decision-making and determined the most beneficial coalition partner. This decision is constrained by the client-defined QoS preferences of the original composite service request.

If all conditions are met, the sender sends the coalition-request message to a service provider agent that acts in the engaged role. The coalition-request message represents a direct offer to collaborate. It is the only means to intentionally form and also extend coalitions of service provider agents. The coalition-request message contains the service advertisement of the sender. If the receiver of the message did not receive the advertisement yet, it performs service matchmaking and updates its knowledgebase as described in Section 6.4.2. Next, if the receiver detects a sufficient number of service advertisements, it may decide to assess the requesting coalition or service provider agent. The outcome of the deliberation process may determine that the requesting coalition is beneficial. However, a response to the coalition-request message also depends on whether the receiver has committed to a complete coalition. If there is no complete coalition the receiver replies with a coalition-accept message. In the other case, it replies with a coalition-reject message.

The original coalition-request message is sent with a timeout in order to ensure the autonomy of the sender. If the sender receives no reply on time or if the sender receives a coalition-reject message on time, it updates its knowledgebase and then both interaction partners terminate the interaction protocol. If the sender receives a coalition-accept message on time, the following happens.

The sender updates its knowledgebase and changes to the leader role. This is because a coalition-accept message indicates that the other service provider agent is willing to join
Figure 6.8: AUML sequence diagram of the coalition initiation interaction protocol.
in a coalition with the sender and it also accepts the sender as the leader of this coalition. The receiver also updates its knowledgebase and changes to the member role. Since the receiver may already be a member or even leader of another coalition, it must leave that coalition in order to join the new coalition. In these cases, the receiver performs either the Leader Leave or the Member Leave interaction protocol. The execution of these interaction protocols is mandatory at this point. Finally, both service provider agents terminate the interaction protocol.

After the Coalition Initiation interaction protocol has been performed, a base coalition may have been formed comprising one leader and one member. An extension of the coalition may be required, depending on the complexity of the requested composite service. In that case, the service provider agents of the coalition engage in the following Coalition Extension interaction protocol.
### Table 6.7: Semi-formal description of the coalition-request message type.

<table>
<thead>
<tr>
<th>Message id</th>
<th>coalition-request</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL Performative</td>
<td>request</td>
</tr>
<tr>
<td>Sender</td>
<td>service provider agent in candidate role or leader role</td>
</tr>
<tr>
<td>Receiver</td>
<td>service provider agent in engaged role</td>
</tr>
<tr>
<td>Type</td>
<td>unicast message: 1 to 1</td>
</tr>
<tr>
<td>Payload</td>
<td>{action, proposition}</td>
</tr>
<tr>
<td></td>
<td>- action: expresses the activity “to join” which means to commit to the coalition as a member (EBNF: (join-coalition ?);)</td>
</tr>
<tr>
<td></td>
<td>- proposition: a set of service advertisements of sender and all existing members (EBNF: {advertisement}+)</td>
</tr>
<tr>
<td>Expected response</td>
<td>coalition-accept or coalition-reject</td>
</tr>
<tr>
<td>Constraints</td>
<td>- message timeout: sender waits in blocking mode until a specified deadline</td>
</tr>
<tr>
<td>Preconditions</td>
<td>- sender has sufficient number of service advertisement available</td>
</tr>
<tr>
<td>(for sender)</td>
<td>- sender performed decision-making to determine the most beneficial coalition partner</td>
</tr>
<tr>
<td></td>
<td>- sender believes receiver is most beneficial coalition partner</td>
</tr>
<tr>
<td></td>
<td>- sender does not believe receiver intends to form a coalition with it</td>
</tr>
<tr>
<td></td>
<td>- sender is not engaged in any blocking mode-restricted conversations</td>
</tr>
<tr>
<td>Rational effects</td>
<td>- receiver performs matchmaking and updates its knowledgebase</td>
</tr>
<tr>
<td>(for receiver)</td>
<td>- receiver deliberates about joining the requesting coalition, if it has a sufficient number of service advertisement available</td>
</tr>
<tr>
<td>(for sender)</td>
<td>- sender waits passively for response until deadline arrives</td>
</tr>
</tbody>
</table>
### Table 6.8: Semi-formal description of the coalition-accept message type.

<table>
<thead>
<tr>
<th>Message id</th>
<th>coalition-accept</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL Performative</td>
<td>agree</td>
</tr>
<tr>
<td>Sender</td>
<td>service provider agent in engaged role</td>
</tr>
<tr>
<td>Receiver</td>
<td>service provider agent in candidate role or leader role</td>
</tr>
<tr>
<td>Type</td>
<td>unicast message: 1 to 1</td>
</tr>
<tr>
<td>Payload</td>
<td>{action, proposition}</td>
</tr>
<tr>
<td></td>
<td>- action: expresses the activity “to join” which means to commit to the coalition as a member (EBNF: (join–coalition true);)</td>
</tr>
<tr>
<td></td>
<td>- proposition: is empty, meaning the acceptance of receiver as coalition leader is unconditional at the time of joining</td>
</tr>
<tr>
<td>Expected response</td>
<td>none</td>
</tr>
<tr>
<td>Constraints</td>
<td>- message must arrive before deadline of previous coalition-request</td>
</tr>
<tr>
<td>Preconditions (for sender)</td>
<td>- sender performed decision-making to assess requesting coalition</td>
</tr>
<tr>
<td></td>
<td>- sender believes it intends to join because coalition is more beneficial than existing arrangements</td>
</tr>
<tr>
<td></td>
<td>- sender does not believe receiver has knowledge about its intentions</td>
</tr>
<tr>
<td></td>
<td>- sender is not engaged in any blocking mode-restricted conversations</td>
</tr>
<tr>
<td>Rational effects (for receiver)</td>
<td>- receiver believes sender intends to join and will do so</td>
</tr>
<tr>
<td></td>
<td>- receiver updates its knowledgebase</td>
</tr>
<tr>
<td></td>
<td>- receiver intends to change to the leader role (if necessary)</td>
</tr>
<tr>
<td></td>
<td>- receiver intends to confirm new member with existing members (if any) accept on time:</td>
</tr>
<tr>
<td></td>
<td>- sender updates knowledgebase and joins new coalition</td>
</tr>
<tr>
<td></td>
<td>- sender intends to change to the member role (if necessary)</td>
</tr>
<tr>
<td></td>
<td>- sender intends to leave current coalition (if necessary)</td>
</tr>
</tbody>
</table>
### Table 6.9: Semi-formal description of the coalition-reject message type.

<table>
<thead>
<tr>
<th>Message id</th>
<th>coalition-reject</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL Performative</td>
<td>refuse</td>
</tr>
<tr>
<td>Sender</td>
<td>service provider agent in engaged role</td>
</tr>
<tr>
<td>Receiver</td>
<td>service provider agent in candidate role or leader role</td>
</tr>
<tr>
<td>Type</td>
<td>unicast message: 1 to 1</td>
</tr>
<tr>
<td>Payload</td>
<td>{action, proposition}</td>
</tr>
<tr>
<td></td>
<td>- action: expresses the activity “not to join” which means not to commit to the coalition as a member (EBNF: (join–coalition false);)</td>
</tr>
<tr>
<td></td>
<td>- proposition: is empty, meaning no reasons for refusal are provided</td>
</tr>
<tr>
<td>Expected response</td>
<td>none</td>
</tr>
<tr>
<td>Constraints</td>
<td>none</td>
</tr>
<tr>
<td>Preconditions</td>
<td>- sender performed decision-making to assess requesting coalition</td>
</tr>
<tr>
<td>(for sender)</td>
<td>- sender believes it does not intend to join because requesting coalition is not more beneficial than existing arrangements</td>
</tr>
<tr>
<td></td>
<td>- sender does not believe receiver has knowledge about its refusal</td>
</tr>
<tr>
<td></td>
<td>- sender is not engaged in any blocking mode-restricted conversations</td>
</tr>
<tr>
<td>Rational effects</td>
<td>- receiver believes sender does not intend to join its coalition</td>
</tr>
<tr>
<td>(for receiver)</td>
<td>- receiver updates its knowledgebase</td>
</tr>
<tr>
<td></td>
<td>- receiver intends to inform existing members about refusal (if any)</td>
</tr>
<tr>
<td></td>
<td>- receiver intends to determine next prospective partner</td>
</tr>
</tbody>
</table>
6.4.4 Coalition Extension Interaction Protocol

The Coalition Extension interaction protocol supports a service provider agent in requesting another service provider agent to join its coalition. It is an extension of the Coalition Initiation interaction protocol, adding interactions for communicating activities for extending a coalition among a set of existing coalition members. Figure 6.9 shows the sequence diagram of the interaction protocol. The difference between the both interaction protocols lies in differing initial conditions. Firstly, the sender is in the leader role. Secondly, there exists at least one coalition member, which means that an initial coalition structure already exists. Thirdly, it has a sufficient number of service advertisements from prospective coalition partners. Sufficient here means that the number of advertisements is sufficient and necessary for forming a coalition that can provide the requested composite service. Fourthly, the sender has performed decision-making and determined the most beneficial coalition partner. This decision is constrained by the client-defined QoS preferences of the original composite service request.

The Coalition Extension interaction protocol executes like the Coalition Initiation interaction protocol, but with four major differences. Firstly, the coalition-request message does not contain only one advertisement, but the advertisements of all existing participants of the coalition. Accordingly, the receiver of the coalition-request message may process various advertisements before it deliberates about the prospects of joining the requesting coalition. Secondly, in case the sender receives a coalition-accept message before the timeout is due, the following happens. The sender updates its knowledgebase. It then multicasts a coalition-inform message to all existing members of its coalition in order to communicate the arrival of a new member. Following this message, the sender and all existing members update their knowledgebases in order to reflect the extension of the coalition. Thirdly, a coalition-inform message is sent by the sender if it receives a coalition-reject message on time. This is required in order to allow coalition members to adjust their local decision-making. Fourthly, after a coalition has been successfully extended, the coalition leader determines if the coalition comprises a sufficient and necessary set of service provider agents that together can provide the requested composite service. If the coalition is complete, the leader triggers a Coalition Completion and Composite Service Contracting interaction protocol (see Section 6.4.7).
Figure 6.9: AUML sequence diagram of the coalition extension interaction protocol.
Table 6.10: Semi-formal description of the coalition-inform message type.

<table>
<thead>
<tr>
<th>Message id</th>
<th>coalition-inform</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL Performative</td>
<td>inform</td>
</tr>
<tr>
<td>Sender</td>
<td>service provider agent in leader role</td>
</tr>
<tr>
<td>Receiver</td>
<td>service provider agent in member role</td>
</tr>
<tr>
<td>Type</td>
<td>multicast message: 1 to ( n ), ( n ) = number of previous coalition members</td>
</tr>
<tr>
<td>Payload</td>
<td>( { \text{proposition} } )</td>
</tr>
<tr>
<td></td>
<td>- proposition: the service advertisement of a new coalition member</td>
</tr>
<tr>
<td></td>
<td>(EBNF: advertisement; )</td>
</tr>
<tr>
<td>Expected response</td>
<td>none</td>
</tr>
<tr>
<td>Constraints</td>
<td>none</td>
</tr>
<tr>
<td>Preconditions (for sender)</td>
<td>- sender believes a new member has joined</td>
</tr>
<tr>
<td></td>
<td>- sender believes existing members have no knowledge about new member</td>
</tr>
<tr>
<td>Rational effects (for receiver)</td>
<td>- receiver believes a new member has joined coalition</td>
</tr>
<tr>
<td></td>
<td>- receiver updates its knowledgebase</td>
</tr>
<tr>
<td></td>
<td>- receiver deliberates to assess prospects of updated coalition</td>
</tr>
<tr>
<td>Rational effects (for sender)</td>
<td>- sender intends to validate completeness of its coalition in order to trigger appropriate actions</td>
</tr>
</tbody>
</table>

After the Coalition Extension interaction protocol has been performed, an existing coalition may have been extended with a new member. It is up to the existing members to decide whether they accept the new member. The local preferences of service provider agents and their local knowledge may drive them to leave the coalition. In addition, every service provider agent may receive a coalition-request message anytime. Thus it is also possible that a leader decides to leave. These cases are supported with the interaction protocols in the two following sections (6.4.5 and 6.4.6).
6.4.5 Leader Leave Interaction Protocol

The Leader Leave interaction protocol allows a coalition leader to communicate its intention to leave its coalition. Figure 6.10 shows the sequence diagram of the interaction protocol. The only initial condition of the protocol is that the coalition leader must not be committed to a complete coalition. Apart from this constraint, a service provider agent in the leader role can leave its coalition any time. To do so, it multicasts a leader-leave message to all existing coalition members. After the leader has sent the message, it either changes to the candidate role, or to the member role if it has agreed to join another coalition. The coalition members update their knowledgebases after receiving the leader-leave message. They then change to the candidate role. The Leader Leave interaction protocol results in the disbanding of the entire coalition. This fact appears to be very harsh. However, the election of a new leader is a complex and time-consuming process, given the local autonomous decision-making of software agents. Existing members may decide to leave the coalition as a reaction to a leader-leave message. Some also may have mutually contradicting local preferences that simply prevent an agreement. In the context of the real-time constraints and the uncertainty of the open market environment it is too risky with respect to the overall goal of forming coalitions before the registration deadline to conduct an election.

Figure 6.10: AUML sequence diagram of the leader leave interaction protocol.
### 6.4. INTERACTION PROTOCOLS

<table>
<thead>
<tr>
<th>Message id</th>
<th>leader-leave</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL Performative</td>
<td>disconfirm</td>
</tr>
<tr>
<td>Sender</td>
<td>service provider agent in leader role</td>
</tr>
<tr>
<td>Receiver</td>
<td>service provider agent in member role</td>
</tr>
<tr>
<td>Type</td>
<td>multicast message: 1 to ( n ), ( n = ) the number of coalition members</td>
</tr>
<tr>
<td>Payload</td>
<td>{proposition}</td>
</tr>
<tr>
<td></td>
<td>- proposition: indicates sender disbands its coalition</td>
</tr>
<tr>
<td></td>
<td>(EBNF: ((\text{is-leader} ; @\text{sender} ; false));)</td>
</tr>
<tr>
<td>Expected response</td>
<td>none</td>
</tr>
<tr>
<td>Constraints</td>
<td>- coalition is not complete</td>
</tr>
<tr>
<td>Preconditions (for sender)</td>
<td>- sender performed decision-making and decided to disband its coalition</td>
</tr>
<tr>
<td></td>
<td>- sender intends to inform receiver about its belief</td>
</tr>
<tr>
<td></td>
<td>- sender believes receiver does not have knowledge about its belief</td>
</tr>
<tr>
<td>Rational effects (for receiver)</td>
<td>- receiver believes sender disbands its coalition</td>
</tr>
<tr>
<td></td>
<td>- receiver updates its knowledgebase</td>
</tr>
<tr>
<td></td>
<td>- receiver intends to change to candidate role</td>
</tr>
<tr>
<td></td>
<td>- receiver intends to deliberate to determine next prospective partner</td>
</tr>
<tr>
<td>(for sender)</td>
<td>- sender intends to change to the member role (if joining another coalition)</td>
</tr>
</tbody>
</table>
6.4.6 Member Leave Interaction Protocol

The Member Leave interaction protocol allows a coalition member to communicate its intention to leave the coalition. Figure 6.11 shows the sequence diagram of the interaction protocol. The only initial condition of the protocol is that the coalition member must not be committed to a complete coalition. Apart from this constraint, a service provider agent in the member role may leave the coalition any time. To do so, it multicasts a member-leave message to all existing participants of its coalition. After the member has sent the message, it either changes to the candidate role, or remains in the member role if it has agreed to join another coalition. The leader and the other members of the coalition update their knowledgebases after receiving the member-leave message. The leader then has to decide whether it needs to change its role. If no coalition members are left in the coalition, it abandons its leader role and changes to the candidate role. The Member Leave interaction protocol results in the decrease of the number of coalition members. It will result in the disbanding of the entire coalition if no members are left.

Figure 6.11: AUML sequence diagram of the member leave interaction protocol.
### Table 6.12: Semi-formal description of the member-leave message type.

<table>
<thead>
<tr>
<th>Message id</th>
<th>member-leave</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL Performative</td>
<td>disconfirm</td>
</tr>
<tr>
<td>Sender</td>
<td>service provider agent in member role</td>
</tr>
<tr>
<td>Receiver</td>
<td>service provider agent in leader role or member role</td>
</tr>
<tr>
<td>Type</td>
<td>multicast message: 1 to $n$, $n = \text{number of remaining members} + 1$ leader</td>
</tr>
<tr>
<td>Payload</td>
<td>{proposition}</td>
</tr>
<tr>
<td></td>
<td>- proposition: indicates sender leaves coalition</td>
</tr>
<tr>
<td></td>
<td>(\text{EBNF: (is-member @sender false);})</td>
</tr>
<tr>
<td>Expected response</td>
<td>none</td>
</tr>
<tr>
<td>Constraints</td>
<td>- coalition is not complete</td>
</tr>
<tr>
<td>Preconditions (for sender)</td>
<td>- sender performed decision-making and decided to leave coalition</td>
</tr>
<tr>
<td></td>
<td>- sender intends to inform receiver about its belief</td>
</tr>
<tr>
<td></td>
<td>- sender believes receiver does not have knowledge about its belief</td>
</tr>
<tr>
<td>Rational effects (of the receiver)</td>
<td>- receiver believes sender leaves coalition</td>
</tr>
<tr>
<td></td>
<td>- receiver updates its knowledgebase</td>
</tr>
<tr>
<td></td>
<td>- receiver intends to change to the candidate role (if in the leader role and no members are left)</td>
</tr>
<tr>
<td></td>
<td>- receiver intends to deliberate to determine next prospective partner (if in the leader role)</td>
</tr>
<tr>
<td></td>
<td>- receiver intends to deliberate to determine prospects of coalition (if in the member role)</td>
</tr>
</tbody>
</table>
6. MULTIAGENT SYSTEM DESIGN

6.4.7 Coalition Completion and Contracting Interaction Protocol

The Coalition Completion and Composite Service Contracting interaction protocol serves two related purposes. Firstly, it facilitates the participants of a coalition to determine whether their coalition is complete and to signal their willingness to commit to it. Secondly, it is nested into the overarching Composite Service Request interaction protocol in order to leverage the finalisation of a contract between service consumer agent and the participants of the coalition. Figure 6.12 shows the sequence diagram of the interaction protocol. The initial condition of the protocol is that the coalition leader detected that the coalition comprises a sufficient and necessary set of service provider agents that together can provide the requested composite service.

If the condition is fulfilled, the leader first determines the commitment of the coalition members to the coalition by multicasting a coalition-complete-query message. Now the coalition members have time to assess their commitment to the coalition. They have until the associated message timeout has passed. The coalition members respond with a coalition-complete-inform message that carries a boolean value. If all members respond positively and on time, the leader engages in the second part of the interaction protocol. In all other cases, the coalition leader falls back into other interaction protocols in order to extend its coalition or to abandon it.

The members that reply with a positive coalition-complete-inform message specify timeouts for their messages. The leader must use the available time during the second phase of the protocol to send a composite-service-proposal message to the requesting service consumer agent. If this message arrives on time, the leader must inform all members about the successful registration of the coalition’s proposal by sending a coalition-registration-confirm message. However, the leader may fail to deliver the message to all members on time. In this case, it invalidates the previously sent message with a coalition-registration-disconfirm message. The leader also multicasts a coalition-registration-disconfirm message, if the proposal registration with the service consumer agent was unsuccessful.

If all members have received a coalition-registration-confirm message on time, the interaction protocol proceeds. In this case, the coalition leader waits for a composite-service-accept or composite-service-reject message from the service consumer agent until
6.4. INTERACTION PROTOCOLS

Figure 6.12: AUML sequence diagram of the coalition completion and composite service contracting interaction protocol.
the timeout of its composite-service-proposal message is due. If the leader receives a composite-service-accept message on time, the leader attempts to confirm the proposal of the coalition. If the leader then manages to send back a composite-service-confirm message to the service consumer agent on time, a contract between the coalition and the service consumer agent is formed. The leader agent then multicasts a coalition-confirm message to all members in order to finalise the on-demand service composition process. The leader sends a coalition-disconfirm in all other cases (composite-service-confirm message was sent late, a composite-service-reject message was received on time or no response message from the service consumer agent was received on time).

This interaction protocol marks the last interaction protocol in the chronological order of the activities of the on-demand service composition concept. If it does not lead to a contracted composite service (proposal), the leader of a coalition may perform decision-making activities and initiate other interaction protocols in order to achieve a valid composite service proposal.

6.5 Summary

This chapter outlines the static and dynamic design of a multiagent system that implements the concept of on-demand service composition in open environments presented in this thesis. The chapter describes the components and structure of a multiagent system that implements the open market environment of Section 4.1 according to the constraints and assumptions listed in Section 4.4. Moreover, the chapter defines a minimum set of roles for the two agent types and fundamental data structures. Both are required for ensuring interoperability among autonomous software agents. They form the foundation for the subsequently specified set of interaction protocols. This set of interaction protocols facilitates the implementation of the on-demand service composition process as described in Chapter 5. The static and dynamic design of the multiagent system can be understood as a minimalist approach that incorporates only the minimal set of necessary and sufficient structural elements for enabling the implementation of the concept of on-demand service composition in open environments. The next chapter provides a specification of agent-internal designs of the two agent types.
### Table 6.13: Semi-formal description of the coalition-complete-query message type.

<table>
<thead>
<tr>
<th>Message id</th>
<th>coalition-complete-query</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL Performative</td>
<td>query-if</td>
</tr>
<tr>
<td>Sender</td>
<td>service provider agent in leader role</td>
</tr>
<tr>
<td>Receiver</td>
<td>service provider agent in member role</td>
</tr>
<tr>
<td>Type</td>
<td>multicast message: 1 to ( n ), ( n = ) number of members</td>
</tr>
<tr>
<td>Payload</td>
<td>{proposition}</td>
</tr>
<tr>
<td></td>
<td>- proposition: being the statement 'coalition is complete'</td>
</tr>
<tr>
<td></td>
<td>(EBNF: ( is-)complete coalition ?))</td>
</tr>
<tr>
<td>Expected response</td>
<td>coalition-complete-inform</td>
</tr>
<tr>
<td>Constraints</td>
<td>- message timeout: sender waits in blocking mode until a specified deadline</td>
</tr>
<tr>
<td>Preconditions</td>
<td>- sender deliberated and decided its coalition comprises a sufficient number</td>
</tr>
<tr>
<td>(for sender)</td>
<td>of members for providing requested composite service</td>
</tr>
<tr>
<td></td>
<td>- sender has no knowledge if coalition is complete (members may have left or</td>
</tr>
<tr>
<td></td>
<td>have failed due to errors)</td>
</tr>
<tr>
<td></td>
<td>- sender believes receiver can confirm if proposition is true/false</td>
</tr>
<tr>
<td>Rational effects</td>
<td>- receiver deliberates and makes decision if coalition is complete</td>
</tr>
<tr>
<td>(for receiver)</td>
<td>- receiver intends to respond with truth value {true, false} to (dis)confirm</td>
</tr>
<tr>
<td>(for sender)</td>
<td>proposition</td>
</tr>
<tr>
<td></td>
<td>- sender waits passively for response until deadline is due</td>
</tr>
</tbody>
</table>
### Table 6.14: Semi-formal description of the coalition-complete-inform message type.

<table>
<thead>
<tr>
<th>Message id</th>
<th>coalition-complete-inform</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL Performative</td>
<td>inform</td>
</tr>
<tr>
<td>Sender</td>
<td>service provider agent in member role</td>
</tr>
<tr>
<td>Receiver</td>
<td>service provider agent in leader role</td>
</tr>
<tr>
<td>Type</td>
<td>unicast message: 1 to 1</td>
</tr>
<tr>
<td>Payload</td>
<td>{proposition}</td>
</tr>
<tr>
<td></td>
<td>- proposition: stating either 'coalition is complete' or 'coalition is not complete'</td>
</tr>
<tr>
<td></td>
<td>(EBNF: (is-complete coalition true)</td>
</tr>
<tr>
<td>Expected response</td>
<td>coalition-registration-confirm or coalition-registration-disconfirm</td>
</tr>
<tr>
<td>Constraints</td>
<td>- message timeout: sender waits in blocking mode until a specified deadline</td>
</tr>
<tr>
<td></td>
<td>- message must arrive before deadline of previous coalition-complete-query</td>
</tr>
<tr>
<td>Preconditions (for sender)</td>
<td>- sender deliberated and decided coalition is either complete or not complete</td>
</tr>
<tr>
<td></td>
<td>- sender believes proposition</td>
</tr>
<tr>
<td></td>
<td>- sender believes receiver has no knowledge about its belief</td>
</tr>
<tr>
<td>Rational effects (for receiver)</td>
<td>- receiver believes proposition</td>
</tr>
<tr>
<td></td>
<td>- receiver updates knowledgebase</td>
</tr>
<tr>
<td></td>
<td>- receiver checks if deadline of previous coalition-complete query is due and if it has received all anticipated coalition-complete-inform and if coalition is complete has been confirmed by all senders</td>
</tr>
<tr>
<td></td>
<td>- receiver engages in interactions with service requester to register its coalition if yes:</td>
</tr>
<tr>
<td></td>
<td>- receiver deliberates to find next prospective partner</td>
</tr>
<tr>
<td></td>
<td>- receiver waits passively for (dis)confirm until deadline is due (if proposition is true and coalition-complete arrived on time)</td>
</tr>
<tr>
<td>(for sender)</td>
<td></td>
</tr>
</tbody>
</table>
Table 6.15: Semi-formal description of the coalition-registration-confirm message type.

<table>
<thead>
<tr>
<th>Message id</th>
<th>coalition-registration-confirm</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL Performative</td>
<td>confirm</td>
</tr>
<tr>
<td>Sender</td>
<td>service provider agent in leader role</td>
</tr>
<tr>
<td>Receiver</td>
<td>service provider agent in member role</td>
</tr>
<tr>
<td>Type</td>
<td>multicast message: 1 to n, n = number of coalition members</td>
</tr>
<tr>
<td>Payload</td>
<td>{proposition}</td>
</tr>
<tr>
<td></td>
<td>- proposition: stating coalition is successfully registered with service requestor</td>
</tr>
<tr>
<td></td>
<td>(EBNF: (is–registered coalition true);)</td>
</tr>
<tr>
<td>Expected response</td>
<td>none</td>
</tr>
<tr>
<td>Constraints</td>
<td>- message must arrive before deadline of previous coalition-complete-inform</td>
</tr>
<tr>
<td>Preconditions</td>
<td>- sender registered coalition with requester on time</td>
</tr>
<tr>
<td></td>
<td>- sender intends receiver also to believe the proposition</td>
</tr>
<tr>
<td></td>
<td>- sender believes receiver is uncertain about proposition</td>
</tr>
<tr>
<td>Rational effects</td>
<td>- receiver beliefs proposition</td>
</tr>
<tr>
<td>(for receiver)</td>
<td>- receiver does not engage in activities that can lead to leaving its coalition</td>
</tr>
<tr>
<td></td>
<td>(if confirm arrived on time)</td>
</tr>
<tr>
<td></td>
<td>- receiver finalises coalition formation process and returns to the idle role (if confirm arrived late)</td>
</tr>
<tr>
<td></td>
<td>all confirm on time:</td>
</tr>
<tr>
<td></td>
<td>- sender waits passively for accept/reject from service requester</td>
</tr>
<tr>
<td></td>
<td>one confirm late:</td>
</tr>
<tr>
<td></td>
<td>- sender intends to inform members, finalises coalition formation process and returns to the idle role</td>
</tr>
<tr>
<td>(for sender)</td>
<td></td>
</tr>
</tbody>
</table>
Table 6.16: *Semi-formal description of the coalition-registration-disconfirm message type.*

<table>
<thead>
<tr>
<th>Message id</th>
<th>coalition-registration-disconfirm</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL Performative</td>
<td>disconfirm</td>
</tr>
<tr>
<td>Sender</td>
<td>service provider agent in leader role</td>
</tr>
<tr>
<td>Receiver</td>
<td>service provider agent in member role</td>
</tr>
<tr>
<td>Type</td>
<td>multicast message: 1 to ( n ), ( n = \text{number of coalition members} )</td>
</tr>
<tr>
<td>Payload</td>
<td>{proposition}</td>
</tr>
<tr>
<td></td>
<td>- proposition: stating coalition is not successfully registered with service requester (EBNF: ( \text{is\text{-}registered coalition false});)</td>
</tr>
<tr>
<td>Expected response</td>
<td>none</td>
</tr>
<tr>
<td>Constraints</td>
<td>none</td>
</tr>
<tr>
<td>Preconditions (for sender)</td>
<td>- sender did not register coalition with requester on time</td>
</tr>
<tr>
<td></td>
<td>- sender intends receiver also to believe proposition is false</td>
</tr>
<tr>
<td></td>
<td>- sender believes that receiver is uncertain about or believes proposition</td>
</tr>
<tr>
<td>Rational effects (for receiver)</td>
<td>- receiver believes that proposition is false</td>
</tr>
<tr>
<td></td>
<td>- receiver finalises coalition formation process and returns to the idle role</td>
</tr>
<tr>
<td>Rational effects (for sender)</td>
<td>- sender finalises coalition formation process and returns to the idle role</td>
</tr>
</tbody>
</table>
Table 6.17: *Semi-formal description of the coalition-confirm message type.*

<table>
<thead>
<tr>
<th>Message id</th>
<th>coalition-confirm</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL Performative</td>
<td>confirm</td>
</tr>
<tr>
<td>Sender</td>
<td>service provider agent in leader role</td>
</tr>
<tr>
<td>Receiver</td>
<td>service provider agent in member role</td>
</tr>
<tr>
<td>Type</td>
<td>multicast message: 1 to ( n ), ( n = ) number of coalition members</td>
</tr>
<tr>
<td>Payload</td>
<td>{proposition}</td>
</tr>
<tr>
<td></td>
<td>- proposition: stating composite service proposal of coalition was accepted by service requester (EBNF: ( \text{is-contracted coalition true} ));</td>
</tr>
<tr>
<td>Expected response</td>
<td>none</td>
</tr>
<tr>
<td>Constraints</td>
<td>none</td>
</tr>
<tr>
<td>Preconditions (for sender)</td>
<td>- sender has sent all coalition-registration-confirms on time, has received a composite-service-accept on time and has sent composite-service-confirm on time</td>
</tr>
<tr>
<td></td>
<td>- sender believes proposition</td>
</tr>
<tr>
<td></td>
<td>- sender intends receiver also to believe proposition</td>
</tr>
<tr>
<td></td>
<td>- sender believes receiver is uncertain about proposition</td>
</tr>
<tr>
<td>Rational effects (for receiver)</td>
<td>- receiver believes proposition (a contract was fixed)</td>
</tr>
<tr>
<td>(for sender)</td>
<td>- receiver finalises coalition formation process and returns to the idle role</td>
</tr>
<tr>
<td></td>
<td>- sender believes a contract was fixed</td>
</tr>
<tr>
<td></td>
<td>- sender finalises coalition formation process and returns to the idle role</td>
</tr>
</tbody>
</table>
Table 6.18: Semi-formal description of the coalition-disconfirm message type.

<table>
<thead>
<tr>
<th>Message id</th>
<th>coalition-disconfirm</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL Performative</td>
<td>disconfirm</td>
</tr>
<tr>
<td>Sender</td>
<td>service provider agent in leader role</td>
</tr>
<tr>
<td>Receiver</td>
<td>service provider agent in member role</td>
</tr>
<tr>
<td>Type</td>
<td>multicast message: 1 to ( n ), ( n = ) number of coalition members</td>
</tr>
<tr>
<td>Payload</td>
<td>{proposition}</td>
</tr>
<tr>
<td></td>
<td>- proposition: stating composite service proposal of coalition was not accepted by service requester (EBNF: (\textit{is-contracted coalition} \textit{false});))</td>
</tr>
<tr>
<td>Expected response</td>
<td>none</td>
</tr>
<tr>
<td>Constraints</td>
<td>none</td>
</tr>
<tr>
<td>Preconditions</td>
<td>- sender has received a composite-service-reject or deadline of previous composite-service-proposal is due or composite-service-accept was received on time but composite-service-confirm arrived late</td>
</tr>
<tr>
<td>(for sender)</td>
<td>- sender intends receiver also to believe proposition is false</td>
</tr>
<tr>
<td></td>
<td>- sender believes receiver is uncertain about or believes proposition</td>
</tr>
<tr>
<td>Rational effects</td>
<td>- receiver believes proposition is false (no contract was fixed)</td>
</tr>
<tr>
<td>(for receiver)</td>
<td>- receiver finalises coalition formation process and returns to the idle role</td>
</tr>
<tr>
<td>(for sender)</td>
<td>- sender believes no contract was fixed</td>
</tr>
<tr>
<td></td>
<td>- sender finalises coalition formation process and returns to the idle role</td>
</tr>
</tbody>
</table>
Chapter 7

Software Agent Design

This chapter presents the static and dynamic design of the two agent types that implement the approach to on-demand service composition in open environments presented in this thesis. The chapter commences with the presentation of the internal structure of a service consumer agent and a service provider agent. It then outlines the dynamic design of the two agent types. It describes two finite state machines as the core control structure in the agent types and presents pseudocode listings of the implementation of the various states of each finite state machine. The final part of this chapter describes the decision-making capabilities that are implemented with both agent types. These agent types exemplify implementations of the static and dynamic design of the multiagent system outlined in Chapter 6. Hence this thesis promotes heterogeneous software agents that form meaningful structures dynamically by communicating with each other. The two agent types have been implemented in order to populate a multiagent system for conducting an experimental evaluation of the on-demand service composition concept (Chapter 9).

7.1 Agent-internal Structure

This section presents the agent-internal static design of two agent types on an abstract level. The presentation is given on an abstract level for two reasons. Firstly, a fully detailed definition of all components and relationships inside a software agent would substantially stretch the size of this thesis beyond its scope. Secondly, a fully detailed definition of the agent-internal design is not possible without being specific about a particular
agent architecture and agent toolkit. This thesis, however, focuses on the definition and evaluation of the approach of on-demand service composition in open environments in general. Hence the implementation details of any particular agent architecture and agent toolkit are not considered. The number of available agent toolkits is large and they differ significantly in the agent architectures they implement and how they are programmed. Thus the design concepts in this section are presented in a fashion that allows for their interpretation and implementation with any agent toolkit.

There are two agent types specified in the architecture of the multiagent system that implements the concept of on-demand service composition in open environments (Section 6.1). These agent types are the service consumer agent type and the service provider agent type. Their internal structures are presented together here, because they exhibit a number of similarities. In particular, both agent types incorporate a number of software components that are identical or similar.

The internal structure of the service consumer agent type is depicted in Figure 7.1. The internal structure of the service provider agent type is shown in Figure 7.2. Each of the UML component diagrams illustrates a software system or component that fits into the general definition of a software agent component shown in Figure 2.4 of Section 2.5.1 by replacing the actor class AgentImpl.

**Figure 7.1:** UML component diagram of the main elements of the service consumer agent type.

Both agent types comprise a specific subcomponent that provides capabilities to control agent internal processing and to interact with a software agent’s hosting agent platform and other software agents. This component is called AgentCore. AgentCore represents the core component of both agent types. Hence software agents are implemented to
execute in their own thread of control. In this respect, AgentCore provides a central point of control inside a respective software agent. It controls a software agent’s main loop and manages agent internal processing according to the agent’s life-cycle model and incoming and outgoing events (e.g. decoding and encoding of messages and forwarding of incoming events to registered subcomponents).

![UML component diagram of the internal elements of the service provider agent type.](image)

AgentCore also provides interfaces to the underlying agent platform for managing an agent’s mailbox component and a timer object. The required interface `IAction` provides access to the agent platform for setting up a timer and for sending out messages. The provided interface `IEvent` enables the software agent’s mailbox to notify the agent when a new message has arrived. If a software agent sets up a timer, the timer uses the `IEvent` interface to notify the software agent of an elapsed timeout. Incoming events always result in a restart of a software agent’s main loop, if the agent is not active already. Both interfaces are outlined in a generic and abstract fashion to illustrate their key aspects. A more concrete design and implementation of these interfaces may structure them further, e.g. by splitting them up according to the different event types that messages and timer events cause.

AgentCore is typically provided by a concrete agent toolkit in which it is integrated in the core functionality of the agency of a concrete agent platform. Its capabilities are usually available to agent programmers in the form of a common superclass that can be used for defining custom software agents, e.g. class `jade.core.Agent` in JADE [77] or class `java.lang.Runnable` in Tracy (Braun and Rossak [18]). The presentation of AgentCore is rather abstract. A more detailed discussion of the component is out of the scope of this thesis as it would involve agent-toolkit-specific issues.
The next common component of both agent types is StateMachine. It models a finite state machine, e.g. based on the State pattern (see Gamma et al. [56]). StateMachine is invoked by AgentCore in order to activate or resume the software agent for processing incoming events (from the agent’s mailbox or a timer component). Moreover, StateMachine triggers AgentCore for performing actions external to the agent (interacting with the agent’s mailbox or setting up a timer). The finite state machine of StateMachine codifies the behaviour of an agent type. Both finite state machines are described in more detail in Section 7.2.

There are a number of additional components of each agent type. KnowledgeBase facilitates a data container for storing the state of a software agent and information about the agent’s environment (e.g. availability of or relationships to other software agents). The contents of KnowledgeBase is manipulated by StateMachine, and is queried by DecisionMaking in the case of the service consumer agent type. In addition, it may also be manipulated by DecisionMaking in the case of the service provider agent type.

KnowledgeBase embodies a structured data container. However, the internal design of KnowledgeBase is out of the scope of this thesis because it strongly depends on the API of a specific agent toolkit and the particular implementation of decision-making and reasoning capabilities.

Furthermore, DecisionMaking provides a toolbox of different decision-making capabilities that a software agent can utilise for proactively initiating actions or reactively responding to external events. Decision-making and reasoning capabilities are a core feature of software agents. They are adapted in this thesis to the requirements of on-demand service composition in open environments, the market-based control of the open market concept (Section 4.3.1) and the interaction protocols of the on-demand coalition formation concept (Section 6.4). The capabilities of DecisionMaking are requested by StateMachine.

Decision-making and reasoning capabilities are described in the following sections of this chapter. The service consumer agent type employs the capabilities of DecisionMaking in the form of automated planning (Section 7.3) and for ranking and selecting the most beneficial composite service proposal (Section 7.6). The service provider agent type incorporates decision-making and reasoning capabilities in the form of capability matchmaking (Section 7.4), expected utility calculation (Section 7.5) and for deciding the completeness
of a coalition (Section 7.6).

In contrast to service provider agents, service consumer agents also comprise a User-Interface component. UserInterface allows a human who is associated with a service consumer agent to interact with that agent. UserInterface can be used for triggering activities in StateMachine, e.g., for creating and disseminating a composite service request. The interactions between StateMachine and UserInterface are bi-directionally. StateMachine triggers UserInterface to display information (e.g., results, warnings and error messages) about its state to the human user. UserInterface is treated as a black box. It is out of the scope of this thesis because its internal design and implementation clearly depend on a concrete application. However, this thesis is concerned with the concept of on-demand service composition in open environments in general. An example of a graphical user interface of a service consumer agent for travel planning is presented in Chapter 8.

Service provider agents may provide user interfaces as well, e.g., for setting up and configuring a service provider agent to act on behalf of the specific services of a service providing organisation. However, these user interfaces are out of the scope of this thesis. They merely represent a technical detail. They have no tangible impact on the concept of on-demand service composition in open environments.

7.2 Agent-internal Behaviour

This section outlines an important requirement for the model of a generic finite state machine before introducing concrete instances of it for the service consumer agent type and the service provider agent type. A finite state machine in general is an abstract model of the behaviour of a system or component with a well-defined finite set of states, well-defined transitions between states and well-defined actions and conditions that trigger those transitions. In the context of this thesis, the set of states of the finite state machine for each agent type corresponds to the set of agent roles defined in Section 6.2. Software agents of both types change their states according to the status of their interactions with other software agents during the on-demand service composition process. In case of the service consumer agent type, the state of a software agent also depends on the status of its interactions with an associated human user.

The alignment of the set of states with the set of agent roles adapts the modelling of the
agent-internal behaviour to the characteristics of the interaction protocols; in particular, the required agent roles. A software agent type, of course, may be modelled with a different set of states.

It is assumed that the reader is familiar with the basic concepts of finite state machines. Thus this section does not present an introduction to the theoretical background of finite state machines nor the description of a modelling technique for finite state machines. In the remainder of this section, UML state machine diagrams in combination with pseudocode listings are used for specifying finite state machines and their states. The programmatic realisation of a state defines all valid actions (including associated conditions) that can be performed in the state and that may manipulate the knowledgebase of the software agent or trigger the transition to another state.

However, it is important to emphasise one particular characteristic of the states of the finite state machine described in the following. This thesis is based on the weak notion of agency which defines software agents to be both reactive and proactive. In order to stress this fact, states in this thesis are defined with two separate parts called reactive behaviour and proactive behaviour as shown in Listing 7.1.

**Listing 7.1: Pseudocode of the definition of the class State.**

```plaintext
1 CLASS STATE
2   REACTIVE–BEHAVIOUR( message )
3   PROACTIVE–BEHAVIOUR()
4 ENDCLASS STATE
```

The reactive behaviour of a state is triggered by agent-external events, namely, new incoming messages. It accepts a message as an argument. The proactive behaviour is executed in case agent-internal events occur, such as timer events. It does not accept arguments. Moreover, the proactive behaviour should be triggered if a software agent has no messages left to process and is about to become passive (see the main loop of a software agent in Listing 7.2). This procedure increases the ability of the software agent to achieve its design goals. However, the chosen balance between the reactive and proactive behaviours in the example of Listing 7.2 is only one of several alternatives. Proactive behaviours may be triggered more often, e.g. after the execution of every reactive behaviour (swapping lines 5 and 6 in Listing 7.2).
7.2. AGENT-INTERNAL BEHAVIOUR

Listing 7.2: Pseudocode of an example main loop.

```
BEGIN MAIN−LOOP()
  WHILE agent is not terminated AND (new message event OR new timer event occurred)
  FOR every new message
    CALL REACTIVE−BEHAVIOUR( message )
  ENDFOR
  CALL PROACTIVE−BEHAVIOUR()
ENDWHILE
END MAIN
```

7.2.1 Service Consumer Agent

This section describes the finite state machine of the StateMachine component of the service consumer agent type. The finite state machine comprises four states that correspond to the agent roles defined in Section 6.2.1. Figure 7.3 gives an overview of the finite state machine with a UML state machine diagram.

![UML state machine diagram of the StateMachine component of the service consumer agent type.](image)

Figure 7.3: UML state machine diagram of the StateMachine component of the service consumer agent type.

The diagram has been simplified for the sake of brevity and clarity. Not all actions and conditions of the actual finite state machine that are associated with state transitions are illustrated. Moreover, the UML state machine diagram type is not capable of modelling the particular characteristics of the states described above. It does not have the means for distinguishing between reactive and proactive state behaviours. Additional information
about actions and conditions in the states of the finite state machine and aspects of reactive and proactive behaviours of the states are presented with pseudocode listing for every state in the following sections.

**Listener State**

The initial state is the *listener* state. This is the state of a service consumer agent on creation. The knowledgebase of the service consumer agent is in its initial configuration. The name of the state suggests that the service consumer agent “listens” to its associated user for an input in the form of a service request. The state does not handle incoming messages, as illustrated in the pseudocode listing of the reactive behaviour of the listener state (Listing 7.3).

**Listing 7.3: Pseudocode of the service consumer agent listener state reactive behaviour.**

```plaintext
BEGIN LISTENER−STATE−REACTIVE−BEHAVIOUR( message )
/* empty, no actions */
END LISTENER−STATE−REACTIVE−BEHAVIOUR
```

The proactive behaviour of the listener state as shown in Listing 7.4 includes two activities that are triggered by a user input. If the service consumer agent receives a correct service request input from its associated user, it changes to the *planner* state. If the service request input is incorrect the service consumer agent remains in the listener state but informs its associated user of the error.

**Listing 7.4: Pseudocode of the service consumer agent listener state proactive behaviour.**

```plaintext
BEGIN LISTENER−STATE−PROACTIVE−BEHAVIOUR()
check knowledgebase for new user−defined service request
IF service−request exists AND input is correct THEN
change to planner state
ELSE
update user interface
ENDIF
END LISTENER−STATE−PROACTIVE−BEHAVIOUR
```

**Planner State**

The service consumer agent engages in an automated planning activity (see Section 7.3) in order to create a valid composite service request from the user input. The knowledge-
base supports automated planning with information that describes the particular problem domain (e.g. travel planning) in which planning takes place. The automated planning capability uses this information for the planning process and stores valid plans it has inferred in the knowledgebase. Automated planning is a proactive activity of the service consumer agent. The planner state does not perform activities with its reactive behaviour (see Listing 7.5).

Listing 7.5: Pseudocode of the service consumer agent planner state reactive behaviour.

```
BEGIN PLANNER-STATE-REACTIVE-BEHAVIOUR( message )
    /* empty, no actions */
END PLANNER-STATE-REACTIVE-BEHAVIOUR
```

The outcome of the planning process affects the state of the service consumer agent. If planning is unsuccessful, e.g. no valid composite service request could be found, the service consumer agent informs its associated user, resets its knowledgebase and changes back to the listener state. In contrast, if the planning activity results in at least one valid plan, the service consumer agent has already stored it as a composite service request in the knowledgebase. It then sends out a composite-service-request message, sets up a conversation object for handling the message timeout and changes to the requester state. The proactive behaviour of the planner state is depicted in Listing 7.6.

Listing 7.6: Pseudocode of the service consumer agent planner state proactive behaviour.

```
BEGIN PLANNER-STATE-PROACTIVE-BEHAVIOUR()
    perform automated planning
    IF planning successful THEN
        store the composite-service-request
        send a composite-service-request message
        set conversation-status: active
        change to requester state
    ELSE
        update user interface
        reset knowledgebase
        change to listener state
    ENDIF
END PLANNER-STATE-PROACTIVE-BEHAVIOUR
```
7. SOFTWARE AGENT DESIGN

Requester State

The service consumer agent waits passively in the requester state for composite service proposals that fulfil the previously stated composite service request. The service consumer agent waits until the message timeout of the original composite-service-request message is due. The reactive behaviour of the requester state handles incoming composite-service-proposal messages as presented in Listing 7.7. It adds a composite service proposal to the knowledgebase if the proposal is received on time.

**Listing 7.7:** Pseudocode of the service consumer agent requester state reactive behaviour.

```plaintext
1 BEGIN REQUESTER–STATE–REACTIVE–BEHAVIOUR( message )
2 IF conversation–status = active THEN
3     CASE message OF
4         composite–service–proposal:
5             add composite service proposal to knowledgebase
6     ENDCASE
7 ENDIF
8 END REQUESTER–STATE–REACTIVE–BEHAVIOUR
```

The proactive behaviour of the requester state as depicted in Listing 7.8 gets activated once the message timeout of the original composite service request message is due. Firstly, the behaviour checks the service consumer agent’s mailbox for any further composite service proposals. If it detects any, the respective proposals are stored in the knowledgebase.

**Listing 7.8:** Pseudocode of the service consumer agent requester state proactive behaviour.

```plaintext
1 BEGIN REQUESTER–STATE–PROACTIVE–BEHAVIOUR()
2 IF conversation–status = timed–out THEN
3     check mailbox for further composite–service–proposal messages
4     IF messages detected THEN add composite service proposals to knowledgebase ENDIF
5     IF composite service proposals exist in knowledgebase THEN
6         update user interface
7         reset conversation
8         change to contractor state
9     ELSE
10        update user interface
11        reset knowledgebase
12        reset conversation
13        change to listener state
14 ENDIF
15 END REQUESTER–STATE–PROACTIVE–BEHAVIOUR
```
The behaviour then checks whether there are any composite service proposals stored in the knowledgebase. If there are, the service consumer agent sends update information to the user interface, resets the conversation object and changes to the contractor state. If there are no composite service proposals in the knowledgebase, the composite service request terminates unsuccessfully. In this case the behaviour triggers a status update of the user interface, resets the knowledgebase and resets the conversation object. The state of the agent then changes to listener.

**Contractor State**

The contractor state facilitates the creation of a contract between the service consumer agent and the winning coalition of service provider agents. The reactive behaviour of the state is shown in Listing 7.9.

**Listing 7.9: Pseudocode of the service consumer agent requester state reactive behaviour.**

```
BEGIN CONTRACTOR–STATE–REACTIVE–BEHAVIOUR( message )

CASE conversation–status OF
    replied:
        CASE message OF
            composite–service–confirm:
                /* contract has been successfully created */
                update knowledgebase
                update user interface
                IF more composite service proposals exist in knowledgebase THEN
                    send composite–service–reject message to all remaining leaders
                ENDIF
            ENDCASE
        ENDCASE

END

END CONTRACTOR–STATE–REACTIVE–BEHAVIOUR
```

The reactive behaviour gets executed when the service consumer agent has received a composite-service-confirm message from the leader agent of the winning coalition. If so, a contract between the service consumer agent and the winning coalition has been successfully created. Thus the reactive behaviour notifies the user interface of the success and updates the knowledgebase with the contract. Furthermore, the reactive behaviour
7. SOFTWARE AGENT DESIGN

sends out composite-service-reject messages to all other leader agents if there are any other composite service proposals in the knowledgebase. Finally, the service consumer agent returns to the listener state after resetting the knowledgebase and the conversation object. The service consumer agent is then ready to process a new user-defined service request.

Before the service consumer agent can receive a composite-service-confirm message, it needs to send out a composite-service-accept message. This is done with the proactive behaviour of the contractor state shown in Listing 7.10.

Listing 7.10: Pseudocode of the service consumer agent requester state proactive behaviour.

```
BEGIN CONTRACTOR−STATE−PROACTIVE−BEHAVIOUR()
CASE conversation−status OF
    none:
        perform proposal ranking and select winning proposal
        send composite−service−accept message to leader of winning coalition
        conversation−status= active
    timed−out:
        update knowledgebase
        reset conversation
        WHILE more composite−service−proposals exist DO
            pick most beneficial composite−service−proposal from knowledgebase
            IF timeout associated with proposal is not due THEN
                send composite−service−accept message to associated coalition (leader)
                set conversation−status: active
            BREAK
            ELSE update knowledgebase ENDF
        ENDFWHILE
        IF no composite−service−proposals exist THEN
            reset knowledgebase
            reset conversation
            update user interface
            change to listener state
        ENDFIELDCASE
END CONTRACTOR−STATE−PROACTIVE−BEHAVIOUR
```

If there is no conversation object in use, the service consumer agent has not yet sent the mentioned message. Thus the proactive behaviour triggers a decision-making and reasoning activity for ranking all available composite service proposals and for selecting the
most beneficial one. Based on the decision, the service consumer agent sends a composite-
service-accept message to the leader of the winning coalition and sets up a conversation
object for handling an associated message timeout.

If there is, however, a conversation object with status timed-out, a previous composite-
service-accept message has not been replied to on time. In this case the proactive be-
haviour updates the knowledgebase (e.g. removing the failed composite service proposal) and resets the conversation object. The proactive behaviour then attempts to contract the next most beneficial composite service proposal, as illustrated with the WHILE loop in Listing 7.10. As long as there are composite service proposals in the knowledgebase, the most beneficial one is picked. If the timeout that is associated with the proposal has not passed, the proactive behaviour sends a composite-service-accept message to the asso-
ciated leader agent and sets up a conversation object for handling an associated message timeout.

If there are no composite service proposals left in the knowledgebase, the service consumer agent cannot create a successful contract. In this case, the user interface is notified of the failure, the knowledgebase and conversation object are reset and the service consumer agent returns to the listener state. The service consumer agent is then ready to process a new user-defined service request.

7.2.2 Service Provider Agent

This section outlines the finite state machine of the StateMachine component of the service provider agent type. The states of the automaton correspond to the roles of the agent type defined in Section 6.2.2. The finite state machine is depicted in Figure 7.4 with a UML state machine diagram. As with the service consumer agent type, the diagram here is simplified for the sake of brevity and clarity. Additional information about all actions and conditions, as well as reactive and proactive behaviours of the states, is presented with pseudocode listings for every state in the following sections.

Idle State

The initial state is the *idle* state. This is the state of a service provider agent on creation. The knowledgebase of the service provider agent is in its initial configuration. In the idle
7. SOFTWARE AGENT DESIGN

Figure 7.4: UML state machine diagram of the StateMachine component of the service provider agent type.

state, the service provider agent is inactive. It waits passively for a composite-service-request message to arrive. If a composite-service-request message is received, the service provider agent performs the idle state’s reactive behaviour shown in Listing 7.11.

Listing 7.11: Pseudocode of the service provider agent idle state reactive behaviour.

```
BEGIN IDLE–STATE–REACTIVE–BEHAVIOUR( message )
CASE message OF
  composite–service–request:
  store composite service request in knowledgebase
  perform matchmaking of own service with composite service request
  IF matchmaking successful AND composite service request not timed out THEN
    IF coalition is complete THEN
      send composite–service–proposal message
      set conversation–status: active
      change to leader state
    ELSE
      send service–advertisement message
      change to candidate state
    ENDIF
  ELSE
    reset knowledgebase
  ENDIF
ENDCASE
END IDLE–STATE–REACTIVE–BEHAVIOUR
```
The reactive behaviour stores the request in the knowledgebase and then matches the service provider agent’s service with the service types of the composite service request. If the matchmaking is successful and the composite service request is not timed out, the reactive behaviour engages the agent in a coalition formation process. It checks whether the sole service of the service provider agent is necessary and sufficient for providing a composite service proposal (that is, whether the coalition of the service provider agent is already complete). If so, the reactive behaviour sends out a composite-service-proposal message and sets up a conversation object for handling the associated message timeout. The state of the service provider then changes to the leader state.

If the coalition of the service provider agent is not complete, a service-advertisement message is sent out in order to advertise the agent’s service and to get in contact with potential collaboration partners. After sending this message, the service provider agent changes to the candidate state. Moreover, if the matchmaking activity terminates without a successful matching, the service provider agent is not capable of contributing to a composite service proposal. Thus the reactive behaviour resets the agent’s knowledgebase and the agent remains in the idle state.

Since the service provider agent is passively waiting for composite-service-request messages in the idle state, it does not perform any actions with the proactive behaviour of the state, as shown in Listing 7.12.

**Listing 7.12: Pseudocode of the service provider agent idle state proactive behaviour.**

```plaintext
BEGIN
IDLE-STATE-PROACTIVE-BEHAVIOUR()

/\ empty, no actions */

END IDLE-STATE-PROACTIVE-BEHAVIOUR
```

**Engaged State**

The engaged state is an abstract state that expresses some common properties for all derived sub-states. For example, all sub-states handle incoming service-advertisement messages (see Listing 7.13) and terminate the coalition formation process if the message timeout of the original composite service request is due (see Listing 7.14).

If a service-advertisement message is received, the reactive behaviour of the engaged state triggers matchmaking activity in order to match the advertised service with the service types of the composite service request. The matching is successful if the advertised
service matches at least one of the service types of the request. In this case the service advertisement is added to the service provider agent’s knowledgebase.

Listing 7.13: Pseudocode of the service provider agent idle state proactive behaviour.

```plaintext
BEGIN ENGAGED−STATE−REACTIVE−BEHAVIOUR( message )
  IF composite−service−request not timed out THEN
    CASE conversation−status OF
      timed−out:
        update knowledgebase
        reset conversation
      CASE message OF
        service−advertisement:
          perform matchmaking
          IF matchmaking successful THEN add service−advertisement to knowledgebase ENDIF
      ENDCASE
    none:
      CASE message OF
        service−advertisement:
          perform matchmaking
          IF matchmaking successful THEN add service−advertisement to knowledgebase ENDIF
      ENDCASE
  ENDCASE
END ENGAGED−STATE−REACTIVE−BEHAVIOUR()
```

The message timeout of the original composite service request may pass without the service provider agent becoming a member of a complete coalition. In this case, the proactive behaviour of the engaged state resets the service provider agent (particularly its knowledgebase and conversation object) and returns the service provider agent to the idle state. Afterwards the service provider agent again waits passively for incoming composite-service-requests messages.

Listing 7.14: Pseudocode of the service provider agent idle state proactive behaviour.

```plaintext
BEGIN ENGAGED−STATE−PROACTIVE−BEHAVIOUR()
  IF composite−service−request timed out THEN
    reset conversation
    reset knowledgebase
    change to idle state
  ENDIF
END ENGAGED−STATE−PROACTIVE−BEHAVIOUR()
```
Candidate State

The candidate state is a sub-state of the engaged state. In the candidate state, a service provider agent has engaged in the coalition formation process but has not yet established concrete relationships with other service provider agents. Therefore, the service provider agent attempts to initiate a coalition or it joins an existing coalition. The reactive behaviour of the candidate state (Listing 7.15) handles incoming requests and replies separately. In the case of an incoming reply (see line 8 of Listing 7.15), the reactive behaviour distinguishes between a coalition-accept message and a coalition-reject message. The reactive behaviour updates the knowledgebase of the service provider agent and resets the conversation object in both cases. However, a coalition-accept message indicates that a communication partner has joined the coalition of the service provider agent. Therefore, the agent changes to the leader state. A coalition-reject message signals an unsuccessful attempt to create a coalition. No further actions are taken.

**Listing 7.15:** Pseudocode of the service provider agent candidate state reactive behaviour.

```
1 BEGIN CANDIDATE−STATE−REACTIVE−BEHAVIOUR( message )
2   IF composite−service−request not timed out THEN
3     CASE conversation−status OF
4       replied:
5         CASE message OF
6           coalition−accept:
7             update knowledgebase
8             reset conversation
9             change to leader state
10           coalition−reject:
11             update knowledgebase
12             reset conversation
13         ENDCASE
14       timed−out:
15         update knowledgebase
16         reset conversation
17         CALL CANDIDATE−STATE−HANDLE−MESSAGE( message )
18       none:
19         CALL CANDIDATE−STATE−HANDLE−MESSAGE( message )
20     ENDCASE
21 ENDF
22 END CANDIDATE−STATE−REACTIVE−BEHAVIOUR
```
7. SOFTWARE AGENT DESIGN

The reactive behaviour of the candidate state also handles incoming request messages by calling an auxiliary procedure. If the conversation object is in status timed-out, the reactive behaviour first updates the knowledgebase and resets the conversation object.

An auxiliary procedure is presented in Listing 7.16. It handles incoming coalition-request messages. The first activity is the detection and matchmaking of unknown service-advertisements that are part of the coalition-request message. Successfully matched service-advertisements are added to the knowledgebase. Secondly, the reactive behaviour updates the knowledgebase and validates the requesting coalition. If the requesting coalition is more beneficial than the agent’s current coalition, it sends a coalition-accept message. If this message is sent before the associated message timeout is due, the service provider agent updates its knowledgebase and changes to the member state. The service provider agent has successfully joined an existing coalition. If the requesting coalition is not more beneficial than the current one, the reactive behaviour triggers a coalition-reject message.

Listing 7.16: Pseudocode of the service provider agent candidate state reactive behaviour auxiliary procedure.

```
BEGIN CANDIDATE−STATE−HANDLE−MESSAGE( message )
    CASE message OF
        coalition−request:
            WHILE unknown service−advertisement exists DO
                perform matchmaking
                IF matchmaking successful THEN add service−advertisement to knowledgebase ENDIF
            ENDWHILE
            update knowledgebase
            validate requesting coalition
            IF coalition most beneficial THEN
                send coalition−accept message
                IF coalition−accept message sent on time THEN
                    update knowledgebase
                    change to member state
                ENDIF
            ELSE send coalition−reject message ENDIF
        END CASE
    END
END CANDIDATE−STATE−HANDLE−MESSAGE
```

The proactive behaviour of the candidate state is defined in Listing 7.17. If a previous conversation is timed-out, the proactive behaviour updates the knowledgebase accordingly
and resets the conversation object. The proactive behaviour then initiates the creation and sending of a coalition-request message, if a sufficient number of service-advertisements are available in the knowledgebase. In this case, the most beneficial coalition partner is determined and the message is sent out. The conversation object is set up for handling the associated message timeout.

Listing 7.17: Pseudocode of the service provider agent candidate state proactive behaviour.

```plaintext
BEGIN CANDIDATE-STATE-PROACTIVE-BEHAVIOUR()
    IF conversation-status= timed-out THEN
        update knowledgebase
        reset conversation
    ENDIF
    IF sufficient number of advertisements THEN
        determine potential partner
        send coalition-request message
        conversation-status = active
    ENDIF
END CANDIDATE-STATE-PROACTIVE-BEHAVIOUR
```

Leader State

The leader state is a sub-state of the engaged state. If a service provider agent adopts this state it already has successfully initiated a coalition and now seeks to complete it. The reactive behaviour of the state is shown in Listing 7.18. It handles incoming request messages and reply messages separately. Coalition-accept and coalition-reject messages are handled similarly to what is described above for the reactive behaviour of the candidate state. The difference is that the leader sends out a coalition-inform message to all current member agents of its coalition to update them about the decision of the communication partner. Furthermore, the reactive behaviour of the leader state handles composite-service-accept messages, composite-service-reject messages and coalition-complete messages. A coalition-accept message indicates that the requesting service consumer agent decided that the coalition of the leader agent is the winner. In this case the leader attempts to send a composite-service-confirm message. However, this message must be sent before the timeout of the composite-service-accept message is due. If so, a contract has been established successfully and the reactive behaviour triggers a coalition-confirm message to all members. If no contract has been created, the reactive behaviour triggers a
coalition-disconfirm message to all members. The service provider agent then returns to the idle state and is ready again to process new composite service requests. The receiving of a composite-service-reject message also effectively disbands a coalition after the leader has sent a coalition-disconfirm message to all its members.

**Listing 7.18**: Pseudocode of the service provider agent leader state reactive behaviour.

```
BEGIN LEADER–STATE–REACTIVE–BEHAVIOUR( message )
   IF composite–service–request not timed out THEN
      CASE conversation–status OF
         replied:
            CASE message OF
               coalition–accept:
                  update knowledgebase
                  reset conversation
                  send coalition–inform message to all members
               coalition–reject:
                  update knowledgebase
                  reset conversation
                  send coalition–inform message to all members
               composite–service–accept:
                  update knowledgebase
                  reset conversation
                  send composite–service–confirm message
                  IF composite–service–confirm message sent on time THEN
                     send coalition–confirm message to all members
                  ELSE
                     send coalition–disconfirm message to all members
                  ENDIF
               ENDIF
               reset knowledgebase
               change to idle state
               composite–service–reject:
                  update knowledgebase
                  reset conversation
                  send coalition–disconfirm to all members
               reset knowledgebase
               change to idle state
               coalition–complete–inform:
                  update knowledgebase
                  IF all votes received AND all votes positive THEN
                     send composite–service–proposal message
                     IF composite–service–proposal message sent on time THEN
                        send coalition–registration–confirm message to all members
                        IF all coalition–registration–confirm messages sent on time THEN
```

222
Coalition-complete-inform messages carry the votes of member agents that determine whether all member agents perceive a coalition as complete or not. The leader agent collects the votes from coalition members. If all members confirm the coalition to be complete, the leader attempts to send a composite service proposal to the requesting service consumer agent. If this message arrives on time and the leader is also able to send subsequently a coalition-registration-confirm message to all members on time, the coalition has successfully placed a proposal. The leader then sets up a conversation object to handle the associated message timeout. In all other cases, the coalition did not place a proposal successfully. The leader therefore disconfirms with the coalition-registration-disconfirm message. If the composite-service-proposal message arrived after the message timeout, the coalition formation process terminates for the service provider agent and the agent changes to the idle state.

The handling of incoming requests shown in Listing 7.19 is similar to the handling of requests in the candidate state. The difference in handling a coalition-request is that the service provider agent sends out a leader-leave message to all the members of its coalition if the requesting coalition is more beneficial. Moreover, the reactive behaviour
handles member-leave messages. If such a message is received, the service provider agent
updates its knowledgebase and, if no members are left in its coalition, changes back to
the candidate state.

**Listing 7.19:** Pseudocode of the service provider agent leader state proactive behaviour.

```plaintext
BEGIN LEADER-STATE-PROACTIVE-BEHAVIOUR()
    IF conversation-status = timed-out THEN
        update knowledgebase
        reset conversation
    ENDIF
    IF coalition complete THEN
        send coalition-complete-query to all members
        set conversation-status: active
    ELSE
        determine potential partner
        send coalition-request message
        set conversation-status: active
    ENDIF
END LEADER-STATE-PROACTIVE-BEHAVIOUR
```

The task of the proactive behaviour of the leader state, as presented in Listing 7.19,
is to test whether the coalition is complete and if it is not complete, to trigger a coalition
extension. If the coalition is complete, the service provider agent sends out a coalition-
complete-query message to all members and sets up a conversation object for handling the
associated message timeout. If the coalition is not complete, the proactive behaviour trig-
gers reasoning about the next prospective coalition partner and sends a coalition-request
message according to the decision. It also sets up a conversation object for handling the
associated message timeout.

**Listing 7.20:** Pseudocode of the service provider agent leader state reactive behaviour auxiliary
procedure.

```plaintext
BEGIN LEADER-STATE-HANDLE-MESSAGE( message )
    CASE message OF
        coalition-request:
            perform matchmaking
            IF matchmaking successful THEN add service—advertisement ENDIF
            update knowledgebase
            validate requesting coalition
            IF coalition most beneficial THEN
                send coalition—accept message
```
7.2. AGENT-INTERNAL BEHAVIOUR

IF coalition—accept message sent on time THEN
  send leader—leave message to all members
  update knowledgebase
  change to member state
ENDIF
ELSE send coalition—reject message ENDIF
member—leave:
  update knowledgebase
  IF no members in coalition THEN
    change to candidate state
  ENDIF
ENDCASE
END LEADER−STATE−HANDLE−MESSAGE

Member State

The member state is a sub-state of the engaged state. If a service provider agent acts in the member state, it has joined a coalition and expects the leader to expand the coalition to its completeness. The reactive behaviour of the member state is defined in Listing 7.21.

Listing 7.21: Pseudocode of the service provider agent member state reactive behaviour.

BEGIN MEMBER−STATE−REACTIVE−BEHAVIOUR( message )
  IF composite−service−request not timed out THEN
    CASE conversation−status OF
      replied:
        CASE message OF
          coalition−registration−confirm:
            update knowledgebase
            reset conversation
          coalition−registration−disconfirm:
            reset knowledgebase
            reset conversation
            change to idle state
          timed−out:
            update knowledgebase
            reset conversation
          CALL MEMBER−STATE−HANDLE−MESSAGE( message )
        ENDCASE
      no:
        CALL MEMBER−STATE−HANDLE−MESSAGE( message )
    ENDCASE
  ENDIF
END MEMBER−STATE−REACTIVE−BEHAVIOUR
The reactive behaviour handles incoming request messages and reply messages separately. Replies are expected in the form of coalition-registration-confirm and coalition-registration-disconfirm messages. The former message indicates that the coalition has successfully placed a composite service proposal with the requesting service consumer agent. The reactive behaviour updates the knowledgebase, resets the conversation object and waits for a confirmation whether the coalition is the winner. The latter message conveys that the coalition failed to register a composite service proposal. The proactive behaviour resets the knowledgebase and the conversation object and triggers the service provider agent to change to the idle state.

The handling of request and update messages in the member state is supported by the auxiliary procedure shown in Listing 7.22. As before, a coalition request message may result in the software provider agent joining another coalition. If so, the agent sends out a member leave message to the members and the leader of its previous coalition. The coalition-inform message, member-leave message and coalition-registration-disconfirm message trigger the update of the service provider agents knowledgebase. The leader-leave message indicates the disbanding of the coalition. The reactive behaviour triggers an update of the knowledgebase and the change to the candidate state. The coalition-complete-query message triggers the reactive behaviour to perform decision-making in order to determine the completeness of the coalition from the service provider agents perspective. The result is returned to the coalition leader with a coalition-complete-inform message. Moreover, a conversation object is set up for handling the associated timeout. The coalition-confirm message signals that the coalition has successfully created a contract with a requesting service consumer agent. The reactive behaviour updates or resets the knowledgebase and changes the agent’s state to idle. The coalition-disconfirm message indicates that the coalition could not create a contract with the requesting service consumer agent. The reactive behaviour resets the knowledgebase and the service provider agent changes to the idle state.

**Listing 7.22:** Pseudocode of the service provider agent member state reactive behaviour auxiliary procedure.

```plaintext
1 BEGIN MEMBER−STATE−HANDLE−MESSAGE( message )
2     CASE message OF
3         coalition−request:
```

226
7.2. AGENT-INTERNAL BEHAVIOUR

A service provider agent in the member state has chosen to react only. Therefore, the proactive behaviour of the state member does not perform any actions on top of the actions of the proactive behaviour of the engaged state (see Listing 7.23).

**Listing 7.23:** Pseudocode of the service provider agent member state proactive behaviour.

```
BEGIN MEMBER—STATE—HANDLE—MESSAGE

perform matchmaking
IF matchmaking successful THEN add service—advertisement ENDIF
update knowledgebase
validate requesting coalition
IF coalition most beneficial THEN
send coalition—accept message
IF coalition—accept message sent on time THEN
send member—leave message to coalition leader and all members
update knowledgebase
ENDIF
ELSE send coalition—reject message ENDIF
coalition—inform:
update knowledgebase
leader—leave:
update knowledgebase
change to candidate state
member—leave:
update knowledgebase
coalition—complete—query:
query knowledgebase
send coalition—complete—inform message
IF coalition—complete—inform message sent on time AND coalition complete THEN
set conversation—status: active
ENDIF
coalition—registration—disconfirm:
update knowledgebase
coalition—confirm:
update and/or reset knowledgebase
knowledgebase
change to idle state
coalition—disconfirm:
reset knowledgebase
change to idle state
ENDCASE
END MEMBER—STATE—PROACTIVE—BEHAVIOUR

/* empty, no actions */
END MEMBER—STATE—PROACTIVE—BEHAVIOUR
```
7. SOFTWARE AGENT DESIGN

7.3 Automated Planning

This section introduces the notion of automated planning. It starts with a brief introduction to the theory of automated planning before outlining the application of automated planning in the context of on-demand service composition.

As Klusch [91] states, an automated planning problem can in general be defined as a triple \( (I, G, A) \) of an initial state \( I \in S \), a goal state \( G \in S \) and a set \( A \) of possible actions in the state–action space of a problem domain. Each action \( a_i \in A \) is a valid transition from one state to another. It is defined with preconditions that need to be met in order to apply the action and effects that will be in place after performing the action. Every state \( S_j \in S \) is specified with a set of propositions that are true or false in this state. The basic problem to be solved by automated planning is to generate a sequence of actions \( a_i \) that represents a complete transition from the initial state \( I \) into the goal state \( G \), or in other words transform the propositions of the initial state into the ones of the goal state.

A number of classical planning methods exist, each using a different strategy for generating a valid action sequence for a given problem. **Forward-chaining** (used, for example, in STRIPS by Fikes and Nilsson [47]) starts at the initial state and applies a transformation function \( f : S \times A \mapsto S \) that maps a state \( S_j \) and the transition by a particular action \( a_i \) into a new state \( S_k \). Function \( f \) is then applied on the new state \( S_k \) in order to derive the next state, and so on until the goal state has been reached.

Forward-chaining implements backtracking in case it is not possible to apply any of the available actions to a particular state. In this case, the algorithm drops the current state and backtracks to the previous state with respect to the sequence of actions concatenated so far. Determining whether an action can be applied to a state is based on practical logic reasoning (e.g. inference by resolution) that establishes whether the preconditions of an action are satisfied by a state. The outcome of a planning process is a sequence of actions \( (a_1 \ldots a_n) \) which forms a valid plan for a given planning problem if \( G = f(I, (a_1 \ldots a_n)) \), that is, the propositions of the goal state are satisfied by the propositions derived from applying the sequence of actions on the initial state.

Another way to infer an action sequence is to use **backward-chaining**, which is basically reverse forward-chaining. The reasoning starts at the goal state and aims to find a sequence of actions to reach the initial state. Both approaches are based on the principle
of practical means–end reasoning. They are aimed at creating a complete solution for a
given problem with respect to available knowledge about a problem domain.

Classical planning methods provide generic general purpose planning algorithms that
are formulated independently of specific problem domains. Their generality, however,
is traded-off with high computational costs due to the exponential complexity of the
search space of possible combinations of actions. Furthermore, they are based on the
assumption of full knowledge about the initial and goal states and assert that no changes
in the environment occur and that the effects of an action are deterministic. Classical
automated planning is also called static planning. Plan creation and plan execution are
strictly separated into two successive steps.

In contrast, by interleaving the plan creation and plan execution steps dynamic plan-
ning methods support automated planning even with changes in the environment. Reac-
tive planning, in Agre and Marinova [3], infers always only a single action from a current
state and executes this action immediately. Thus reactive planning can take the effects of
a current action and other possible changes into account when inferring the next action.
This approach may allow for a better adaptation to dynamic environments. However, it
does not support means–end reasoning. Due to the interleaved inference and execution of
actions, it can never be determined whether the planning process terminates with success.
Accordingly, already performed actions would have to be rolled back in case the planning
process terminates with failure. This may be impractical in some problem domains.

Numerous classical static and dynamic planning techniques, as well as extension and
new developments for planning under uncertainty, have been intensively studied. A com-
prehensive introduction to automated planning can be found in Ghallab et al. [60].

Automated planning techniques are attractive for dynamic service composition due
to the similarities between the state–action space of a planning problem and the state–
service space of a service composition problem. In this context, a request for a composite
service can be understood as a goal state that is achieved by combining and executing a
sequence of different services. Consequently, numerous approaches have been developed
applying planning techniques to solve the Web-service or Semantic Web-service composi-
tion problem (see Klusch [91] for an overview). However, following Srivastava and Köhler
[157], the problem of dynamic service composition is too complex for a naive adoption of
automated planning techniques for the following reasons:

- **Incomplete information about the initial state** (e.g. if a travel service offers discount fares for a specific connection)

- **Possible dynamic changes in the environment at planning time** (e.g. all discount fares for a specific connection are sold out)

- **Non-deterministic effects of actions due to conditional effects that are unknown at planning time** (e.g. different discounts depending on the number of passengers)

- **Non-atomic actions due to the complexity of data and control flow dependencies as, for example, with complex message exchange patterns** (e.g. the conditional order of providing a travel service with necessary inputs).

Besides these issues, the inherent computational complexity of classical planning techniques is inappropriate for practical solutions to dynamic service composition in open environments. Partial observability, dynamics and non-determinism in usually time-continuous spaces make it necessary to perform dynamic service composition in real-time based on a potentially large and variable number of available services. Moreover, an interleaved composition and enactment of services, as with reactive planning, is not desirable. Service composition is aimed at business process automation and thus should allow for process and cost control. Hence it is essential that a contract (e.g. SLA) is created prior to service execution in order to support the measuring and monitoring of contracted services and to provide a platform for detecting and clearing violations and errors at a technical and legal level.

In this thesis, automated planning is applied only for realising the planning activity (see Section 2.2) in the composite service life-cycle instead of the whole service composition process. This is because, as Srivastava and Köhler [157] point out, the entire problem of dynamic service composition is too complex for a purely automated planning-based solution. Automated planning can be applied for providing an automated way to break down the complexity of client-defined service requests and to transform them into proper business process definitions. This approach does not only offer advantages with regard to usability, flexibility and extensibility over rigid hard-coded workflow solutions. It also provides means to reduce aforementioned shortcomings.
Static automated planning is assumed in this thesis for the decomposition of problems in order to create one or perhaps several alternative composite service requests. A composite service request is not an executable plan in contrast to the original meaning of plans in automated planning. A composite service request represents a generic composite service comprising interlinked generic service types rather than concrete service instances. As such, it can be argued that there is no need for complete information about the initial state during the automated planning. In other words, the service composition process is split into two successive steps: firstly the creation of possible generic structures using automated planning, and secondly, the instantiation of the most beneficial structure based on additional “live” information about available services which is performed with agent-based coalition formation (Chapter 5).

The on-demand bottom-up approach of disseminating service advertisements in this thesis supports up-to-date information about participating service providers. It is no longer the problem of a central planner to obtain and maintain complete information about its environment. Instead it is up to the service providers to provide proper information and to guarantee that this information – including necessary resources and data – is available and correct. A service advertisement is a binding offer and an integral part of the on-demand service composition concept (Section 4.2).

This thesis is not aimed at contributing to the theory of automated planning. It is based on an existing universal planning method which is, however, adapted to meet the specific requirements of on-demand service composition in open environments. The result is a specialised planning method which is outlined in the following sections. Moreover, this thesis incorporates static planning. That means service composition and service enactment are strictly separated. In particular, dynamic replanning is out of the scope of this thesis. Finally, the difficult problem of eliciting goal specifications and user-defined preferences from human users is also acknowledged here. However, this problem is also out of the scope of this thesis. A graphical user interface-driven approach is outlined in Chapter 8.

7.3.1 HTN Planning

Service planning activity is implemented by applying hierarchical task network (HTN) planning. Hierarchical task network planning is a derivate of classical automated planning
with two major differences. Firstly, planning is not performed by the transition of atomic actions. The fundamental building blocks of HTN planning are atomic and compound tasks. **Atomic tasks** are associated with a concrete operation that can be directly executed. **Compound tasks** are complex tasks that cannot be directly achieved. They need to be further decomposed. Secondly, the original goal state is replaced with a **task network**. A task network specifies a set of goal tasks (atomic or compound) and logical and temporal dependencies as well as binding constraints between them. Thus HTN planning supports explicit modelling of the domain knowledge of a particular problem domain.

The basic processing of an HTN planner is to reduce all compound tasks of a goal task network to atomic tasks that can be later executed by corresponding operators. The reduction is performed with methods. A method specifies a task network and a set of conditions under which the given task network can be applied to decompose a compound task. The task network of a method can include further compound tasks that need to be recursively reduced as well. Once the original goal task network contains only atomic tasks and a valid instantiation of a variable binding can be found, a complete plan has been generated successfully.

An HTN planning problem can be formalised as a 4-tupel \((d, I, O, M)\). A task network \(d\) specifies the goal to be accomplished. \(I\) is the set of propositions about the initial state. \(O\) is the set of operators that are associated with atomic tasks. \(M\) is the set of methods. The task network \(d = (t_1 \ldots t_n, \phi, \xi, \psi)\) where \(\forall t_i \in T\) form the set of tasks. \(\phi\) is the set of logical dependencies and \(\xi\) is the set of temporal dependencies between tasks \(t_i\) of \(d\). \(\psi\) defines a set of binding constraints between variables of tasks \(t_i\) in \(d\).

Every task \(t_i \in T\) is specified with a tupel \((n, x_1 \ldots x_n)\) where \(n\) denotes the name of the task and every \(x_i\) is a term representing an argument of the task. There are atomic tasks and compound tasks. Atomic tasks are the equivalent of actions in classical planning systems. An atomic task is a task that can be reduced by a planning operation \(o \in O\) which is defined as a triple \(((n, v_1 \ldots v_n), p_1 \ldots p_m, e_1 \ldots e_k)\) where \((n, v_1 \ldots v_n)\) denotes the head \(head(o)\) of the operation which comprises the operation’s name \(n\) and its parameters \(v_i\), as well as the operation’s set of preconditions \(p_i\) and the set of effects \(e_i\).

If the head \(head(o)\) of an operation unifies with (is equivalent to) a task’s name \(n\) and list of arguments \(x_i\) and if all preconditions \(p_i\) of the operation are satisfied with
7.3. AUTOMATED PLANNING

respect to the initial state $I$, an operation can be substituted for a task. The initial state $I$ is specified with a set of atoms that describe the state of the planning environment. It is manipulated during the planning process when a task is reduced by an operation. This is because the effects of the operation take effect (propositions that become true and propositions that become false). Every atomic task is always associated with only a single operation. In contrast, a compound task cannot be reduced by a planning operation.

Tasks can be further decomposed into a task network of subtasks. The decomposition of tasks is implemented with methods. A method $m \in M$ is defined as a triple $(n, x_1 \ldots x_n, p_1 \ldots p_m, d')$ where $(n, x_1 \ldots x_n)$ denotes the head $head(m)$ of the method. The method includes a name $n$ and a list of arguments $x_i$, a set of preconditions $p_i$ and a task network of subtasks of $m$ denoted by $d'$. If $head(m)$ unifies with (is equivalent to) a task’s name and arguments and all preconditions $p_i$ are satisfied, a method $m$ decomposes a task $t_i$.

There may be several methods that decompose a given task, more than one set of variable bindings that satisfy the preconditions of a method or multiple ways to accomplish a method due to different alternatives in its task network or the task network’s subtasks. Alternatives result in additional branches in an HTN planning search space. Branches increase the robustness of the planning process. If one alternative fails to produce a valid plan, the HTN planner backtracks in order to restart planning with a different branch. If planning was successful, the resulting plan $P = (o_1 \ldots o_n, \phi, \xi, \psi)$ is an instantiation of operators for the tasks of the original goal task network $d$. Listing 7.24 depicts a prominent example of an HTN planning procedure in the form of SHOP2 (Nau et al. [126]).

The use of pre-defined methods is sometimes seen as a major drawback of HTN planning because it prevents the solution of any arbitrary planning problem given its predetermined structure. It can be argued, however, that most, if not all, problem domains are structured – usually in a hierarchical fashion due to the common practice of modelling complex problems with decomposition based on the principle of separation of concerns. According to Ghallab et al. [60, ch.11], HTN planning is the most widely used planning method for implementing practical applications because of its properties. HTN planning offers an intuitive and structured method for domain experts to model a specific problem domain. It can be used for modelling composite service requests in the domain of dynamic
service composition. Hence elements of HTN planning share similarities with the elements
in the domain of service composition. Table 7.1 contrasts equivalent terms of HTN plan-
ning with the fundamental data structures of the on-demand service composition concept
(Section 6.3).

The application of HTN planning to the decomposition of client-defined goals and the
creation of generic composite services during on-demand service composition offers the
following benefits:

- Relieves users from the complexity of a problem domain and necessary knowledge
  when specifying a service request in the form of a goal instead of a complete com-
  posite service
7.3. AUTOMATED PLANNING

- Simple formulation of a problem request but with flexible solutions, in particular, avoidance of static goal templates or fixed workflow templates

- Domain experts control the complexity by tailoring a service composition problem to a particular problem domain with the effect of restricting the computational costs for goal decompositions (automated planning)

- High flexibility and extensibility in contrast to fixed workflow definitions by providing alternative ways to decompose a goal or to instantiate a service type with a concrete service

- Plans are the basis for monitoring and controlling composite services or business processes and support legal actions in case of violation

- Supports dynamic service matchmaking and selection.

Table 7.1: Different terms, same concepts; a comparison of terminology of HTN planning and the on-demand service composition concept.

<table>
<thead>
<tr>
<th>HTN planning</th>
<th>On-demand service composition</th>
</tr>
</thead>
<tbody>
<tr>
<td>goal task network</td>
<td>composite process</td>
</tr>
<tr>
<td>task</td>
<td>simple process</td>
</tr>
<tr>
<td>atomic task</td>
<td>atomic process</td>
</tr>
<tr>
<td>composite task</td>
<td>composite process</td>
</tr>
<tr>
<td>operation</td>
<td>simple process</td>
</tr>
<tr>
<td>method</td>
<td>composite process</td>
</tr>
<tr>
<td>plan</td>
<td>composite process</td>
</tr>
</tbody>
</table>

However, there are two major shortcomings of existing HTN planners in the context of on-demand service composition in open environments beyond the issues identified by Srivastava and Köhler [157]. Firstly, HTN planners work on the assumption that so-called world information services exist which can be queried about the world state without changing it. This assumption is unrealistic because of the partial observability and dynamics of open environments and the potential complexity of service components (e.g. see Peer [136]). In this thesis, information about world state becomes available with service advertisements in an ad hoc fashion. Service provider agents push information based
on their local knowledge and assure a certain service for a period of time (including its preconditions and effects). In other words, the approach of this thesis employs a set of interacting agents to solve a service composition problem instead of a single entity trying to reason about an open environment.

Secondly, Table 7.1 is incorrect in that a plan in HTN planning is merely a sequence of partially ordered operators in contrast to the complex control structures that are available for modelling composite services (see Section 6.3). So even though Sirin et al. [152] and Klusch et al. [94] outline the use of HTN planners for dynamic service composition of OWL-S modelled Semantic Web services in general, they also report difficulties with handling concurrency and other control structures. Klusch and Gerber [92] use a combination of fast forward-chaining and HTN planning. The approach only supports sequential plans. Klusch et al. [94] propose an HTN planner that converts OWL-S service descriptions into the planning description language PDDL for tackling the problem of Semantic Web-service composition. The approach does not support all control structures, such as choice and unordered set. It is also potentially NP-complete due to the infinite state–action space. Sirin et al. [152] outline the application of the SHOP2 planner for Semantic Web-service composition. The presented approach cannot handle the split and split–join constructs. In contrast McIlraith and Son [117] describe a planning approach to Web-service composition that is capable of handling concurrency (split and split–join). However, the approach is based on the Golog programming language and therefore requires that a problem is fully translated into Golog. Hence it is not directly applicable to the elements of the original problem domain.

The problems with concurrency in HTN planners, and SHOP2 in particular, stems from the sequential and random selection of the next succeeding task during the planning process. Figure 7.5 illustrates this issue with a simple example. Assume that there is a goal task network $d$ defined with the tasks $t_1, t_2, t_3$ with $t_2$ and $t_3$ organised in two parallel branches, an initial state $I$, operators $O$ and methods $M$ (Step 1).

The algorithm presented in Listing 7.24 commences by determining all non-preceded tasks (task $t_1$ in the example). Depending on whether task $t_1$ is an atomic or compound task, the algorithm then attempts to obtain an operation or a method that can be used to substitute the task. In the example, operation $o_1$ matches task $t_1$. The algorithm
7.3. AUTOMATED PLANNING

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
</table>
| 1)   | initial situation  
      \[ d = t_{1} \rightarrow t_{2} \]  
      \[ O = \{ o_{1}, o_{2}, o_{3} \} \quad M = \{ m_{1} \} \quad \{ t_{1}, t_{2} \rightarrow t_{22} \} \quad I = \text{initial} \] |
| 2)   | substitute task \( t_{1} \) with operation \( o_{1} \)  
      \[ d = o_{1} \rightarrow t_{1} \]  
      \[ l = l + (\text{effects of } o_{1}) \] |
| 3)   | substitute task \( t_{i} \) with subtask network of method \( m_{i} \)  
      \[ d = o_{1} \rightarrow t_{2i} \]  
      \[ l = l + (\text{effects of } o_{1}) \] |
| 4)   | substitute task \( t_{21} \) with operation \( o_{21} \)  
      \[ d = o_{21} \rightarrow t_{21} \]  
      \[ l = l + (\text{effects of } o_{21}) + (\text{effects of } o_{2}) \] |
| 5)   | substitute task \( t_{2} \) with operation \( o_{2} \)  
      \[ d = o_{2} \rightarrow t_{2} \]  
      \[ l = l + (\text{effects of } o_{2}) \]  
      \[ \text{failure! I is invalid for this substitution!} \] |

Figure 7.5: Simplified visualization of a planning process to illustrate the shortcomings of the SHOP2 algorithm with respect to split and split-join control structures.

substitutes them and updates the world state by adding the effects of operation \( o_{1} \) to the initial state \( I \) (Step 2). In Step 3, the algorithm again determines all unpreceded tasks. Assume that it picks task \( t_{2} \) and substitutes it with method \( m_{1} \) because it is a compound task. As a result of the method substitution, the algorithm searches for unpreceded tasks only in the new sub-network \( \{ t_{21}, t_{22} \} \) and finds task \( t_{21} \) (which gets substituted with operation \( o_{21} \) in Step 4). Accordingly, the world state is updated by adding the effects of operation \( o_{21} \) to the already altered initial state \( I \).

So far the algorithm has substituted two tasks and went into the upper branch of the split control structure. Thus the world state \( I \) is adjusted to the current position of the planning procedure in the branch located directly after operation \( o_{21} \). In Step 5, a problem can occur because of lines 13 and 5 of the algorithm. Given the example, line 13 results in the set \( \{ t_{3}, t_{22} \} \) of non-preceded tasks and because of the random selection in line 5, the algorithm may choose task \( t_{3} \) to proceed. However, the world state \( I \) is inappropriate to process tasks of the lower branch of the split control structure because it includes the effects of the upper parallel branch. In consequence, task \( t_{3} \) cannot be planned because the world state is crucial for matching the preconditions of operators (see line 7). The integrity of the planning process is violated.
The HTN planning procedure of this thesis takes a different approach in order to facilitate all OWL-S defined control constructs (see Section 6.3). A recursive algorithm is proposed in Listing 7.25 to remedy the problem with concurrency in general HTN planning (including SHOP2). The main idea is to ensure the integrity of the world state at every decomposition step and to guarantee consistent recursive decomposition of composite structures (sub-task networks) until the leaf nodes of the spanning decomposition tree have been reached and can be substituted with operators. In other words, the planning procedure handles every compound task (sub-task network) in a planning problem as a separate new planning problem whose outcome, if successful, is integrated in the overall solution. The following presents a number of assumptions of and differences between the proposed planning procedure with respect to existing HTN planners:

- A goal is specified with a task network \( d \) which is always a generic service request (see Section 6.3). Thus even if the goal comprises only a single generic service type, it is defined with a control structure, e.g. a sequence or unordered set

- The elements of parallel structures such as split and split-join are fully disjoint in their effects. Elements are generic atomic services and/or generic composite services

- The preconditions in loop structures such as repeat–while and repeat–until are equivalent to loop invariants that need to be true for every single iteration of a loop

- Conversions between the service-oriented domain (e.g. in the form of a OWL-S process model) and the planning domain (e.g. PDDL or SHOP2) are not required. The planning procedure, including all methods and tasks, is implemented using the data structures of the on-demand service composition problem domain of this thesis

- The planning procedure utilises all composite control structures, e.g. sequence, unordered, choice, loop and concurrency. It is in particular orientated towards the control structures defined by the OWL-S process model specification (see Section 6.3)

- The outcome of the recursive planner is not a sequential plan. The planning procedure preserves the composite structure of an original goal task network as well as of substituted methods.
The recursive planning algorithm starts with storing the initial state $I$ in a local variable $S$ for `split`, `split-join`, `choice` and `if-then-else` control structures. This is necessary to preserve the original state that is in place at the beginning of each existing branch. The algorithm is subdivided into two major parts. The first part checks whether the next chosen task can be reduced by an operation and the second part attempts to substitute a task with a method. If both parts fail, the search space is fully exploited and the algorithm
returns with an error message, as no valid composite service request could be found.

The first part of the algorithm (lines 6–15) is identical to the SHOP2 procedure in
Listing 7.24, with two exceptions. Firstly, the recursive planner does not select tasks and
operators on a random basis. Tasks are chosen with respect to the control structure and
the order it implies. Operators are chosen based on a ranking according to a similarity
heuristic (e.g. based on capability matchmaking). Secondly, the modification of the state
also depends on the current control structure. The global state \( I \) is updated in case of
sequence, unordered set and loops. The local state \( S \) is updated in case of concurrency
and choice structures. Finally, a task that has been successfully reduced by an operation
is marked in order to exclude it from further processing.

The second part (lines 17–29) differs from the original procedure in that it recursively
calls the algorithm to detect the next decomposition level. First of all, the algorithm ranks
all matching methods that unify with (are equivalent to) the task according to a similarity
heuristic (e.g. capability matchmaking). It then chooses the highest ranked method and
substitutes it for the task in question. If the subtask network of the method is not empty,
the algorithm starts traversing it by a recursive call and then waits until the traversing
is finished. Thus every subtask network is treated as a new planning problem. Recursion
may occur several times, resulting in a complete traversal of the decomposition tree from
the goal to its leaves in a kind of depth-first search (pre-order tree walk). Only when the
subproblem is successfully solved the algorithm updates its state at the current level with
respect to the given control structure. The update is very similar to the one performed in
the first part. Finally, after a task has been successfully substituted with a method, the
method head is marked in order to exclude it from further processing. Hence the method
head is the placeholder for its associated subtask network in the current control structure.

The algorithm terminates successfully if no unmarked tasks exist in every control
structure. The recursive planning procedure has no explicit return parameter. It uses
call-by-reference for the procedure call. Thus if a valid plan for a composite service
request exists, it can be referenced via the input parameter \( d \) of the procedure call. The
description of the algorithm may become more plastic with respect to dynamic service
composition if the reader substitutes the terms of the planning domain with notions from
the on-demand service composition concept as listed in Table 7.1.
The recursive planning procedure is executed by service consumer agents in this thesis. Since the service consumer agent does not have access to information about concrete services available during planning time, the operations of the planning procedure are not equivalent to concrete services (instances) but to generic service type declarations of a particular problem domain. The binding of operations (generic service types) to concrete service instances is performed during dynamic coalition formation. It does not only depend on a similarity heuristic but also QoS attributes evaluated with a multi-attribute utility function (see Section 7.5.1).

The nature of the on-demand bottom-up approach in open environments does not allow for a central entity to have complete control over the service composition process. The centralised planner (service consumer agent), however, still has some influence on the variable binding (binding of concrete services) because it decides on the winning coalition. If no proper coalition has been proposed, the central planner can restart the entire service composition process by changing the current candidate plan according to available alternative decompositions.

It would be interesting to move the planning procedure into service provider agents which then autonomously perform automated planning triggered by a service request and based on incoming service advertisements. This way a fully ad hoc planning and composition process may be established. This scenario is out of the scope of this thesis but envisaged as future work.

As mentioned above, the recursive planning procedure specifies the planning elements (e.g. tasks, methods, operations) in terms of the particular problem domain of on-demand service composition. Hence all these elements are represented by the notion of atomic and composite processes (see Section 6.3) including inputs, outputs, preconditions and effects (also denoted IOPE). The reduction of tasks with methods or operations bases on IOPE-based capability matchmaking. It is described in Section 7.4.

7.4 Capability Matchmaking

The term capability or semantic matchmaking was coined in the Semantic Web domain. It refers, in general, to automated reasoning about the similarity between a service request and a set of available service advertisements. Capability matchmaking is concerned
with determining the most suitable service that satisfies a service request by inferring
the similarity between a service request and a service advertisement from syntactic and
semantic specifications. The underlying concept of capability matchmaking is rooted in
research in software reuse and software components. Most notably, Zaremski and Wing
[178, 179] studied the fundamental principles of matching software components based on
syntactical and semantic component specifications.

With the advent of the Semantic Web initiative, a large body of research work has
gone into investigating of various matchmaking methods and approaches in the context of
the specific characteristics and capabilities of Semantic Web technologies such as knowl-
edge representation formats and inference methods (e.g. see Daconta et al. [33]). Hence
services, and Semantic Web services in particular, are software components that publish
a richer description of their capabilities than traditional software components. The aim
of these research efforts is to facilitate Semantic Web-based automated dynamic service
discovery and composition. Klusch [91] provides a classification of matchmaking methods
and an overview of currently available matchmaker implementations.

Given the rich descriptions of services, there are various characteristics of a service
that can be used for matchmaking purposes. Sycara et al. [158] for example define a
specification language called LARKS for service advertisements that contains the following
elements:

- **Context**, the specific application or problem domain, e.g. travel planning
- **Types**, the definition of variable types
- **Input**, the input parameters of a service
- **Output**, the output parameters of a service
- **InConstraints**, the preconditions of a service
- **OutConstraints**, the effects of a service
- **ConcDescriptions**, ontological description of a service.

Sycara et al. [158] also specify a set of different matchmaking modules or filters on top
of the LARKS specification, including:
7.4. CAPABILITY MATCHMAKING

- **Context filter**, matches the context information of two services for matching application domains

- **Profile filter**, matches the entire specifications of two services for determining similar word frequencies

- **Similarity filter**, matches the semantic distance between words of two service specifications

- **Signature filter**, matches the input and output parameters of two service specifications

- **Constraint filter**, matches the preconditions and effects of two service specifications.

The LARKS specification provides a good way to illustrate commonly adopted matchmaking filters. The LARKS specification of a service also has many similarities to the OWL-S process model (Martin et al. [114]) which forms the basis of the data structures used in this thesis (see Section 6.3). Due to this close relationship, LARKS has influenced the definition of service advertisements in this thesis (see Section 4.2.3) and the respective matchmaking activities.

Matchmaking does not only determine whether a match between a service request and a service advertisement exists. It also determines the degree of matching. At the lowest level, matchmaking compares atomic properties such as i) class or type of input and output parameters or ii) logical propositions and clauses of preconditions and effects. Figure 7.6 illustrates the five matching classes that can be distinguished with respect to matching a single property A to a single property B.

An *exact match* is the most accurate matching class. It implies that the two compared properties are equivalent. The next two classes represent implications. Property A *generalises* property B, if property B implies property A. In other words, B forms a subset of A. Similarly, property A *specialises* property B if property A implies property B. These two matching classes still provide a high but non-universal degree of similarity. Property A *intersects* with property B if they share commonalities but also exhibit significant differences. Intersection represents a partial matching and a low degree of similarity.
Finally, Property A does not match property B if both properties have no similarities and the intersection between the two properties is the empty set.

The matching of two components, e.g., a service request and a service advertisement, can be derived from the degree of similarity in matching single properties. A set of matching classes is documented in the literature, but with inconsistent interpretations. (The reader is referred to Klusch et al. [95], Li and Horrocks [105] or Paolucci et al. [131] for representative examples.) In this thesis, only two matching classes are of interest: exact matching and plug-in matching. They provide the highest degree of similarity and allow for an effective application during dynamic service composition. Furthermore, both matching classes can be detected with the context, signature and constraint filters. The following two definitions introduce the exact matching and plug-in matching classes based on the example of a signature filter. The two definitions embody simplified and slightly adapted versions of their specification in Klusch et al. [95].

The two definitions are presented with the following terminology. ∀ is the universal quantifier. It refers to all elements of a set. ∃ is the existential quantifier. It refers to one element in a given set. ⇔ represents logical equivalence. ∧ symbolises logical conjunction. \( \text{input}_A \) (\( \text{input}_R \)) denotes an input parameter of a service advertisement (service request). Likewise, \( \text{output}_A \) (\( \text{output}_R \)) denotes an output parameter of a service advertisement (service request).

A service advertisement \( A \) exactly matches a service request \( R \) if the service advertisement’s input/output signature perfectly matches the service request’s input/output
7.4. CAPABILITY MATCHMAKING

signature (Equation 7.1). The service advertisement may have less input parameters and may provide more output parameters than the service request. However, the matched parameters are pair-wise equivalent.

\[ \forall \text{input}_A \exists \text{input}_R : \text{input}_A \Leftrightarrow \text{input}_R \land \forall \text{output}_R \exists \text{output}_A : \text{output}_R \Leftrightarrow \text{output}_A \]  

A service advertisement \( A \) **plug-in matches** a service request \( R \) if the service advertisement may require equal or less input than the service request and also may provide equal or more output than the request. This way, the service advertisement should in principle be able to produce the desired output of the service request under the condition that the service advertisement input (output) parameter types generalise (specialise) the input (output) parameter types of the service request (Equation 7.2).

\[ \forall \text{input}_A \exists \text{input}_R : \text{input}_A \supseteq \text{input}_R \land \forall \text{output}_R \exists \text{output}_A : \text{output}_R \supseteq \text{output}_A \]  

Figure 7.7 is a simple illustration of plug-in matching between a service advertisement and a service request. The service request has a single input parameter of integer type \( \text{long} \) and a single output parameter of integer type \( \text{short} \). The service advertisement plug-in matches the service request because its input parameter generalises the short type of the service request. Vice versa, the service advertisement provides a specialised long type. Thus the service advertisement can be plugged-in to replace the service request.

![Figure 7.7: Simple Illustration of plug-in matching between a service advertisement and a service request.](image)

It needs to be stressed that capability matchmaking based on logical inference, as outlined so far, is restricted. This is because the logical implication between two constraints is in general not decidable in first-order predicate logic (e.g. denoted in Sycara et al. [158]). Thus for matching of preconditions and effects with a constraint filter something
else must be used. Sycara et al. [158] suggest the weaker but decidable subsumption relation introduced by Muggleton and Raedt [123]. Similarly, Zaremski and Wing [179] discuss the similarity of plug-in matchings to class–type subsumptions in object-oriented programming languages (which yields a means for implementing matchmakers). Klusch et al. [95] assert that Semantic Web technology is based on decidable description logic that enables logical inference of plug-in matchings, e.g. based on the OWL-S notation. In contrast, Klusch [91] lists a number of non-logic-based matchmakers that avoid the decidability problem all together.

A matchmaker is typically embedded in a service discovery framework in the Semantic Web-services domain. Such a framework usually comprises three roles: a service provider, a service requester and a service broker or matchmaker. The service provider publishes service advertisements, typically with the service broker. The broker establishes a third party that provides dynamic service discovery and matchmaking capabilities to the service requester. Hence capability matchmaking is solely performed in a central point.

In contrast, on-demand service composition in open environments does not define third parties for matchmaking. Capability matchmaking is performed by all software agents in the multiagent system presented in this thesis (see Section 6.1). Service consumer agents perform capability matchmaking during the automated planning process in order to determine matching methods and operations that help to refine a client-defined service request into a composite service request in the form of a valid plan. The automated planning incorporates capability matchmaking because it is directly performed on the OWL-S-derived data structures (see Table 7.1). Service provider agents conduct matchmaking to match their own service advertisements with incoming composite service requests and to match incoming service advertisements from other service provider agents (see Section 5).

The definition and matchmaking of service advertisements in this thesis is inspired by the LARKS approach. Moreover, due its dependence on the OWL-S process model, the implementation of matchmaking capability in this thesis is based on the matchmaking algorithms proposed in Klusch et al. [95] and Paolucci et al. [131]. There is, however, one key difference between the approach in this thesis and these other approaches. This thesis incorporates QoS attributes in the specification of service requests as well as service advertisements. QoS attributes are modelled similarly to input and output parameters.
They comprise key-value pairs. Hence QoS attributes can be matched using an adapted signature filter.

Even though QoS attributes are not considered by Klusch et al. [95] and Paolucci et al. [131], this thesis is not aimed at contributing to the theory of capability matchmaking. QoS-driven matchmaking has been investigated, e.g. in Vu et al. [165]. This thesis applies the general concepts of capability matchmaking tailored to the specific requirements of on-demand service composition in open environments (QoS, OWL-S-based data structures). This thesis does not develop a generic all-purpose matchmaker. The extension to include QoS attributes leverages the market-based control mechanism of the open market concept (see Section 4.3.1). On the other hand, the approach presented is restricted to exact and plug-in matches only. It provides only an exemplifying implementation and may be replaced with a more complex matchmaker. However, this is out of the scope of this thesis.

### 7.4.1 Matching Algorithm

This section outlines a matchmaking algorithm with pseudocode listings. It is based on the work in Klusch et al. [95] and Paolucci et al. [131] and exhibits strong similarities with those two approaches. Only two specific component-level matching type classes are considered. They are encoded in the implementation with numerical values. In the pseudocode listings they are represented by the following constants: equal match (*equal-matching*), plug-in match (*plug-in-matching*) and no match (*no-matching*).

Listing 7.26 outlines the main procedure of the matchmaking algorithm. The input of the procedure is an original service specification (*service*) and another service specification (*match*) that is to be matched against the former. Both parameters are provided with simple process data structures as specified in Section 6.3. In the case of the service consumer agent, the former is a service specification of a service request and the latter may be a method or operation (see Section 7.3.1). In the case of the service provider agent, the former is a service specification as part of a composite service request and the latter is a service advertisement.

The main procedure controls the execution of four matchmaking filters. The order of the filters is chosen so as to reduce the work of the matchmaking algorithm (similar
to Zaremski and Wing [179]). Only if the overall contexts (e.g. application domains) of both specifications match, and the latter specification provides an appropriate set of QoS attributes, is the signature filter executed. If the signatures of both specifications match, then the original service specification can be replaced on a syntactic level (e.g. by a concrete service advertisement for a generic service type). In this case the constraint filter must be executed in order to verify the semantic similarity between both specifications and to determine the final matching type. The main procedure returns the final matching type with the value of variable score.

Listing 7.26: Pseudocode listing of the main matchmaking procedure.

```
BEGIN MATCHMAKER( service, match )
  score: no-matching
  IF CONTEXT-FILTER( service, match ) successful AND QOS-FILTER( service, match ) successful THEN
    CALL SIGNATURE-FILTER( service, match, score )
    IF score > no-matching THEN
      CALL CONSTRAINT-FILTER( service, match, score )
    ENDIF
  ENDIF
RETURN score
END MATCHMAKER
```

The context filter shown in Listing 7.27 is simple. It basically tests whether the latter specification belongs to the same or a subdomain of the former specification’s domain (travel domain and intercity travel domain, for example).

Listing 7.27: Pseudocode listing of the context filter.

```
BEGIN CONTEXT-FILTER( service, match )
  // match type descriptions of the service and match specifications
  IF service.type equals match.type OR service.type subsumes match.type THEN
    RETURN TRUE
  ELSE
    RETURN FALSE
  ENDIF
END CONTEXT-FILTER
```

The QoS filter is depicted in Listing 7.28. It determines for every QoS parameter of the former specification a matching (equal or subsumed) QoS parameter of the latter specification. Only if there is such a match for every QoS parameter does the filter
confirm a successful matching. This is a fairly strict procedure. But the market-based control mechanism of the open market concept (see Section 4.3) requires service provider agents, for example, to comply with the client-defined QoS preferences in order to be competitive.

Listing 7.28: Pseudocode listing of the QoS filter.

```plaintext
BEGIN QOS-FILTER( service, match )
    // match qos attributes of the service and match specifications
    FOR all service.qos DO
        matching: FALSE
        FOR all match.qos DO
            IF service.qos equals match.qos OR service.qos subsumes match.qos THEN
                matching: TRUE
                BREAK
            ENDIF
        ENDFOR
        IF matching = FALSE THEN RETURN FALSE ENDIF
    ENDFOR
    RETURN TRUE
END QOS-FILTER
```

The signature filter is implemented according to the definition of exact matches and plug-in matches given in equations 7.1 and 7.2. Listing 7.29 shows the matching of input and output parameters of the two given specifications. First all input parameters of the latter service specification are checked for matching counterparts in the former service specification. If a match has been detected for every input parameter, the output parameters are matched too. Otherwise the filter returns no-matching as the matching type. Output parameters are handled similarly, but only with the order of the two specifications swapped.

Listing 7.29: Pseudocode listing of the signature filter.

```plaintext
BEGIN SIGNATURE-FILTER( service, match, score )
    //match input attributes of the service and match specifications
    FOR all match.input DO
        result: no-matching
        FOR all service.input DO
            degree= TYPE-MATCHING( match.input, service.input )
            IF degree > result THEN
                result: degree
                BREAK
            ENDIF
        ENDFOR
    ENDFOR
    RETURN result
END SIGNATURE-FILTER
```
The signature filter calls a specific matching procedure for matching the types of two concrete input parameters or two concrete output parameters. If both types are equal, the type exact-matching is returned. If the first type subsumes (is a superclass of) the second type, a plug-in matching exists. In all other cases, no matching is detected. The type matching procedure is depicted in Listing 7.30.

Listing 7.30: Pseudocode listing of the type matching procedure.
Finally, the constraint filter is presented with Listing 7.31. This filter operates like the signature filter. It first checks the preconditions of both service specifications and then their effects following the definitions given in equations 7.1 and 7.2.

**Listing 7.31:** Pseudocode listing of the constraint filter.

```plaintext
BEGIN CONSTRAINT-FILTER( service, match, score )

//match preconditions of the service and match specifications
FOR all match.precondition DO
    result: no-matching
    FOR all service.precondition DO
        degree = CLAUSE-MATCHING( match.precondition, service.precondition )
        IF degree > result THEN
            result: degree
            BREAK
        ENDIF
    ENDFOR
    IF result < score THEN score: result ENDF
    IF score = no-matching THEN RETURN score ENDF
ENDFOR

//match effects of the service and match objects
FOR all service.effect DO
    result: no-matching
    FOR all match.effect DO
        degree = CLAUSE-MATCHING( service.effect, match.effect )
        IF degree > result THEN
            result: degree
            BREAK
        ENDIF
    ENDFOR
    IF result < score THEN score: result ENDF
    IF score = no-matching THEN RETURN score ENDF
ENDFOR

// some degree of matching has been detected
RETURN score
END CONSTRAINT-FILTER
```

Instead of matching type classes, the constraint filter incorporates the procedure of Listing 7.32 in order to determine the equivalence or implication between logical clauses of both service specifications.
7. SOFTWARE AGENT DESIGN

Listing 7.32: Pseudocode listing of the atom matching procedure.

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BEGIN CLAUSE-MATCHING( clause1, clause2 )</td>
</tr>
<tr>
<td>2</td>
<td>IF clause1 equals clause2 THEN</td>
</tr>
<tr>
<td>3</td>
<td>RETURN exact-matching</td>
</tr>
<tr>
<td>4</td>
<td>ELSE</td>
</tr>
<tr>
<td>5</td>
<td>IF clause1 subsumes clause2 THEN</td>
</tr>
<tr>
<td>6</td>
<td>RETURN plug-in-matching</td>
</tr>
<tr>
<td>7</td>
<td>ELSE</td>
</tr>
<tr>
<td>8</td>
<td>RETURN no-matching</td>
</tr>
<tr>
<td>9</td>
<td>ENDIF</td>
</tr>
<tr>
<td>10</td>
<td>ENDIF</td>
</tr>
<tr>
<td>11</td>
<td>END CLAUSE-MATCHING</td>
</tr>
</tbody>
</table>

7.5 Expected Utility Calculation

The main focus of a service provider agent’s decision-making is to determine and rank prospective coalition partners according to their capabilities during on-demand coalition formation (Section 5) in order to form a competitive coalition. A service provider agent needs to perform such decision-making in two different situations. Firstly, it proactively determines its next most beneficial coalition partner in order to initiate or extend its own coalition. Secondly, it reactively reasons about the prospects of joining requesting coalitions.

The decision-making is driven by market-based control of the open market concept (see 4.3). The incentive for a service provider agent is to maximise the utility of its coalition with respect to the client-defined QoS preferences of a composite service request. Such maximisation effectively means the incorporation of coalition partners that offer maximal utility. However, this maximisation is difficult to achieve because a service provider agent does not have definite knowledge to base its decisions on. Hence a service provider agent cannot be certain about the behaviour of other software agents because of the characteristics of open environments. Service provider agents experience uncertainty during decision-making for two reasons. Firstly, all service provider agents are autonomous decision-makers (with potentially conflicting and changing local preferences and knowledge). Secondly, a service provider agent does not have complete knowledge about the availability of potential partners. Thus it cannot determine the utility of potential partners for certain. Moreover, its own preferences can change in reaction to
7.5. EXPECTED UTILITY CALCULATION

messages it receives during an on-demand coalition formation process. A service provider agent constantly updates its ‘beliefs’ or, more precisely in the context of this thesis, the utility values of potential partner agents to keep track of changes in its environment.

It must be noted that this section does not contribute to decision-making theory. The following presents a novel use of an existing theory for modelling the decision-making of service provider agents under the uncertainty of the open market concept and environment (see Section 4.1). What is presented is just an illustrative implementation. It may be extended or replaced with another decision-making algorithm. However, this is out of the scope of this thesis.

7.5.1 Expected Utility of Service Provider Agents

The von Neumann/Morgenstern expected utility theory [164] delivers a sound foundation for modelling decision-making under uncertainty. Every service provider agent assesses potential partners by calculating their expected utility values according to the von Neumann/Morgenstern Theorem and based on the client-defined QoS preferences and private preferences it possesses. A separate calculation of expected utility is needed for each type of service in the composite service request that the service provider agent (hereafter the decision-maker) does not itself provide. The following describes the calculation of expected utility for all potential partners associated with one service type. However, for an informed decision, the decision-maker performs the same calculation for all service types of the composite service request.

The starting point is the set of advertisements the decision-maker has received so far from other service provider agents (Equation 7.3). Expected utility of a potential partner is calculated based on its advertisement. Therefore, \( a_i \) is used synonymously for denoting both a potential partner and its advertisement.

\[
A = \{a_1, \ldots, a_n\} \tag{7.3}
\]

Expected utility of a potential partner \( a_i \) depends on two consequences modelled with set \( C \) (Equation 7.4). Success \( \hat{c}_{a_i} \) denotes that the potential partner accepts or suggests to join a coalition. Reject \( \check{c}_{a_i} \) denotes that the potential partner rejects a request to join a coalition or its decision is simply unknown to the decision-maker. Each consequence \( c \) is represented by the product of a probability \( p(c_{a_i}) \) and a utility \( u(c_{a_i}) \). The calculation

\[
E[U(a_i)] = \sum_{c \in C} p(c_{a_i})u(c_{a_i})
\]
of probabilities and utilities is presented in the following subsections. Expected utility of a potential partner $EU(a_i)$ is the sum of the representation of both consequences (Equation 7.5) under the condition that their associated probabilities sum to 1 (Equation 7.6). Furthermore, the utility of a consequence is denoted by $u_d$ in order to emphasise that this approach also discounts the local preferences of the decision-maker, as explained later.

$$c(a_i) \in C = \begin{cases} \hat{c} - \text{success} \\ \hat{c} - \text{failure} \end{cases}$$ (7.4)

$$EU(a_i) = \sum_{c_{a_i}} p(c_{a_i}) \times u_d(c_{a_i})$$ (7.5)

$$\sum_{c_{a_i}} p(c_{a_i}) = 1$$ (7.6)

Expected utility is calculated for every potential partner in set $A$. The decision-maker repeats the calculation for each service type. As a result, it obtains a ranking of all potential partners for all service types. The decision-maker then selects the potential partner with the highest utility.

### 7.5.2 Expected Utility of Success and Failure Consequence

The calculation of the utility of the success consequence $\hat{c}$ is based on the levels of importance of QoS parameters specified with a service request and the concrete QoS parameter values given in service advertisements $a_i$. In that sense, the utility of the success consequence does not reflect the private preferences of the decision-maker. The major incentive of a decision-maker is to successfully join coalitions and to then get contracted by a client. Thus it needs to take the client’s preferences into consideration.

Therefore, the decision-maker determines which advertisement associated with one task is the optimal one. Since all advertisements provide the same functionality, optimality must be decided according to their QoS parameter values and the QoS preferences as defined with a service request. The decision-maker ranks the service advertisements by calculating a weighted multi-attribute utility value for each one. Before applying the
7.5. EXPECTED UTILITY CALCULATION

multi-attribute utility function, the decision-maker calculates single-attribute utility values of every advertisement relative to all other advertisements. Again, the utility does not reflect the private preferences of the decision-maker. Its sole aim is to preserve the order amongst service advertisements with respect to single QoS parameters ordered from the best to the worst.

Single-attribute value functions for linear monotonic increasing and decreasing QoS domains can be used to rank advertisements in many simple monotonic QoS domains, e.g. costs, bandwidth, uplink, downlink, response time, etc. Additional functions can be found for non-linear monotonic and non-monotonic value functions. However, that is out of the scope of this thesis.

The aim of a single-attribute value function is to normalise QoS attribute values by mapping them into the interval $[0, 1]$. The starting point is the set $A$ of all advertisements (Equation 7.3). Each advertisement contains a vector $\mathbf{Q}$ of QoS parameters (Equation 7.7).

$$\forall a_i \in A: \mathbf{Q}(a_i) = \{q_1(a_i), \ldots, q_m(a_i)\}$$ (7.7)

All QoS parameters $q_j$ at position $j$ of every QoS parameter vector $\mathbf{Q}(a_i)$ refer to the same domain concept, e.g. costs. Thus they are comparable with respect to their value bindings. The notion $q_j$ is used to denote both a QoS parameter and its value binding.

The decision-maker determines the minimum $q_j^{\text{min}}$ and maximum $q_j^{\text{max}}$ parameter values for every QoS parameter $q_j(a_i)$:

$$q_j^{\text{min}} = \min\{q_j(a_i) : \forall a_i \in A\}$$ (7.8)

$$q_j^{\text{max}} = \max\{q_j(a_i) : \forall a_i \in A\}$$ (7.9)

These values are used in a linear monotonic value function to calculate utility values for each QoS parameter relative to them. Equation 7.10 applies to decreasing monotonic domains and Equation 7.11 applies to increasing monotonic domains.

$$v_j(q_j) = \frac{q_j^{\text{max}} - q_j}{q_j^{\text{max}} - q_j^{\text{min}}}$$ (7.10)

$$v_j(q_j) = \frac{q_j - q_j^{\text{min}}}{q_j^{\text{max}} - q_j^{\text{min}}}$$ (7.11)
After the application of single-attribute value functions, every advertisement \( a_i \) is associated with a vector storing single-attribute utility values \( v_j \) that correspond to their associated QoS parameters \( q_j \) (Equation 7.12). Every \( v_j \) is a value in the interval \([0,1]\) with 1 representing the highest and 0 the lowest possible utility.

\[
\forall a_i \in A: \mathbf{V}(a_i) = \{v_1(a_i), \ldots, v_m(a_i)\} \tag{7.12}
\]

A multi-attribute utility value needs to be calculated in order to establish a normalised ranking of advertisements with respect to all QoS parameters. The approach in this thesis incorporates a weighted multi-attribute utility function (Equation 7.14) that aggregates for each advertisement a weighted sum of the products of its QoS parameter values multiplied by corresponding QoS parameter weights (Equation 7.13) that are defined with the service request. The weighted multi-attribute utility function maps every advertisement onto the interval \([0, 1]\).

\[
\mathbf{W} = \{w_1, \ldots, w_m\} \tag{7.13}
\]

\[
\forall a_i \in A: u(a_i) = \sum_{j=1}^{m} w_j \times v_j(a_i) \tag{7.14}
\]

So far the decision-maker has only determined the utility of each potential partner with respect to client-defined QoS preferences. The next step is to discount this utility according to its private preferences. A function \( r \) is defined for assigning every potential partner \( a_i \) a value in the interval \([0, 1]\). It can be interpreted as a private trust and reputation function (Equation 7.15) with 1 representing highest trust and 0 no trust.

\[
r : a_i \mapsto [0, 1], \forall a_i \in A \tag{7.15}
\]

The final utility of the success consequence for potential partner \( a_i \) is computed as the product of corresponding utility and reputation values (Equation 7.16).

\[
\forall a_i \in A: u_d(\hat{c}_{a_i}) = (u(a_i) + 1) \times r(a_i) \tag{7.16}
\]

The adding of summand 1 is required to allow the decision-maker to distinguish between potential partners with minimal multi-attribute utility values (e.g. 0.0, see Equation 7.14) and potential partners with minimal reputation score (e.g. 0.0, see Equation 7.15). The key difference between these two factors is that potential partners with low multi-
attribute utility are still eligible whereas potential partners with a reputation score of 0.0 are not considered as potential collaboration candidates at all.

The utility of the failure consequence expresses what a decision-maker can gain from a potential partner that has rejected the offer to collaborate or whose decision is unknown. Since a decision-maker cannot gain anything from such a potential partner, the failure consequence does not provide any utility. Thus it is set to the value of 0.0.

### 7.5.3 Probability of Success and Failure Consequences

The probability of the success consequence of a potential partner $p(\hat{c}_{a_i})$ is calculated according to number and outcome of previous interactions with the potential partner during the current coalition formation process. The calculation is based on Bayesian systems, which are often used in trust and reputation systems [87]. Bayesian systems use a beta probability density function to calculate a current reputation score with respect to previous ratings (Equation 7.17):

$$a_{posteriori} = \frac{new\ score \otimes a\ priori}{normalising\ constant} \quad (7.17)$$

The inputs of a Bayesian system are binary ratings. They are in the case of the approach of this thesis the outcome of previous interactions between the decision-maker and a potential partner: accept or reject. Accordingly, the probability density function is not interpreted in terms of trust and reputation but as an indicator of the probability that a potential partner accepts a request to join a coalition in the future.

The advantage of using a Bayesian system is that it provides a sound basis for computing probability distributions over the two options starting from a uniform distribution. Equal distribution illustrates the decision-maker’s lack of explicit knowledge about a potential partner’s behaviour at the beginning of a coalition formation process. This knowledge is updated during the coalition formation process based on the encounters between decision-maker and potential partner.

The expected probability of the success consequence of a potential partner $\hat{c}_{a_i}$ is calculated according to the simplified Equation 7.18. The variables $success(a_i)$ and $failure(a_i)$ act as counters representing all observed encounters of either positive or negative outcomes with partner agent $a_i$ so far. Probability $p(\hat{c}_{a_i})$ is updated after every interaction with potential partner $a_i$ that either increases $success(a_i)$ or $failure(a_i)$ by 1.
The probability of the failure consequence \( p(\hat{c}_{a_i}) \) of a potential partner \( a_i \) can be calculated according to Equation 7.19 (which is based on Equation 7.6).

\[
p(\hat{c}_{a_i}) = 1 - p(\hat{c}_{a_i})
\]  

(7.19)

The decision-maker is now able to predict probabilities for both consequences. However, since the environment is dynamic and the coalition formation process takes time, potential partners may change their local preferences and thus their behaviours. In other words, information about previous interactions between decision-maker and potential partners ages. The older the information gets, the more unlikely it is that it reflects current situation. To tackle this problem, the decision-maker discounts older information during the calculation of \( p(\hat{c}_{a_i}) \).

The concept of forgetting as described in Jøsang [86] is applied in order to discount ageing information. The prerequisite of the forgetting technique is that feedback for every previous interaction with a partner agent \( a_i \) is provided in tuples \((success(a_i)_j, failure(a_i)_j)\) indexed by an increasing counter \( j \) of the number of previous interactions with the particular partner agent. The current values of \( success(a_i) \) and \( failure(a_i) \) for \( n \) previous interactions with partner agent \( a_i \) without forgetting can be expressed with Equation 7.20 and Equation 7.21.

\[
success(a_i) = \sum_{j=1}^{n} success(a_i)_j
\]  

(7.20)

\[
failure(a_i) = \sum_{j=1}^{n} failure(a_i)_j
\]  

(7.21)

The basic idea now is to use a forgetting factor \( \lambda \) that is multiplied by every summand of the above two equations with a different impact for a summand at position \( 1 \leq j \leq n \) as shown in Equation 7.22 and Equation 7.23. The forgetting factor needs to be chosen from the interval \([0, 1]\). A \( \lambda = 1 \) is equal to no ageing of information because the forgetting factor is constantly 1 for every \( j \) in this case, due to the characteristics of the power function. A \( \lambda = 0 \) can be interpreted as extreme ageing because the forgetting factor equals 0 for all \( j < n \) but is 1 for \( j = n \), due to the characteristics of the power function. Thus only the last information is considered in this case.
7.5. Expected Utility Calculation

\[ \text{success}(a_i) = \sum_{j=1}^{n} \text{success}(a_i)_j \times \lambda^{n-j} \]  \hfill (7.22)

\[ \text{failure}(a_i) = \sum_{j=1}^{n} \text{failure}(a_i)_j \times \lambda^{n-j} \]  \hfill (7.23)

Jøsang [86] also describes a simplification of the calculation that makes it unnecessary to store all but the last tuple \((\text{success}(a_i)_j, \text{failure}(a_i)_j)\) based on a recursive algorithm, depicted below. Equations 7.24 and 7.25 are then substituted into Equation 7.18 in order to calculate the probability \(p(\hat{c}_a)\) of the success consequence with forgetting factor \(\lambda\). To do so, \(\text{success}(a_i)_j \in [0, 1]\) and \(\text{failure}(a_i)_j \in [0, 1]\) provide simultaneously positive and negative feedback for the latest encounter with partner agent \(a_i\) given the constraint that \(\text{failure}(a_i)_j = \neg \text{success}(a_i)_j\) in \([0, 1]\).

\[ \text{success}(a_i) = \text{success}(a_i)_{j-1} \times \lambda + \text{success}(a_i)_j \]  \hfill (7.24)

\[ \text{failure}(a_i) = \text{failure}(a_i)_{j-1} \times \lambda + \text{failure}(a_i)_j \]  \hfill (7.25)

7.5.4 Expected Utility of Coalitions

The calculation of the expected utility of coalitions is based on the expected utility of service advertisements. The decision-maker needs to decide whether to remain in its current coalition \(CC\) (Equation 7.26) or to join the requesting coalition \(RC\) (Equation 7.27). Each coalition is defined by a set of service advertisements of their associated members. \(CC\) may be the empty set where the decision-maker acts in the candidate role.

\[ CC = \{a_1, \ldots, a_n\}, n \geq 0 \]  \hfill (7.26)

\[ RC = \{a_1, \ldots, a_m\}, m > 0 \]  \hfill (7.27)

The decision-maker aggregates expected utilities of the members of both coalitions in order to determine expected utilities of the coalitions (Equations 7.28 and 7.29), except for the case in which the cardinality of \(CC\) is \(n = 0\). In this case \(EU(CC) = 0\).

\[ EU(CC) = \frac{\sum_{i=1}^{n} EU(a_i)}{n} \]  \hfill (7.28)

\[ EU(RC) = \frac{\sum_{j=1}^{m} EU(a_j)}{m} \]  \hfill (7.29)

259
The decision-maker compares both expected utilities and will decide to leave its current coalition if $EU(RC) > 0 \land EU(RC) > EU(CC)$. Given the hard real-time constraints on the decision-maker (implied by the service request timeout), it also decides to leave if the current coalition has less partners than the requesting coalition. This way, the decision-maker increases its chance to be part of a complete coalition on time.

What has been just presented does not contribute to decision-making theory. It is simply illustrative of one approach to decision-making. It may be extended or replaced with a different decision-making algorithm. However, this is out of the scope of this thesis.

### 7.6 Basic Decision-making Capabilities

The following two decision-making capabilities are included in this thesis for the sake of completeness. They are simple straight-forward measures for deciding whether a coalition is complete and for ranking and selecting a winning coalition.

#### 7.6.1 Coalition Complete Decision

The decision whether a coalition is complete is performed by service provider agents in the leader role. The aim is to decide whether a coalition can attempt to place a composite service proposal with a requesting service consumer agent. The coalition complete decision depends on two things: the composite service request and the number and type of members in the leader’s coalition. Only if there is a coalition member for every atomic process (service) in the composite service request plan is a coalition complete. In all other cases it is not complete. The implementation of this decision-making capability performs a simple query operation on the information in the leader agent’s knowledgebase.

#### 7.6.2 Proposal Ranking and Selection

The ranking of composite service proposals and the selection of a winning coalition is performed by a service consumer agent in the contractor role. After the timeout of the composite service request has elapsed, the service consumer agent determines whether it has received a sufficient number of composite service proposals. If such a number of composite service proposal exists in its knowledgebase, the service consumer agent
calculates the utility of every proposal (or more precisely of the coalitions that provided
the proposals). The calculation is very similar to the calculation of the expected utility
of single service provider agents presented in Section 7.5.

The service consumer agent performs the following operations for every single coalition.
It aggregates the concrete values of the QoS attributes of all coalition members separately
for every single QoS attribute. Based on this aggregation of the QoS values of the coalition
members, a coalition can now be interpreted as a single software agent (for the purpose
of the decision-making algorithm only).

The service provider agent then proceeds as outlined in Section 7.5.2, with the differ-
ence being that the term coalition now replaces the term software agent. This way, the
service consumer agent normalises the different QoS attributes and assigns a single utility
value to every coalition. The different coalitions become comparable this way. The service
consumer ranks the different coalitions according to their utility values and selects as the
winner the coalition with the greatest utility value. This is the coalition whose compos-
ite service proposal best satisfies the client-defined preferences of the original composite
service request.

7.7 Summary

This chapter presents the design of two agent types that implement the respective parts
of the approach to on-demand service composition in open environments presented in this
thesis. The chapter introduces the agent internal structure and behaviour that is modelled
with a finite state machine. Furthermore, the chapter presents key decision-making and
reasoning capabilities. The two agent types of this chapter represent illustrative imple-
mentations of the static and dynamic design of the multiagent system outlined in Chapter
6. The agent internal design may be altered or augmented. Hence this thesis promotes
heterogeneous software agents that form meaningful structures dynamically by communi-
cating with each other. The two agent types have been implemented in the course of this
thesis in order to populate a multiagent system for conducting an experimental evalua-
tion of the on-demand service composition concept (Chapter 9). Moreover, the multiagent
system design discussed in the previous chapter and the agent-internal design discussed
in this chapter are used in a demo application presented in the next chapter.
Part III

Assessment
Chapter 8

Demo Application

This chapter presents a demo application that illustrates the concept of on-demand service composition in open environments in the problem domain of travel planning. The chapter starts with the outline of a concrete use case and a description of the demo application. Three scenarios of the use case are then considered to highlight the impact of client-defined preferences on the on-demand service composition process. Finally, a summary compares the three scenarios. Some of the material in this chapter was introduced in Section 1.4, where the problem of travel planning was highlighted with a use case for organising a return trip from Melbourne to Sydney. The demo application implements this use case based on the design outlined in the Chapters 6 and 7. However, it hides the complexity and the processing of the approach of this thesis. A detailed evaluation of the concept of on-demand service composition in open environments is provided in Chapter 9.

8.1 Scope

The problem of travel planning is concerned with organising a travel itinerary for one or more persons under various constraints (e.g. connections between different means of transport) and preferences (e.g. budget, timing, quality standards, dietary needs, etc.). The goal is to find an itinerary, including transportation and accommodation, that best matches the preferences of the travellers. This, however, is problematic in realistic scenarios because there are typically alternative itineraries, with numerous transportation and accommodation providers (including third parties such as travel agencies), various special offers, etc. A manual approach to the problem is very time-consuming and prone
8. DEMO APPLICATION

to fail to deliver optimal outcomes, due to the limited ability to exhaustively search the problem space in an acceptable time.

Service composition can be understood as a means to automate travel planning. The aim of the automation is to reduce the time spent on the task at hand and to increase the optimality of the outcome. A service composition approach to travel planning models the resulting itinerary as a composite service in which several services of available transportation and accommodation providers are combined. Service composition automates the search for prospective candidate services and the composition of the final travel itinerary out of them.

The concrete travel-planning use case outlined in Section 1.4 is considered for visualising the approach of this thesis. The task is to compose an itinerary for a return trip from Melbourne to Sydney that includes a one–night stay at the travel destination. The itinerary can be decomposed into three activities: outbound travel, accommodation, inbound travel. Each of these activities can be understood and modelled as a subtask that is provided by a single (or even multiple) services (e.g. an airfare, a hotel booking). Hence each activity may itself be subject to further refinement. One possible refinement of the outbound activity for example could be the following sequence: local transport from Melbourne city to airport, intercity transport from airport to airport and local transport from Sydney airport to city (see Figure 1.2 in Section 1.4). The demo application implements the use case of composing a plan for this travel.

The aim of the demo application is to demonstrate how the approach of this thesis works in principle and to sketch how it can be applied in practice. For the sake of clarity, the demo application focuses on a basic scenario with a small set of travel services and only two QoS parameters: duration and costs. The chosen QoS parameters, however, should be understood as examples only. The demo application can be extended to accommodate further QoS parameters such as dates and times, personal details, billing and payment details, etc.

8.2 Overview

The architecture of the demo application is illustrated in Figure 8.1. It is based on the generic architecture of the approach of this thesis outlined in Section 6.1. The demo
application contains a multiagent system comprising a set of service provider agents and one service consumer agent and provides a graphical user interface (GUI) to a human service–consumer.

Every service provider agent is associated with exactly one service, either transportation or accommodation. The composite service is provided by the service provider agents following the approach outlined in Chapters 4 and 5. The service provider agents are implemented according to the specifications in Chapters 6 and 7. The service provider agents and their associated services are created for the demo application using scripts.

The demo application contains a total of 100 service provider agents:

- 32 provide local transportation services in Melbourne (e.g. taxi, bus, tram and train services)
- 32 provide local transportation services for Sydney (e.g. taxi, bus, tram and train services)
- 22 provide intercity transportation services (e.g. various airfare services)
- 14 provide accommodation services in Sydney (e.g. budget, mid-range and top-end services).

Figure 8.1: Architecture of the multiagent system of the demo application.
Table 8.1 lists the properties of one such service: an airfare from Melbourne airport to Sydney airport. The solitary *input* and *output* parameters are of type *location*. They determine the departure and arrival locations of the service. The service has two *preconditions*. Firstly, the passenger must be at Melbourne’s airport in the suburb of Tullamarine in order to be able to board an aeroplane. Secondly, since this is an intercity airfare service, the input and output locations must differ. The *effect* of the service is that the passenger will be at Sydney’s airport in the suburb of Mascot after using the service (or the associated airfare to be more precise). Preconditions and effects are used to align different services into an itinerary that eventually satisfies a service request.

**Table 8.1:** Concrete service advertisement of an intercity transportation service that offers airfares between Melbourne and Sydney.

<table>
<thead>
<tr>
<th>Service advertisement</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Provider</strong></td>
</tr>
<tr>
<td><strong>Key</strong></td>
</tr>
<tr>
<td><strong>Input</strong></td>
</tr>
<tr>
<td><strong>Output</strong></td>
</tr>
<tr>
<td><strong>Precondition</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Effect</strong></td>
</tr>
<tr>
<td><strong>QoS</strong></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Services are also associated with *QoS* parameters. For the sake of simplicity, the demo application supports only two QoS parameters. Parameter *costs* defines the costs of a service. It is denoted in Australian dollars (AUD). Parameter *duration* specifies the travel time associated with the service. It is expressed in minutes (min). All other services within the demo application are specified similarly.

The service consumer agent of the demo application operates as outlined in Chapters 4 and 5. It is implemented according to the specifications in Chapters 6 and 7. The service consumer agent constitutes the interface between a user of the demo application and the multiagent system. The user interacts with the service consumer agent in order
to lodge service requests for travel planning and to view the composite service proposals that represent concrete travel itineraries. Beyond this, the demo application hides the actual functioning and complexity of the on-demand service composition process, as well as the characteristics of open environments, from the user.

8.3 Graphical User Interface

The user interface of the demo application is provided in the form of a graphical user interface (GUI) as depicted in Figure 8.2. It comprises three horizontal sections.

**Figure 8.2:** Overview of the graphical user interface of the demo application.

The *top section* provides an input mask for creating transportation and accommodation tasks and for combining these tasks into a service request. A transportation task is specified with an *input* and an *output* parameter that together define the departure (from) and arrival (to) locations of a task. Further, the user can choose to specify weights for the two QoS parameters with the weight of each parameter lying in the interval \([0, 1]\) and the sum of both equal to 1.0. The weight of a QoS parameter indicates its level of impor-
tance (or utility), with higher weights indicating more importance or higher priority. The user expresses their preferences with weights. This way, the on-demand service composition process can be directly influenced to guide service provider agents to propose travel itineraries that are optimised with respect to the user’s preferences. Optimisation with respect to the QoS parameters costs and duration actually means minimisation (from the user perspective). Of course, there are other QoS parameters where optimisation means maximisation (from the user perspective), such as available discounts or frequent flyer points. Since the minimisation and maximisation of QoS parameters follow the same principle of utility calculation (see Section 7.5.1), this chapter discusses only the former for the sake of clarity.

An accommodation task is specified with one of four pre-defined quality categories: no preferences, budget, mid-range and top-end accommodation. Each attribute value represents a particular price span. As there is no time spent travelling when staying in accommodation, there is no QoS parameter for duration for the user to alter. Furthermore, the demo application is based on the assumption that the user chooses a transportation task first before selecting an accommodation task. This way, the location of the accommodation task (which equals the values of the task’s input and output parameters) is always defined and cannot be input manually. For this reason, the accommodation input mask does not provide a means for defining locations. For both tasks transportation and accommodation, there is an add button at the right of the GUI. The user adds a task to the itinerary of a service request by clicking this button.

The middle section of the GUI displays a sequential list of the tasks added to the itinerary during service request creation. It expands into the top section after the on-demand service composition process is finished and then displays all composite service proposals that the service consumer agent has received that match the requested travel itinerary.

During service request creation, the bottom section of the GUI allows the user to specify a registration delay for the service request. The timeout is specified in milliseconds. It also has a submit button for triggering the on-demand service composition process. The bottom section also displays additional information about composite service proposals after the on-demand service composition process is finished.
8.4 Use Case Scenarios

The aim of the demo application is twofold. Firstly, it is designed to illustrate the on-demand service composition approach of this thesis with a practical application. Secondly, it is designed to highlight the effect of the user-defined preferences that make up the market-based control of the open market metaphor used in Chapter 4.

Table 8.2: Illustration of the use case of a return trip from Melbourne to Sydney including a one-night stay.

<table>
<thead>
<tr>
<th>Task 1: outbound trip</th>
<th>Task 2: stay over</th>
<th>Task 3: inbound trip</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>Key</td>
<td>Key</td>
</tr>
<tr>
<td>Transportation</td>
<td>Accommodation</td>
<td>Transportation</td>
</tr>
<tr>
<td>Input</td>
<td>Output</td>
<td>Input</td>
</tr>
<tr>
<td>{from, CBD (Melbourne)}</td>
<td>{in, CBD (Sydney)}</td>
<td>{from, CBD (Sydney)}</td>
</tr>
<tr>
<td>{quality, no preferences}</td>
<td>{in, CBD (Sydney)}</td>
<td>{to, CBD (Melbourne)}</td>
</tr>
<tr>
<td>Ordering of travel tasks: sequential</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Service request timeout: 5000 ms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of service provider agents: 100</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The basic use case that the demo application implements is shown in Table 8.2. Based on this generic use case, this section illustrates three scenarios that merely differ in the specification of the user-defined QoS preferences presented in Table 8.3.

Table 8.3: QoS preferences for three scenarios of the use case of a return trip from Melbourne to Sydney including a one-night stay.

<table>
<thead>
<tr>
<th></th>
<th>Scenario 1</th>
<th>Scenario 2</th>
<th>Scenario 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weight of costs</td>
<td>1.0</td>
<td>0.0</td>
<td>0.7</td>
</tr>
<tr>
<td>Weight of duration</td>
<td>0.0</td>
<td>1.0</td>
<td>0.3</td>
</tr>
</tbody>
</table>

In Scenario 1, the costs parameter has the full utility of 1.0 which means that the user is interested in the travel itinerary with the lowest total costs. The travel time of the trip is of no concern. In Scenario 2, the duration parameter has the full utility of 1.0 which can be interpreted as the user being after the travel itinerary with the fastest connection and thus the shortest total travel time. The travel costs are of no concern. In Scenario
8. DEMO APPLICATION

3, the cost parameter is 0.7 and the duration parameter is 0.3. In this scenario the user wants to find a low cost itinerary but is willing to compromise on cost to achieve moderate travel time. In other words, travel time matters but it is of less importance than travel cost.

8.4.1 Scenario 1

Figure 8.3 presents the service request of Scenario 1. The request comprises three tasks: outbound transportation, accommodation and inbound transportation. The weights for the QoS preferences have been set as defined in Table 8.3, with the full utility of 1.0 assigned to the costs parameter for both transportation tasks.

![Figure 8.3: Screen shot of the Scenario 1 service request for a return trip from Melbourne to Sydney including a one-night stay.](image)

Figure 8.4 shows the outcome of the on-demand service composition process. Four composite service proposals (tabs Proposal 1 to 4) have been registered with the service consumer agent. The middle section of the GUI presents an overview of relevant information for every task of the travel itinerary of Proposal 1, including information about the travel costs and travel time. The proposals are sorted in descending order based on the
utility they yield with respect to the client-defined QoS preferences. The most beneficial proposal (Proposal 1) is selected in Figure 8.4. Its overall score (1.0), total travel costs (229 AUD) and total travel time (6 hours) are displayed in the bottom section of the GUI.

Figure 8.4: Screen shot of the overview of the most beneficial composite service proposal of Scenario 1.

To obtain more detailed information, the user can expand the view of every single task of a proposal. Figure 8.5 displays an expanded view of the outbound travel task of Proposal 1. The outbound travel is provided by three service provider agents offering

1. Local transport from Melbourne city centre to Melbourne airport
2. Intercity airfare from Melbourne to Sydney
3. Local transport from Sydney airport to Sydney city centre.

Figure 8.6 presents an overview of the second most beneficial proposal of Scenario 1. It shows the reduced utility of Proposal 2, due to its increased costs of 286 AUD (compare the bottom section of Figures 8.4 and 8.6).
8. DEMO APPLICATION

Figure 8.5: Screen shot of the expanded view of the outbound trip of the most beneficial composite service proposal of Scenario 1.

Figure 8.6: Screen shot of the overview of the second most beneficial composite service proposal of Scenario 1.
8.4. USE CASE SCENARIOS

8.4.2 Scenario 2

Figure 8.7 presents the service request of Scenario 2. The request comprises three tasks: outbound transportation, accommodation and inbound transportation. The weights for the QoS preferences have been set as defined in Table 8.3, with the full utility of 1.0 assigned to the duration parameter for both transportation tasks.

Figure 8.7: Screen shot of the Scenario 2 service request for a return trip from Melbourne to Sydney including a one-night stay.

Figure 8.8 shows the outcome of the on-demand service composition process. Four composite service proposals (tabs Proposal 1 to 4) have been registered with the service consumer agent. The middle section of the GUI presents an overview of relevant information for every task of the travel itinerary of Proposal 1, including information about the travel costs and travel time. The most beneficial proposal (Proposal 1) is selected in Figure 8.8. Its overall score (1.0), total travel costs (337 AUD) and total travel time (4:10 hours) are displayed in the bottom section of the GUI.

The user can expand the view for every single task of a proposal. Figure 8.9 displays an expanded view of the outbound travel task of Proposal 1. Given the different client-defined QoS preferences, a different set of service provider agents offers a trip for the outbound travel in comparison to the respective proposal in Scenario 1 (Figure 8.5).
8. DEMO APPLICATION

Figure 8.8: Screen shot of the overview of the most beneficial composite service proposal of Scenario 2.

Figure 8.9: Screen shot of the expanded view of the outbound trip of the most beneficial composite service proposal of Scenario 2.
8.4. USE CASE SCENARIOS

8.4.3 Scenario 3

Figure 8.10 presents the service request of Scenario 3. The request comprises three tasks: outbound transportation, accommodation and inbound transportation. The weights for the QoS preferences have been set as defined in Table 8.3: 0.7 for the costs parameter and 0.3 for the duration parameter for both transportation tasks.

![Screen shot of the Scenario 3 service request for a return trip from Melbourne to Sydney including a one-night stay.](image)

Figure 8.10: Screen shot of the Scenario 3 service request for a return trip from Melbourne to Sydney including a one-night stay.

Figure 8.11 shows the outcome of the on-demand service composition process. Again four composite service proposals (tabs Proposal 1 to 4) have been registered with the service consumer agent. The middle section of the GUI presents an overview of relevant information for every task of the travel itinerary of Proposal 1, including information about the travel costs and travel time. The most beneficial proposal (Proposal 1) is selected in Figure 8.11. Its overall score (1.0), total travel costs (250 AUD) and total travel time (4:55 hours) are displayed in the bottom section of the GUI.

Figure 8.12 displays an expanded view of the outbound travel task of Proposal 1. Given the different client-defined QoS preferences, the outbound travel is again provided by a different set of service provider agents in comparison to the most beneficial proposals of Scenario 1 (compare Figure 8.5) and of Scenario 2 (see Figure 8.9).
8. DEMO APPLICATION

**Figure 8.11:** Screen shot of the overview of the most beneficial composite service proposal of Scenario 3.

**Figure 8.12:** Screen shot of the expanded view of the outbound trip of the most beneficial composite service proposal of Scenario 3.
8.5 Summary

This chapter provides an illustration of the concept of on-demand service composition in open environments with a demo application in the problem domain of travel planning. It presents an overview of a demo application and displays a set of screen shots of three scenarios of a use case of planning a return trip from Melbourne to Sydney including a one-night stay. The screen shots show how a travel-seeker can use the demo application to create travel requests and compare composite service proposals that represent concrete travel itineraries. The screen shots also highlight the effect of user-defined preferences on the on-demand service composition process. All three scenarios have been executed using the same 100 service provider agents and respective transportation and accommodation services. Table 8.4 compares the most beneficial proposals of each scenario (with each composed according to differing client-defined QoS preferences). Scenario 1 results in a cost-optimised proposal, Scenario 2 in a time-optimised proposal and Scenario 3 embodies a compromise between the other two scenarios.

Table 8.4: Comparison of most beneficial proposals for the three scenarios of the use case of a return trip from Melbourne to Sydney including a one-night stay.

<table>
<thead>
<tr>
<th></th>
<th>Scenario 1</th>
<th>Scenario 2</th>
<th>Scenario 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weight of costs</td>
<td>1.0</td>
<td>0.0</td>
<td>0.7</td>
</tr>
<tr>
<td>Weight of duration</td>
<td>0.0</td>
<td>1.0</td>
<td>0.3</td>
</tr>
<tr>
<td>Total costs</td>
<td>229 AUD</td>
<td>337 AUD</td>
<td>250 AUD</td>
</tr>
<tr>
<td>Total duration</td>
<td>6:00 h</td>
<td>4:10 h</td>
<td>4:55 h</td>
</tr>
</tbody>
</table>

Given the approach of this thesis, the demo application does not illustrate its complexity and processing. A detailed evaluation of the concept of on-demand service composition in open environments is provided in the next chapter.
8. DEMO APPLICATION
Chapter 9

Experimental Evaluation

This chapter presents the experimental evaluation of the approach to on-demand coalition formation in open environments described in this thesis. It commences with a description of the experimental setup and then details the evaluation of the properties of termination and completeness, optimality, scalability and stability. An experimental evaluation is necessary because the approach presented reflects the characteristics of open environments. As such, the uncertainty and dynamics of interactions between autonomous selfinterested decision-makers imply emergent-system behaviour. This chapter demonstrates key properties of the presented approach in a quantitative manner based on empirical data.

9.1 Experimental Setup

The experimental evaluation of the approach of this thesis was based on a reference implementation of the concept of on-demand coalition formation that was implemented according to the specification of the static and dynamic design in the Chapters 6 and 7. The reference implementation was developed and hosted with the Tracy agent toolkit (Braun et al. [19], Braun and Rossak [18]). The Tracy agent toolkit is a light-weight all-purpose agent toolkit. Tracy is entirely based on the Java programming language, version J2SE 1.4. As such, Tracy runs on any operating system for which a Java runtime environment (JRE) is available. The only minimal requirement for developing software agents and for hosting them with a Tracy-based agent platform is a minimum of 128 MB of main memory (Braun and Rossak [18]). The reference implementation was hosted using
9. EXPERIMENTAL EVALUATION

Tracy in a stand-alone fashion, on a single personal computer with 1 GB of memory and a 1.6 GHz single-core processor.

The experimental evaluation was organised in experiments that followed the same experimental setup. An experiment was defined as a set of test scenarios. A test scenario was defined with the following parameters:

- The number of different service types of a composite service request
- The number of service provider agents that provide a concrete service for a particular service type
- The total number of service provider agents
- The service request timeout in milliseconds.

Every experiment was conducted with 19 test scenarios that varied the number of service provider agents between 6 and 60. The presentation of the experiments is simplified for the sake of clarity and brevity. The analysis of the presented approach is based on 5 test scenarios. This simplification is justified by the similarity of neighbouring test scenarios. Thus 5 representative test scenarios (A–E) have been selected. They are shown in Table 9.1.

Table 9.1: Experimental setup of the 5 representative test scenarios A–E.

<table>
<thead>
<tr>
<th>Test scenario</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of service types</td>
<td>4</td>
<td>8</td>
<td>12</td>
<td>16</td>
<td>20</td>
</tr>
<tr>
<td>Number of providers per service type</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Number of service provider agents</td>
<td>12</td>
<td>24</td>
<td>36</td>
<td>48</td>
<td>60</td>
</tr>
<tr>
<td>Service request timeout in msec</td>
<td>6000</td>
<td>6000</td>
<td>6000</td>
<td>6000</td>
<td>6000</td>
</tr>
</tbody>
</table>

A test scenario was instantiated with test runs. A test run encompasses the execution of exactly one composite service request including the run-time capturing of evaluation data. Exactly 1000 test runs were conducted for every test scenario.

The mechanism for capturing evaluation data was directly implemented in the software agents in a fashion that was aimed at minimising the impact on both execution time and
memory use. This was achieved by encoding evaluation data with basic data types (e.g. small, boolean, etc.) which exhibit a small memory imprint. Moreover, all evaluation data was accumulated during a test run and converted and stored in a relational database only after a test run completed or terminated. This way, slow database operations did not have an affect on the performance of the multiagent system during on-demand coalition formation in open environments.

The following evaluation data was collected:

**Finalisation of composite service requests**

Every test run was recorded with a unique identifier, the parameters of its associated test scenario, the process model of the composite service request, a total count of the number of registered coalitions, a total count of the number of exchanged messages and the timestamp of the initial composite service request message.

**Actions and decision-making of service provider agents**

Every service provider agent was recorded with a unique identifier, its associated service type and QoS attributes.

Coalition formation actions of a service provider agent were recorded to document the number of times a service provider agent:

- Initiated a coalition
- Joined a coalition
- Extended a coalition (with a new member)
- Left a coalition
- Abandoned a coalition.

The record of an action includes information about all service provider agents that were affected by the action.
9. EXPERIMENTAL EVALUATION

Message exchange

A data record was created for every single message that was sent. It contains information about the message name, performative, sender, receiver, a timestamp to indicate when the message was sent and a flag that indicates whether the message was sent on time.

Registration of coalitions

The registration of coalitions with the service consumer agent was captured. A data record was created for every registered coalition and contains information about the leader and member agents of the coalition, a timestamp and the coalition value. The timestamp determines the time when a coalition was registered with the requesting service consumer agent.

9.2 Termination and Completeness

The term termination is used with this thesis in a meaning that is different from the notion of the mathematical proof of termination in formal verification. It refers to the ability of the proposed approach to terminate after a finite period of time. That is, it finalises computation and frees resources.

Termination must be verified for single software agents because there is no guarantee in general that the approach of this thesis terminates for all participating software agents. This is due to the inherent decentralisation and non-determinism of the approach, e.g. it may be unknown how specific software agents implement message-handling and decision-making.

The presented approach terminates for the service consumer agent. The timeout of a composite service request determines the exact amount of time a service consumer agent will wait for composite service proposals (see Section 4.2.1). After the timeout has passed, the service consumer agent will take some action, e.g. to inform its associated user about the outcome of a composite service request.

The presented approach terminates for a single service provider agent if it implements a decision-making mechanism that triggers the agent to return into some idle state after the timeout of the composite service request has passed (e.g. see Section 7.2.2).
Closely related to the termination property is the question of whether the presented approach terminates with a solution. The term *completeness* is used in this thesis in the traditional meaning of algorithm verification. It refers to the approach’s ability to determine all possible solutions for an input, if any, and to report if no solutions exist. This means that the presented approach is complete if the maximum number of possible complete coalitions has been formed for a concrete composite service request. The theoretical maximum of possible coalitions is determined by the number of available service provider agents of the service type of a composite service request with the smallest number of available service provider agents (in the test scenarios A–E this number is 3).

Figure 9.1 gives the result of the analysis of the completeness property of the presented approach. The axis of abscissae is organised according to the five test scenarios A–E. The axis of ordinates is organised in per cent from 0 to 100.

![Figure 9.1: Completeness of test runs of test scenarios A–E.](image)

Given the setup of the experiment, there are always 3 complete coalitions possible per test scenario. Thus a test run of a test scenario is complete if 3 coalitions were registered with the service consumer agent. Figure 9.1 illustrates the percentage of test runs that resulted in complete outcomes. It also shows the percentage of test runs that terminated with non-complete results: 2, 1 and no registered coalitions.

As shown, the test scenario with the least service types (test scenario A) result in the
9. EXPERIMENTAL EVALUATION

most test runs with a complete outcome (93%). The proportion of test runs resulting in less-complete outcomes increases as the number of service types increases. Test scenarios comprising a large number of service types often terminate with only one registered coalition, but predominantly with no registered coalitions at all (test scenarios D and E).

The decrease in complete outcomes correlates with the increase of the number of service types of a composite service request, and thus with the increased number of service provider agents that must interact in order to form complete coalitions. These additional interactions require additional time, and this may delay the registration of coalitions beyond the timeout of a composite service request. Figure 9.2 illustrates this assertion by depicting the coalition registration times of the first, second and third coalition of the test scenarios A–E. The axis of abscissae is organised according to the five test scenarios A–E. The axis of ordinates is organised in milliseconds from 0 to 6000. Two trends are obvious when comparing. Firstly, the distribution of registration times becomes more widespread over the interval between 0 and 6000 milliseconds as the number of service types increases. Secondly, coalitions are increasingly registering later as the number of service types increases. (Compare the average coalition registration times – over 1000 test runs – in Figure 9.2.)

![Figure 9.2: Registration times of the first, second and third complete coalition of test runs of test scenarios A–E.](image)

286
9.2. TERMINATION AND COMPLETENESS

To underpin the above observation, test scenario E was performed in a modified version E′ that had an extended service request timeout of 15 000 milliseconds as shown in Table 9.2.

Table 9.2: Parameters of the modified test scenario E′.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Test scenario E′</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of service types</td>
<td>20</td>
</tr>
<tr>
<td>Number of providers per service type</td>
<td>3</td>
</tr>
<tr>
<td>Number of service provider agents</td>
<td>60</td>
</tr>
<tr>
<td>Service request timeout in msec</td>
<td>15 000</td>
</tr>
</tbody>
</table>

The completeness of test scenario E′ is compared to the outcome of test scenario E in Figure 9.3. As shown, the extended service request timeout of test scenario E′ improves the completeness in comparison to the original test scenario E. It terminated with 3 registered coalitions in 4% of test runs, 2 registered coalitions in 47% of test runs and 1 registered coalition in 43% of test runs. Accordingly, the probability increases that also test scenarios with a large number of service types and service provider agents terminate with a result (one or more coalitions) when given more time.

Figure 9.3: Completeness of test runs of the test scenarios E and E′.
9. EXPERIMENTAL EVALUATION

Following this result, it can be argued that temporal influences, such as network delays and time-consuming reasoning and decision-making algorithms, can be compensated by extending the service request timeout. However, it must be noted that a service requester has no knowledge of temporal influences. Hence it is impossible to determine the optimal service request timeout, because the service requester has no means of determining the number, nature and location of available service provider agents prior to a composite service request. Moreover, the number of service provider agents might also change during execution. Hence there is no guarantee in general that the presented approach terminates with a solution. This is caused by the

- Local decision-making. The local preferences of service provider agents may prevent the successful completion of coalitions.

- Inherent decentralisation and non-determinism.

Section 10.3 discusses why a composite service request may fail to produce a result.

9.3 Optimality

The term *optimality* in this thesis refers to the assessment of the coalition values of registered coalitions from the point of view of the service consumer agent. As described in Section 4.2.4, service provider agents form coalitions guided by the market-based control of client-defined QoS requirements. In this respect, the specific QoS attributes of the services that make up a composite service proposal can be understood as the coalition value of the proposing coalition of service provider agents. To be successful, service provider agents aim at forming coalitions with coalition values that maximise the utility of a composite service proposal for the requesting service consumer agent. In other words, service provider agents aim at forming coalitions that fulfil best the QoS requirements of a composite service request.

Thus the evaluation of the optimality property of the presented approach investigates how much utility coalitions provide. An empirical evaluation of this property is necessary because the presented approach does not guarantee optimal outcomes due to the inherent uncertainty that is caused by:
9.3. OPTIMALITY

- Local decision-making. The local preferences of service provider agents may prevent the successful completion of coalitions.

- Inherent decentralisation and non-determinism.

Section 10.3 discusses why a composite service request may fail to produce an optimal result.

An optimal coalition yields a utility of 1.0. The optimality of a coalition is evaluated based on its coalition value in relation to the minimal (utility 0.0) and the maximal (utility 1.0) coalition values that are theoretically possible based on the set of available service advertisements. The calculation of the utility of a coalition value is performed according to the multi-attribute utility function defined in Section 7.5. It is in the interval [0,1].

Figure 9.4 is a plot of the utilities of all registered coalitions of test scenarios A–E. The axis of abscissae is organised according to the five test scenarios A–E. The axis of ordinates represents the interval [0,1]. Figure 9.4 separates the utility values of the first, second and third registered coalitions in lanes for every test scenario. It also shows the minimum, maximum and average utility for each lane.

Figure 9.4: Utility of the first, second and third registered coalitions of test runs of test scenarios A–E.

First of all, the small number of dots in the figure surprises. It is the result of the finite number of combinations of the utilities of available service advertisements per service type.
Figure 9.4 shows that the utilities of registered coalitions vary significantly. They are distributed across almost the entire interval $[0,1]$. Thus it is possible that a composite service request results in an outcome with low utility.

However, the utility of the first registered coalition of a test run is, on average, above 0.62 in all test scenarios. The average utilities of the second and third registered coalition are significantly smaller. The interpretation of this observation is that with the presented approach, service provider agents that offer high utility with their service advertisements form coalitions faster than others. The second and third registered coalitions are predominantly formed by service provider agents with less beneficial service advertisements.

Figure 9.4 also shows differences between the test scenarios. Test scenarios with a high percentage of complete outcomes (e.g. test scenario A) typically provide on average higher utility than test scenarios with a high percentage of incomplete outcomes (e.g. test scenario D). This difference is purely stochastic. The more complete coalitions are registered with the service consumer agent, the higher the probability that high-utility coalitions are among them. In this respect, the average utility of first registered coalitions in test scenario D is higher than their counterparts in test scenario C because their number is smaller but in a window with randomly higher utilities.

To increase the chances of high-utility outcomes, composite service requests must specify a service registration timeout that allows enough time for the formation of many complete coalitions. Figure 9.5 depicts the effective average utility of the most beneficial registered coalition of test runs for each test scenario. This is the utility value that really matters to the service consumer agent because it determines the average utility of the typical winning coalition. As shown, the utility of the winning coalition ranges up to the value of 0.87 on average.

Moreover, all test scenarios produced an average utility significantly higher than 0.5. This means that coalition formation that is guided by QoS-driven market-based control provides better outcomes than an approach based on the random creation of coalitions (assuming a normal distribution of all possible combinations of service provider agents).

However, it must be noted that the presented values are only averages. As shown in Figures 9.4 and 9.5, concrete composite service requests may result in coalitions with
9.3. OPTIMALITY

Figure 9.5: Average utility of the most beneficial registered coalition of test runs of test scenarios A–E.

low utility. This is caused by local decision-making and the inherent decentralisation and non-determinism of the presented approach.

So far, the evaluation has focused on investigating the impact of decentralisation and non-determinism. It has only considered implicit local decision-making that is the result of the probability calculations specified in Section 7.5.3.

The test scenarios F, F’ and F” (see Table 9.3) were created in order to analyse the impact of explicit local preferences (e.g. based on trust and reputation – see Section 7.5.2; particularly Equations 7.15 and 7.16).

Table 9.3: Parameters of test scenarios F, F’ and F”.

<table>
<thead>
<tr>
<th>Test scenario</th>
<th>F</th>
<th>F’</th>
<th>F”</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of service types</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Number of providers per service type</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Number of service provider agents</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Service request timeout in msec</td>
<td>3000</td>
<td>3000</td>
<td>3000</td>
</tr>
</tbody>
</table>

The following test scenarios F’ and F” differ from F in the specific local preferences
of one service provider agent. For the sake of brevity, all three test scenarios contain two service types only. Assume the service types are denoted \( a \) and \( b \). There are three service provider agents per service type:

- Agent 1: service type \( b \), value= 8
- Agent 2: service type \( a \), value= 7
- Agent 3: service type \( a \), value= 4
- Agent 4: service type \( b \), value= 9
- Agent 5: service type \( b \), value= 5
- Agent 6: service type \( a \), value= 9.

Agent 3 and Agent 5 form the most beneficial coalition if it is assumed that the aim is to form a coalition with the smallest coalition value. However, the local preferences of Agent 3 may prevent this. Table 9.4 shows some possible local preferences of Agent 3, defined according to Equation 7.15 as a value in the interval \([0,1]\).

**Table 9.4:** Comparison of the local preferences of Agent 3 in the test scenarios \( F \), \( F' \) and \( F'' \).

<table>
<thead>
<tr>
<th></th>
<th>( F )</th>
<th>( F' )</th>
<th>( F'' )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent 5</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Agent 1</td>
<td>1.0</td>
<td>0.2</td>
<td>0.0</td>
</tr>
<tr>
<td>Agent 4</td>
<td>1.0</td>
<td>0.8</td>
<td>0.0</td>
</tr>
</tbody>
</table>

The impact of the explicit local preferences of Agent 3 in test scenario \( F' \) is presented in Table 9.5. The utilities of outcomes are reduced because the local preferences of Agent 3 favour coalitions with the least beneficial service provider agent Agent 4. The maximum utility of the first, second and third registered coalition in any test run drops from 1.0 (in test scenario \( F \)) to 0.67. The average utilities of coalitions registered second and third rise slightly because both of the most beneficial agents, Agent 3 and Agent 5, do not form the first registered coalition and are, therefore, available for coalitions that are registered later.
Table 9.5: Utility of registered coalitions of test scenarios $F$ and $F'$.

<table>
<thead>
<tr>
<th>Test scenario</th>
<th>$F$</th>
<th>$F'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Utility of coalitions registered 1st</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minimum</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Maximum</td>
<td>1.0</td>
<td>0.67</td>
</tr>
<tr>
<td>Average</td>
<td>0.53</td>
<td>0.48</td>
</tr>
<tr>
<td>Utility of coalitions registered 2nd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minimum</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Maximum</td>
<td>1.0</td>
<td>0.67</td>
</tr>
<tr>
<td>Average</td>
<td>0.51</td>
<td>0.52</td>
</tr>
<tr>
<td>Utility of coalitions registered 3rd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minimum</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Maximum</td>
<td>1.0</td>
<td>0.67</td>
</tr>
<tr>
<td>Average</td>
<td>0.30</td>
<td>0.31</td>
</tr>
</tbody>
</table>

Table 9.6 presents the impact of the explicit local preferences of Agent 3 in test scenario $F''$. Since Agent 3 does not prefer to collaborate with any of the service provider agents that serve service type b, the number of registered coalitions per test run drops. Agent 3 is effectively not available for coalition formation in this test scenario. Accordingly, test runs of test scenario $F''$ do not result in optimal outcomes. The average utility of registered coalitions drops from 0.44 to 0.35.

Table 9.6: Number and optimality of outcomes of test scenarios $F$ and $F''$.

<table>
<thead>
<tr>
<th>Test scenario</th>
<th>$F$</th>
<th>$F''$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Percentage of test runs with 3 registered coalitions</td>
<td>100 %</td>
<td>0 %</td>
</tr>
<tr>
<td>Percentage of test runs with 2 registered coalitions</td>
<td>0 %</td>
<td>100 %</td>
</tr>
<tr>
<td>Percentage of outcomes with maximum utility</td>
<td>35 %</td>
<td>0 %</td>
</tr>
<tr>
<td>Average utility</td>
<td>0.44</td>
<td>0.35</td>
</tr>
</tbody>
</table>

Thus the impact of local preferences on the optimality (and even completeness) of the
presented approach is significant. However, since local preferences are private, a service consumer agent does not have the knowledge to unambiguously determine if a particular optimality of a composite service request can be achieved.

## 9.4 Scalability

The term *scalability* is used in this thesis in its traditional meaning in software engineering. It determines a system’s ability to accommodate an increasing amount of workload in a graceful manner. In the context of this thesis, scalability refers to the ability to support interactions among a growing number of service provider agents and is linked to the notion of communication costs.

The interaction protocols (see Section 6.4) influence the scalability of the presented approach most because service provider agents congregate into complete coalitions only through communication. Thus the question is how does the number of exchanged messages increase with increasing numbers of interacting service provider agents.

This thesis does not analyse the scalability of single service provider agents. The approach is intrinsically decentralised and uncertain, e.g. it may be unknown what reasoning and decision-making techniques are implemented by specific service provider agents. Although, the factors mentioned have an impact on the overall performance of the approach (e.g. computational complexity) they are not considered here.

**Table 9.7:** *Communication costs of the broadcasting component of test scenarios A–E.*

<table>
<thead>
<tr>
<th>Test scenario</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Number of service provider agents</strong></td>
<td>12</td>
<td>24</td>
<td>36</td>
<td>48</td>
<td>60</td>
</tr>
<tr>
<td><strong>Number of broadcast messages</strong></td>
<td>144</td>
<td>576</td>
<td>1296</td>
<td>2304</td>
<td>3600</td>
</tr>
</tbody>
</table>

The scalability of the approach depends on two components. The first component is the communication broadcast to disseminate a composite service request or a service advertisement. Table 9.7 shows that the increase in communication costs of the broadcasting component follows a quadratic trend that depends on the number of service provider agents. The communication costs of the broadcasting component can be estimated ac-
9.4. SCALABILITY

According to Equation 9.1 derived from the fact that:

- The service consumer agent sends \( n \) messages to distribute a composite service request among \( n \) service provider agents (see Section 6.4.1)
- \( n \) service provider agents send \( n - 1 \) messages to other service provider agents to disseminate their service advertisements (see Section 6.4.2).

\[ \text{number of broadcasting messages} \leq n + n(n - 1) = n^2 \quad (9.1) \]

The less-than or equal-to sign is in place because not all service provider agents may decide to engage in the coalition formation process for a specific composite service request, and thus some will not send a service advertisement. The broadcasting component exhibits reasonable scalability given the complexity of open environments. However, depending on the implementation of a concrete broadcasting mechanism in an overlay network (e.g. flooding), the approach presented may cause high network load, depending on the throughput and bandwidth of the underlying network technology and message size.

![Figure 9.6: Number of messages exchanged (excluding broadcasting) in test scenarios A–E.](image)

The second component of scalability is the unicast and multicast communication during the coalition formation and coalition registration processes. Figure 9.6 shows the
9. EXPERIMENTAL EVALUATION

number of exchanged messages of the test scenarios excluding broadcast messages. The axis of abscissae is organised according to the five test scenarios A–E. The axis of ordinates is organised in numbers of exchanged messages: from 0 to 8000. Figure 9.6 plots the total number of exchanged messages for all test runs per test scenario and also shows the minimum, maximum and average number of messages exchanged.

The empirical data of the test scenarios is contrasted with the theoretical minimum number of messages required to form 3 complete coalitions in every test scenario (the line with triangles at the bottom of Figure 9.6). The theoretical minimum number of messages can be calculated with Equation 9.2, where $t$ is the number of service types of a composite service request.

\[
\text{min number of messages} = 2(t - 1) + \frac{(t - 2)^2 + t - 2}{2} + 3(t - 1) + 1 \quad (9.2)
\]

The first two summands count the number of messages that are required for initiating and extending a coalition (see the interaction protocols in Sections 6.4.3 and 6.4.4 respectively). The first of the two summands covers necessary coalition-request and coalition-accept messages. The second summand covers the coalition-internal communication via coalition-inform messages. The last two summands count the number of messages that are required for completing a coalition and registering it with the requesting service consumer agent (see the interaction protocol in Section 6.4.7). Thus $3(t-1)$ coalition-complete-query, coalition-complete-inform and coalition-registration-confirm messages are exchanged, and 1 composite-service-proposal message.

Figure 9.6 reveals an approximately quadratic trend for the second component of the scalability property; the component that depends on the number of service types of a composite service request. Figure 9.6 also shows that the measured communication costs of the test scenarios always exceed the theoretical minimum, e.g. the more agents that are involved, the more the average number of messages exchanged lifts up away from the theoretical minimum. This observation can be explained by emergent-system behaviour, the uncertainty and the non-determinism of the approach:

- Service provider agents may join and leave coalitions. Every such action produces additional communication costs.
9.4. SCALABILITY

- Service provider agents may also refuse to join a coalition, with the result that further interactions are required to complete a coalition.

- Concurrency, real-time and other temporal issues (see Sections 10.2 and 10.3) may result in unsuccessful interactions, and thus trigger additional message exchanges.

The extent of the impact of these factors is unpredictable due to the non-deterministic emergent-system behaviour of the presented approach. However, the probability of their occurrence increases with an increasing number of service types and service provider agents (see Figure 9.6).

What Figure 9.6 reveals is that the presented approach does not cause excessive message exchanges and chaotic system behaviour despite the non-determinism and emergent-system behaviour. This observation can be explained by the fact that the interactions between service provider agents are constrained by the structure of composite service requests (the service types) and the market-based control (the client-defined QoS preferences) which together imply selective interactions between service provider agents rather than uncoordinated message exchanges. For example, service provider agents in role member may even stop proactively initiating interactions for extended periods of time. Moreover, the quadratic increase in communication costs with respect to the number of service types (of a composite service request) does not suggest a difficult problem since it can be expected that service requests are unlikely to exceed 15–20 service types in realistic scenarios.

The impact of an extended service request timeout on the number of exchanged messages is shown in Figure 9.7, which compares the number of exchanged messages in test scenarios E and E’. The number of exchanged messages is significantly higher in test scenario E’. It roughly doubles the number of exchanged messages in test scenario E based on its roughly two-times longer service request timeout. Following Figure 9.7, extended service request timeouts significantly impact the scalability of the presented approach. The larger number of messages exchanged is caused by the service provider agents attempting to form all possible complete coalitions. Given the potential incompleteness of the presented approach (see Section 9.2), service request timeouts should not be specified detached from software usability norms with an excessively large number.
9. EXPERIMENTAL EVALUATION

Figure 9.7: Number of messages exchanged (excluding broadcasting) in test scenarios E and E'.

9.5 Stability

The term stability in this thesis refers to the frequency with which service provider agents leave or abandon coalitions. Service provider agents do not possess any other means of reducing the number of coalition members or to completely dissolve a coalition. The less frequently service agents change coalitions, the faster coalitions are successfully completed and are ready for registration. Hence stability in this thesis can be understood as a measure of how volatile the approach of on-demand coalition formation is.

The term stability is not used according to its meaning in the area of static coalition formation, where it refers to the fair and stable division of payoffs among the software agents of complete coalitions using game-theoretical concepts (see Sandholm [147]). The original concept of stability cannot be applied to dynamic coalition formation approaches (such as this approach) because it is based on the assumption of static coalition structures (see Klusch and Gerber [93]) that cannot adapt to changes in open environments.

Figure 9.8 shows the number of times service provider agents left or abandoned a coalition before it was complete. The axis of abscissae is organised in double bars according to the five test scenarios A–E. The left bar is a plot of the number of times a member agent left an incomplete coalition in every test run. The right bar represents the number
of times a leader agent abandoned an incomplete coalition. Both bars are annotated with the maximum, minimum and average value of leave and abandon actions per test scenario. The axis of ordinates is organised in numbers of occurrences of events where a service provider agent left an incomplete coalition (ranging from 0 to 150).

Figure 9.8: Average occurrences of members leaving and leaders abandoning coalitions in test scenarios A–E.

Figure 9.8 shows that the approach of on-demand coalition formation in open environments offers good stability. No arbitrary activities or uncoordinated system behaviour is apparent. For example, there are on average only approximately 4 members that leave a coalition and only 2 leaders that abandon a coalition in test scenario A.

The number of members leaving a coalition increases as the number of service types increases and as the number of service provider agents, that join a coalition formation process, increases – following a linear trend. The increase can be explained by the higher number of small incomplete coalitions in test scenarios with larger numbers of service types. Service provider agents are more likely to change if there are many of them. The reason why service provider agents leave incomplete coalitions can be found in emergent-system behaviour, the uncertainty and the non-determinism of the approach:

- Local decision-making. The local preferences of service provider agents may be in conflict and prevent certain coalition structures (e.g. service provider agents leave
9. EXPERIMENTAL EVALUATION

- Dynamics and non-determinism. Concurrency, real-time and other temporal issues (see Sections 10.2 and 10.3) may cause unsuccessful coalitions, and thus trigger agents to leave or abandon coalitions in favour of more beneficial coalitions.

The extent of the impact of these factors is unpredictable due to the non-deterministic emergent-system behaviour of the presented approach. However, the probability of their occurrence increases with an increasing number of service types and service provider agents. This increasing probability also causes a more widespread distribution of coalition leave actions in the interval between 0 and 140 (e.g. test scenario E). Where complete coalitions cannot be formed quickly, the impact of the above factors is amplified and accordingly the probability increases that service provider agents change coalitions.

Note, the recessive trend of coalition-abandon actions in Figure 9.8. It is caused by the local decision-making of service provider agents. Coalition leaders pick potential coalition members based on their utility. Hence a coalition leader always attempts to extend its coalition with the next most beneficial candidate. Service provider agents that join a coalition as a member typically do not join the most beneficial set of service provider agents. This is because they have no direct influence on the selection of coalition members. In this context, test scenarios with large numbers of service types (e.g. E) do not allow enough time for forming coalitions of high utility that would attract leaders of other coalitions and trigger them to leave their own coalitions. Thus coalition leaders are more likely to remain in their own coalitions.

The impact of an extended service request timeout on the stability of the presented approach is illustrated in Table 9.8. Test scenario $E'$ shows a significantly higher number of leave and abandon actions subject to the service request timeout of 15 000 milliseconds. Although the numbers of test scenario $E'$ seem very high, they indicate good stability in correlation with the duration of service request timeouts. For example the maximum number of leave actions is 416 which means that, equally distributed over all 60 service provider agents, every service provider agent left coalitions on average less than 7 times.

The high stability of the approach can be explained by the fact that relationships between service provider agents are constrained by the structure of composite service
9.6. SUMMARY

Table 9.8: Comparison of the stability of test scenarios $E$ and $E'$.

<table>
<thead>
<tr>
<th></th>
<th>$E$ Leave actions</th>
<th>$E$ Abandon actions</th>
<th>$E'$ Leave actions</th>
<th>$E'$ Abandon actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum</td>
<td>16</td>
<td>2</td>
<td>136</td>
<td>45</td>
</tr>
<tr>
<td>Maximum</td>
<td>141</td>
<td>33</td>
<td>416</td>
<td>167</td>
</tr>
<tr>
<td>Average</td>
<td>83.04</td>
<td>16.07</td>
<td>266.79</td>
<td>88.29</td>
</tr>
</tbody>
</table>

requests (the service types) and the market-based control (the client-defined QoS preferences) which together imply selective decision-making and the subsequent selective creation of coalition structures.

9.6 Summary

This chapter provides an empirical evaluation of key properties of the approach to on-demand coalition formation in open environments implemented according to the specification in the Chapters 6 and 7. An empirical evaluation is required because the presented approach reflects the characteristics of open environments. The chapter analyses the termination and completeness, optimality, scalability and stability of the presented approach. The empirical evaluation shows, in general, that the presented approach terminates with adequate solutions given the complexity of the problem environment. The approach does not exhibit chaotic uncontrolled behaviour, because the interactions between service provider agents are constrained by the structure of composite service requests (the service types) and the market-based control (the client-defined QoS preferences).

The evaluation also highlights that extended service request timeouts trigger a higher probability of terminating composite service requests with complete outcomes. Extended timeouts also attract on average outcomes with a higher utility. However, extended timeouts also cause significantly higher communication costs. Even though the scalability of the approach presupposes a quadratic trend, it may pose a bottleneck in a concrete broadcasting mechanism in an overlay network. The result, depending on the throughput and bandwidth of the underlying network technology and message size, may be high network load.
9. EXPERIMENTAL EVALUATION

The evaluation stresses that the performance of the presented approach is affected by the local-decision making of service provider agents as well as its emergent non-deterministic system behaviour. The following chapter discusses a number of issues that contribute to the complexity of the presented approach.
Chapter 10

Lessons Learnt

This chapter discusses a number of issues raised during the design, implementation and operation of a multiagent system that implements the approach of this thesis (as presented in the chapters 4–7). The chapter commences with an outline of some general conceptual and design issues which is followed by an overview of the problems encountered during the design and implementation of the interaction protocols. Finally, the chapter presents a number of run-time issues.

This chapter focuses on selected key issues that have an impact on the nature and performance of the approach presented. The documented issues are directly related to the interpretation of the notion of agency proposed in this thesis (see Section 2.4). They highlight how closely intertwined the focus, complexity, benefits and shortcomings of an agent-based approach are and how this affects the agent-based software development paradigm in general. This may explain why so far only a limited number of agent-oriented solutions can be found in industry (see Hendler [68]).

10.1 General Conceptual and Design Issues

This section is concerned with general conceptual issues of the agent-oriented paradigm and their impact on particular design tasks. The section commences with a characterisation of agent-oriented design principles in order to clearly distinguish agent orientation from other software engineering paradigms. It provides the reference for the discussion of specific issues in the remainder of this section. A number of subsections then report on
difficulties with specific design tasks that have been encountered during the modelling of the multiagent system and the software agent types of this thesis. The section ends with assessing the uniformity and universality of the agent-oriented paradigm.

10.1.1 Agent-Oriented Abstraction and Decomposition

Complex software systems are typically characterised by a large number of subsystems that interact based on structural, functional or organisational relationships in order to provide functionality in an efficient and reliable fashion. Each subsystem is often a complex structure itself comprising further interacting subsystems.

The conventional approach to modelling complex software systems is applied in the form of functional decomposition. Functional decomposition is based on the design principle of the separation of concerns. It enables the effective structuring of a problem space into smaller subproblems until a level is reached on which the complexity of the subproblems allows for an adequate solution. Functional decomposition introduces structural and functional dependencies between the subsystems, each of which provides a complementary fraction of the functionality of the entire system. Even though the aim of modelling software systems is to achieve high cohesion within a single subsystem and low coupling between different subsystems, functional decomposition imposes a hierarchical structure and strict data and control flows. After all, a software system is designed to model a problem correctly and to operate in a reliably deterministic fashion.

In contrast, the agent-oriented paradigm fosters organisational decomposition (Jennings [79], Zambonelli et al. [177]). It revolves around identifying organisational roles and the interactions between them according to social relationships and dependencies. Roles are identified through and described by goals. Their interactions are guided by organisational policies and modelled on the exchange of messages that express goals and objectives rather than operation names and parameter lists. Zambonelli et al. [177] describe organisational abstraction and decomposition in detail.

In comparison, classic functional decomposition yields constraints that are not suitable for agent-based modelling of complex software systems. Firstly, the hierarchical structure imposed by functional decomposition interferes with the properties of agent autonomy and proactiveness (see Section 2.4). A multiagent system is a system of autonomous goal-
oriented problem-solvers that react to, and trigger, actions based on local decision-making. As such, the pre-defined relationships and dependencies of a hierarchy may prevent a software agent from acting autonomously because it is not free to choose (given the context of the overall system’s correctness). Therefore, Jennings [78] suggests the term ‘organisation’ for referring to the relationships between software agents: “However, hierarchy invariably gives the connotation of control, hence the more neutral term ‘organisation’ is used here. Organisations can be arranged such that they correspond to control hierarchies, however they can also correspond to groups of peers, and anything that falls in-between.”

Secondly, the agent paradigm provides a higher level of abstraction (Odell [129], Wooldridge [169]). Functional decomposition allows the recursive decomposition of a software system down to the detailed design of atomic components. In contrast, Wooldridge and Jennings [170] remark: “Generally, agents should be coarse grained, in that each should embody significant, coherent computational functionality.” Thus an agent-based approach is not meant to break down complexity into small manageable functional units. It remains at the level of system design and is focused on modelling the interactions between organisational entities (Henderson-Sellers and Gorton [67]). Wooldridge and Jennings [170] warn not to model every basic component with software agents and denotes that multiagent systems with more than 10 software agents are regarded as large.

In summary, the distinctive feature of the agent-oriented paradigm is its abstraction model of autonomous problem-solvers that accommodates a natural and effective decomposition of complex software systems based on organisational roles and their flexible relationships and dependencies (Jennings [79, 78]). Odell [129] and Jennings [78] identify the agent-oriented paradigm as a natural evolutionary successor to existing software engineering paradigms (namely object orientation) due to its higher level of abstraction. This claim emphasises that the agent paradigm remains at the level of system design, by modelling the interactions between subsystems. However, it is not suitable for a further breakdown of the complexity of the modelled subsystems. Thus it provides adequate solutions only for specific problems. In this context, Jennings and Wooldridge [82] argue that an agent-oriented approach offers advantages in particular for modelling open systems and helps to create solutions that have not been possible before.
10. LESSONS LEARNT

10.1.2 Human–Agent Interaction

The problem of human–agent interaction is a practical design problem. Interaction between human users and software systems is in general provided for some kind of interface (e.g. console, graphical user interface) that allows a human user to manipulate a software system in order to place requests, trigger actions or acquire information. Multiagent systems are not different in this respect. However, the modelling of human–agent interaction is not straightforward because of the inherent complexity of the notion of agency and its implication for software agent design.

The aim of a user interface is to enable a human to control and manage the processing of a software system. For the acceptance of software systems, it is of utmost importance to keep the “user in the loop”. This means that the user is aware of all important events that occur and the decisions that need to be made. After all, software is just a tool to assist humans in their activities. Users want to be in control of their software, particularly for critical transactions or the handling of sensitive information. Jennings and Wooldridge [82] remark in the context of software agents: “For individuals to be comfortable with the idea of delegating tasks to agents, they must first trust them. Both individuals and organizations will thus need to become more accustomed and confident with the notion of autonomous software components, if they are to become widely used. Users have to gain confidence in the agents that work on their behalf, and this process can take time. During this period, the agent must strike a balance between continually seeking guidance (and needlessly distracting the user) and never seeking guidance (and exceeding its authority).”

The property of agent autonomy is a problem in this respect. A user agent modelled in the conventional fashion appears to be hardly autonomous because it performs operations typically initiated by the commands of its user. The degree of autonomy of a user agent is difficult to determine and may be even transient (as the above quote from Jennings and Wooldridge [82] points out). Moreover, a conventionally modelled user agent also interferes with the proactiveness property because it typically acts as a proxy that passively passes information between a human and other components of the software system. The degree to which a user agent is proactive is also difficult to determine and may change over time.

The difficulties with user agents stem from ambiguity in the definitions of the properties of the weak notion of agency. Interestingly, the characteristics of a software agent
change with different perspectives. A user agent may be seen as fully autonomous and proactive from inside a multiagent system, even though its external view as described above suggests otherwise. It seems perfectly possible to model a user agent that syntactically and semantically complies with the weak notion of agency as long as it hides its interactions with its user (see Figure 10.1). This way, other software agents simply do not recognise what triggered a user agent to perform an action: its own decision-making or someone else.

![Visualisation of the external and internal view to the design of a user agent.](image)

**Figure 10.1:** *Visualisation of the external and internal view to the design of a user agent.*

Obviously, it is possible to change the semantics of a particular software agent (role) from being an autonomous problem-solver (internal view) to being a representative that acts on behalf of someone else (external view). This dualism between an external view and internal view of a multiagent system masks inconsistencies with our general understanding of the notion of agency. Consequently, the semantics of a software agent as a software component is inconsistent and mutable.

The trade-off lies between the application of a uniform software agent paradigm and a practicable solution. Hence the modelling of multiagent systems based solely on software agents that fully comply with the weak notion of agency from any perspective seems to be rather impractical, since it constrains user control.\(^1\)

According to the internal view, the service consumer agent type in this thesis complies with the interpretation of the weak notion of agency presented in this thesis. It exhibits

---

\[^1^\] A drastic example of such a form of human computer interaction is illustrated by the famous negotiations between the crew of a spaceship and the HAL 9000 computer in Kubrick’s science fiction movie “2001: A Space Odyssey.”
autonomy because no other agent has direct influence on an agent’s decision-making. It interacts via messages on a social level. It is proactive because it issues composite service requests. It is reactive because it replies to composite service proposals in order to form binding contracts with the winning coalitions.

10.1.3 Integration of Legacy Software

Complex software systems include, among other components, data sources and data sinks, e.g. database and application servers, which typically provide server functionality or access to data following the client–server paradigm. Such software systems form the backbone of all current IT infrastructure. They are labelled legacy systems when new software systems are developed that incorporate these existing software systems in order to extend the functionality or to amalgamate new and old functionality so as to provide added value or automation benefits.

Legacy systems have been developed using conventional methods based on functional decomposition. They were not implemented to fit into a multiagent system in the first place. Hence the integration of legacy systems into a multiagent system (see Figure 10.2) presents a problem because they, or their fundamental building blocks, were not designed to comply with any notion of agency.

Figure 10.2: Sketch of the integration of legacy software systems into a multiagent system using proxy agents.

Genesereth and Ketchpel [58] survey three approaches for integrating legacy software into multiagent systems that have found currency in many agent-based implementations. However, all three approaches – transducer (more commonly called proxy), wrapper and
10.1. General Conceptual and Design Issues

rewrite – cannot guarantee compliance with the weak notion of agency in general. Firstly, Genesereth and Ketchpel [58] are focused on establishing agent-based interoperability based solely on an agent communication language (neglecting other properties of the notion of agency). Secondly, legacy systems are typically not designed and implemented to comply.

A straightforward integration of passive server applications from a client–server system is unsuitable. It violates the autonomy and the proactiveness properties of the weak notion of agency. For example, a proxy agent of a database system in its simplest form passively waits for requests that it transforms into database queries. It then queries the database and returns a result set. Such a proxy agent is purely passive and not proactive. It is also constrained in its autonomy because database access in most cases is essential for the proper functioning of a software system. Hence there is little room for the proxy agent to say no to a request (an often stressed metaphor to emphasise the property of autonomy, e.g. in Wooldridge [169]).

To implement proxy agents that fully comply with the notion of agency, additional features such as goal-oriented behaviour and proactiveness are necessary. Endowing a proxy agent with such features may not reflect the true nature of the proxied legacy system. Moreover, such an exercise is often unwanted (unnecessarily increasing a system’s complexity) and sometimes even impossible given the original design objectives of the legacy system (e.g. a database system is designed for reliable access to persistent data). Thus, in general, multiagent systems do not offer a universal approach to interoperability and integration of legacy systems because their underlying organisational decomposition differs significantly from the functional decomposition of the legacy software. The trade off is between the application of a uniform software agent paradigm and appropriately complex solutions. Again the dualism between agents as autonomous problem-solvers and agents as representatives acting on behalf of someone is evident. The notion of agency provides the former; in most cases the latter is necessary and sufficient.

10.1.4 Integration of Middleware Services

Middleware services provide support in many distributed software systems, e.g. publication and discovery of services, for indirect communication via blackboards, for remote
interaction and transaction safety, for trust and reputation and visual monitoring of system status. Middleware components provide essential functionality that is required for reliable, robust and secure operation of a software system. Middleware components are typically the result of functional decomposition. They usually operate in a strict client–server fashion.

The modelling of middleware components with software agents bears similar problems as the integration of legacy systems. They too have not been developed according to agent-based design principles in most cases. Middleware components are not in compliance with the weak notion of agency. In this respect, even the AMS and DF agents of the JADE agent toolkit [77] (based on the FIPA Abstract Architecture [49]) for example do not fully comply with the weak notion of agency. Both agents provide registry services for maintaining agents and services on an agent platform respectively. They are solely reactive components that return requests in a client–server fashion.

On the other hand, fully autonomous behaviour of infrastructure agents is not desirable because it may violate the robustness and reliability of a software system. Again the dualism between agents as autonomous problem-solvers and agents as representatives acting on behalf of someone is evident. The notion of agency provides the former; in most cases the latter is necessary and sufficient. Hence middleware infrastructure is on a lower level of abstraction. It should not be modelled with software agents and it should not accommodate the crucial roles of a multiagent system. Middleware components should be incorporated in the underlying agent platform such that a software agent can transparently use their functionality.

10.1.5 Agent Orientation as Uniform Paradigm

A software engineering paradigm is characterised by the properties of its fundamental building block. Uniformity is ensured with a consistent and uniform application of the fundamental building block for modelling elements of a problem space. In particular, syntax and semantics of the fundamental building block are immutable.

Consequently, all software agents of a multiagent system must comply with the same definition of agency, and all software agents must not exhibit varying degrees of compliance with that definition in the context of a uniform agent-oriented paradigm. Unfortunately,
as outlined above, there are some difficulties, e.g. with the integration and modelling of human users, legacy systems and middleware components. A straightforward integration of simple, passive or lightweight elements may result in either unnecessarily complex implementations or heterogeneous multiagent systems that incorporate agents based on various notions of agency and/or different degrees of compliance.

As a result, the uniformity of the agent-oriented paradigm cannot be ensured in all generality. The decision to adapt an agent-based approach must be carefully considered and depends on the characteristics of the problem space and its elements. All elements must embody the full set of properties of a specific notion of agency. The granularity of the elements must be coarse-grain, based on organisational decomposition. The interactions between the elements must be a key characteristic of the problem space, in particular their flexibility, adaptability, robustness, etc.

Only a full match of the characteristics of the problem space and the properties of the agent paradigm helps to leverage the full potential of an agent-based approach. This matching, however, is limited in many cases, as discussed in the next section. Agent orientation is focused on knowledge-level communication between complex autonomous problem-solvers, and hence is not suitable to implement low-level business logic.

It must be noted that it is possible to find a definition of the notion of agency that is general enough to model any kind of software component and any problem space. However, this is not the aim of agent orientation, which is for a higher level of abstraction (Odell [129]). The weak notion defines the very nature of agency. Any weakening of it also weakens the notion and leads to even more ambiguity about what constitutes a software agent and what agent-oriented software engineering is about. It should be noted that many approaches in related work do not abide by the uniformity principle.

10.1.6 Agent Orientation as a Universal Paradigm

The uniformity of an agent-based paradigm with regard to a specific notion of agency is tricky to maintain for several reasons. Many real-world problems and their elements do not match the characteristics imposed by an agent definition. As a result, agent orientation does not embody a universal software engineering paradigm that can be applied to creating solutions for any type of problem. If the paradigm is applied in a uniform manner, it is
not universally applicable because it imposes strict requirements on the properties of its
building blocks. If the paradigm is applied in a non-uniform way, it maybe universal, but
the distinctive character and principles of agent orientation are lost.

An approach often taken to justify the use of agent-oriented methods in mainstream
software engineering is by combining it with other paradigms. For example Odell [129]
recommends to use a ‘well thought-out mixture’ of agent-oriented and object-oriented
paradigm. This matches the author’s experience of designing, implementing and testing
multiagent systems. Today, most agent toolkits are based on object-oriented languages
such as Java. Even communication between software agents (e.g. messages, their content
and semantics) can be modelled and implemented with object orientation.

This thesis supports the claim that software agents are an evolutionary step (Odell
[129]) that can be used to model certain problems at the system level in a very natural
way that no other fine-grain method or paradigm is able to achieve. Agent orientation
helps to describe and model more complex subsystems where object orientation is usually
to fine-grain to find an appropriate way to deal with complexity: “In this respect, agent-
oriented concepts are an extension of those available in other paradigms.” [81] These
concepts focus on the system design level. They can be complemented and refined, e.g.
with object-oriented methods for the detailed design of the software agents themselves.
However, it must be noted that Zambonelli et al. [177] warn that merging object-oriented
and agent-oriented methodologies risks exploiting unsuitable means of abstraction and
decomposition for modelling multiagent systems.

As such, the border with other paradigms that are used together with agent orientation
for the design of a single software system must be clearly defined. Especially what agent
orientation focuses on and for what problems it constitutes a valid design method. An
agent-based approach can be considered under the conditions that are outlined by Jennings
[78, p.36]:

- Natural decentralisation
- Multiple loci of control
- Multiple perspectives
- Multiple interests.
Hence the agent paradigm is particularly useful for modelling emergent non-deterministic system behaviour. Accordingly, software agents are a good means of modelling real-world problems that yield very similar characteristics because in these cases agent-based abstraction, decomposition and structuring principles are a natural match.

On the other hand, there are a number of limitations to an agent-based approach that make it unsuitable for domains in which determinism and optimality of computation are key concerns. Jennings and Wooldridge [82, p.10] and [81, p.17] identify the following limitations to multiagent systems due to intrinsic properties of agency:

- **Lack of a central controller:** “An agent-based solution may not be appropriate for domains in which global constraints have to be maintained, in domains where a real-time response must be guaranteed, or in domains in which deadlocks or livelocks must be avoided.”

- **Non-optimality of solutions:** “An agent’s actions are, by definition, determined by that agent’s local state. However, since in almost any realistic agent system, complete global knowledge is not a possibility, this may mean that agents make globally sub-optimal decisions. The issue of reconciling decision-making based on local knowledge with the desire to achieve globally optimal performance is a basic issue in multiagent systems research.”

- **Unpredictability:** “Although agent interactions represent a hitherto unseen level of sophistication and flexibility, they are also inherently unpredictable in the general case. As agents are autonomous, the patterns and the effects of their interactions are uncertain.”

### 10.2 Design and Implementation Issues of Interaction Protocols

The dynamic aspects of a multiagent system – the agent interactions – form the essential and distinct characteristic of agent orientation (see Henderson-Sellers and Gorton [67, p.3 footnote 2] and Huhns [75]). In contrast to the static design of a multiagent system that is relatively simple in many cases, the dynamic aspects of a multiagent system tend to
be very complex and are yet little understood due to their complex nature (Jennings [80] emphasises: “Although agent interactions represent a hitherto unseen level of sophistication and flexibility, they are also inherently unpredictable in the general case. As agents are autonomous, the patterns and the effects of their interactions are uncertain.”). The inherent non-determinism and temporal aspects of agent interactions result in difficulties during design, implementation and verification of agent interaction protocols. Hence they also affect the approach of this thesis presented in the chapters 4–7, particularly its performance.

10.2.1 Concurrency Issues in Agent Interactions

The majority of agent-based approaches (e.g. see Chapter 3) do not explicitly model message timeouts with the dynamic design of a multiagent systems. The standards body for agent-based specifications, FIPA [48], defines in the FIPA ACL Message Structure Specification [50] an attribute reply_by as part of a message that: “denotes a time and/or date expression which indicates the latest time by which the sending agent would like to receive a reply.” However, FIPA standards do not define an implicit message timeout modelling apart from the reply_by attribute. They recommend handling message timeout issues on the protocol level with additional communication acts. Consequently, message timeout handling must be dealt with manually by the programmer.

A message timeout is a temporal constraint attached to a message by a sending agent. Message timeouts provide the recipient of a message with additional information for determining when a response to the message is expected at the latest. Message timeouts are specified either absolutely – as a concrete time and date – or relative to a particular event with a value denoting the duration in milliseconds.

Message timeouts are essential for ensuring agent autonomy with respect to the weak notion of agency. On the one hand, a sender agent reduces its dependency on a response message by clearly expressing when it will resume processing even without a proper reply message. On the other hand, a recipient agent can use the message timeout of an incoming request to decide whether there is enough time for it to process the request and to reply to it. Moreover, with the help of a message timeout the recipient agent determines whether its reply message has been added to the sender agent’s mailbox on time. Thus both sender
and recipient agent remain in full control of their internal state and resources and may decide on a different way to achieve their goal in case the message timeout constraint is too restrictive.

In general, a software agent adds a message timeout constraint to a message if it expects that the response message will affect its internal state. When a message timeout has expired, the sender agent rolls back to the state it was in before sending the message and deliberates about other ways to accomplish its goal. Also the recipient agent can use the information about the timeliness of its response in order to roll back to a safe state in case it performed changes to its internal state or updated its knowledgebase during the processing of the request and the creation of the reply message. Thus message timeouts are crucial for ensuring full agent autonomy.

Without message timeouts agent autonomy is limited in a number of situations. A sender agent’s autonomy is partially restricted when sending messages without timeouts if the

- Recipient agent is busy performing actions or processing other messages
- Recipient agent needs too much time to calculate a response
- Recipient agent delays deliberately (to achieve some advantage)
- Network between sender and recipient agent experiences delays or high network load.

A sender agent’s autonomy is fully restricted and stalls when sending messages without timeouts if the

- Recipient agent is in a state in which it does not reply to certain requests
- Recipient agent exhibits malicious behaviour
- Recipient agent code causes exceptions and errors that disturb smooth execution
- Recipient agent ceases to exist after receiving the message
- Server hosting the recipient agent fails
- Network between sender and recipient agent fails.
10. LESSONS LEARNT

The implementation of message timeouts for agent interactions turns multiagent systems into real-time systems with hard timing constraints. Real-time systems are a class of software systems in which the timeliness of actions and communication acts are crucial for correctness (e.g. see Douglass [37]). In this respect, the timeliness of a response is essential for the further processing of a software agent. However, a missed message timeout may not cause system failure due to the intrinsic capability of software agents to deliberate about alternatives for achieving a certain goal. But it still effects the optimality of the processing in multiagent systems.

Message timeouts have an impact on the behaviour of software agents and their respective interactions. Multiagent systems are also concurrent systems because every software agent resides in its own thread of control. Software agents interact with asynchronous messaging. Hence what impact the real-time constraints of message timeouts have on concurrency phenomena such as starvation, deadlocks and livelocks needs to be investigated. There are a number of issues to consider to ensure correctness and reliability of an interaction protocol.

Table 10.1: Concurrency problems of different combinations of blocking, timeout and communication mode.

<table>
<thead>
<tr>
<th>Timeout mode</th>
<th>Blocking mode</th>
<th>Off</th>
</tr>
</thead>
<tbody>
<tr>
<td>On</td>
<td>Temporal livelock</td>
<td>Exclusive read/write</td>
</tr>
<tr>
<td>Off</td>
<td>Starvation</td>
<td>Starvation</td>
</tr>
<tr>
<td></td>
<td>Deadlock</td>
<td>Exclusive read/write</td>
</tr>
<tr>
<td>Uni-directional</td>
<td>Bi-directional</td>
<td>Uni-directional</td>
</tr>
<tr>
<td>Communication mode</td>
<td>Bi-directional</td>
<td></td>
</tr>
</tbody>
</table>

In order to investigate the impact of message timeouts, three relevant modes concerning agent interactions exist. Firstly, in blocking mode a sender agent stops processing until a response to a previously sent request message has been received. Without block-
ing mode, the sender agent may perform actions (e.g. processing further messages) after sending a request message. Secondly, in message timeout mode a sender agent sends messages using message timeouts. Thirdly, there is a communication mode. Uni-directional communication exists between two software agents if – at any time – only one agent acts as sender and the other agent acts as recipient. Bi-directional communication exists if two software agents initiate interaction with each other simultaneously. Table 10.1 depicts all combinations of the modes and their particular concurrency problems.

The first case to be discussed are situations in which software agents communicate without message timeouts. Figure 10.3 depicts such a situation. A sender agent performs some operation which results in sending out a request message to another software agent – the recipient. The sender agent then changes into a waiting state. The waiting time of the sender agent converges towards infinity if it does not receive a response message. (See the enumeration of situations above in which the sender agent’s autonomy is fully restricted.) Thus the sender agent will not progress with its execution at all. The technical term for such a situation is starvation.

![Sequence Diagram](image)

**Figure 10.3:** UML sequence diagram of a starvation situation of the sender agent.

Starvation can occur for a sender agent in all situations in which its internal state depends on a response and it operates without message timeouts. Hence starvation may also occur if the sender agent operates without blocking mode as long as no new messages
arrive or no further actions are triggered by the sender agent itself. Starvation can be avoided by using message timeouts, as noted in Table 10.1.

There is another concurrency problem that occurs solely if software agents do not communicate using message timeouts. A deadlock between two software agents occurs if both software agents concurrently send request messages to each other and then block until the respective responses arrive. If the request messages arrive just after each of the two software agents commenced blocking mode, both agents stall indefinitely for a response, as illustrated in Figure 10.4. The use of message timeouts avoids deadlocks in any situation.

![UML sequence diagram of a deadlock situation between two bi-directionally communicating software agents in blocking mode and no message timeouts.](image)

**Figure 10.4:** UML sequence diagram of a deadlock situation between two bi-directionally communicating software agents in blocking mode and no message timeouts.

The concurrency phenomenon of a livelock can occur in cases in which two software agents engage concurrently in bi-directional communication with each other. A livelock is a situation in which the actions of one software agent are performed in response to the action of another software agent in an indefinite self-enforcing cycle. As with deadlocks, software agents trapped in a livelock are unable to progress but they are not halted. In contrast to deadlocks, livelocks keep both software agents busy responding to each other.

Figure 10.5 shows two non-blocking software agents that send request messages to each other. If both do not perform further operations and disregard the request message
of the other software agent, they may deadlock as described above. Assume both software agents commence processing their incoming request messages and they detect that they are concurrently engaged in a cross-interaction. Now both agents face the problem of whether they should reply to the request of the other software agent and disregard their own request. There are four alternative combinations, as modelled in Figure 10.5, and two in which both software agents decide on the same action (alternatives a and d) thus leading to a livelock. If both software agents insist on initiating their own interaction with the other software agent and since they do not have knowledge about the decision of other software agents (autonomy property), both keep exchanging messages until one of the two positive message combinations occurs or at least one of the two software agents decides to give up and try something different.

![Figure 10.5: UML sequence diagram of a logical livelock situation between two bi-directionally communicating software agents without timeout and blocking modes.](image)

A livelock that solely depends on the decision-making of both involved software agents is denoted a *logical livelock*. It may also occur if both software agents use message timeouts (see Table 10.1). However, logical livelocks can be avoided in blocking mode. Dealing with logical livelocks so that software agents can detect cross-interaction increases the
10. LESSONS LEARNT

complexity of the design and implementation of decision-making components.

Figure 10.6 outlines a temporal livelock situation between two bi-directionally communicating software agents in blocking and message timeout mode. A temporal livelock is caused by temporal inter-dependencies between message timeouts and occurs if both software agents have set very similar timeout deadlines: $t_1$ and $t_3$. As with deadlocks, a temporal livelock occurs if both software agents receive requests of the other software agent just after entering blocking mode. Neither software agent processes incoming requests because of the blocking mode until the message timeout of their own requests is due. Only then does a software agent process and respond to a request. However, if both responses arrive after the message timeout of the other software agent (e.g. at time $t_2$ with $t_2 > t_1$ for Agent 1 and time $t_4$ with $t_4 > t_3$ for Agent 2) both interaction attempts fail. If both software agents insist on initiating an interaction with the other software agent, they may repeatedly experience the livelock until changes in process scheduling, network delays or varying message timeouts create a time window for one agent to reply on time or to decide on a different action. A temporal livelock adds to the complexity of a logical livelock.

![Figure 10.6: UML sequence diagram of a temporal livelock situation between two bi-directionally communicating software agents in blocking and timeout modes.](image)

Messaging in non-blocking mode results, in general, in a drastic increase of compu-
tational complexity because it is necessary to keep the integrity of a software agent’s knowledgebase and internal state during potentially multiple interactions with different interaction partners and agent-internal actions. Session management is necessary to organise different conversations. Mutual exclusion between different conversations must be ensured with transaction-based exclusive read and write, two-phase commit protocol and rollback mechanisms between different sessions when they manipulate the agent’s knowledgebase or internal state.

In summary, the modelling of social dependencies between software agents in a multiagent system, and the design of interaction protocols in particular, is limited by concurrency issues. Message timeouts are absolutely crucial for ensuring agent autonomy because they help to avoid starvation and deadlocks. The blocking mode simplifies the design and implementation of a software agent because no extra effort is required to maintain the integrity of the agent’s knowledgebase and internal state (mutual exclusion). The non-timeout mode is only practicable if software agents send single messages for which they do not expect response. Robust and reliable interactions can be established in general only in one specific case when software agents communicate in uni-directional mode using message timeouts and blocking mode (see Figure 10.7 and Table 10.1).

![Image of UML sequence diagram](image_url)

**Figure 10.7:** UML sequence diagram of a uni-directional communication without concurrency problems due to timeout and blocking modes.
10. LESSONS LEARNT

10.2.2 Implementation Issues of Unicast Message Timeouts

Multi-purpose agent toolkits such as JADE [9, 77], FIPA-OS [53, 139] and JIAC [84] implement the FIPA specification by incorporating a reply_by attribute in their respective ACL message or speech act components. Thus an agent programmer can set and get the value of the attribute. However, apart from the attribute, the FIPA standard does not specify an implicit message timeout handling mechanism that supports the reliable validation of message timeouts. Consequently, timeout handling must be implemented manually. As a result, the FIPA specification exhibits the following shortcomings, causing uncertainty of message timeout handling in unicast interactions.

The first and most important issue is the synchronisation of computer clocks. Message timeouts can be specified absolutely with date and time values, or relatively as a time interval after some event. If a message timeout is specified with an absolute value, the synchronisation of clocks by different software agents is of crucial importance. Hence multiagent systems are distributed systems. Therefore, it cannot be assumed that all software agents operate according to exactly the same time.

Mills [121, 122] delineates a synchronisation algorithm and the Network Time Protocol (NTP) for use in large-scale distributed systems that achieves a synchronisation accuracy in the general Internet of a few tens of milliseconds. NTP is today the most common time synchronisation protocol in the Internet. There are more precise point-to-point synchronisation methods available. However, a point-to-point synchronisation between software agents is impractical due to the asynchrony of agent interactions. A software agent may engage in multiple nested conversations with different communication partners. Every new interaction breaks the synchrony with the previous communication partner.

The uncertainty about the synchronisation of two software agents creates a dilemma for the recipient of a message. It simply cannot unambiguously decide whether a reply it sends will arrive on time.

A second issue is the manual character of programming message timeouts following the FIPA standard. Neither a sender agent nor a recipient agent can unambiguously determine if a reply message was sent on time. Message passing is usually implemented with a single connection that channels a remote procedure call. An agent typically calls a specific method of the agent main class or the underlying agent platform (e.g.
sendMessage(ACLMessage(aMessage)) and waits until it receives back the focus.

Figure 10.8 depicts the situation. A recipient agent can only check the time \( t_{\text{before}} \) immediately before sending a response message and the time \( t_{\text{after}} \) after it receives back the focus. These two times are usually different because the message transport takes time, subject to network load, network volatility and network distance between the two software agents. If the duration between \( t_{\text{before}} \) and \( t_{\text{after}} \) is denoted by \( \Delta t \), then the actual time \( t_{\text{add}} \) for adding a reply to the sender agent’s mailbox lies in the interval \( t_{\text{before}} < t_{\text{add}} < t_{\text{after}} \). Thus a reply message may arrive on time (e.g. before \( t_{\text{timeoutB}} \)), even though time \( t_{\text{after}} \) suggests differently, leaving the recipient agent in uncertainty.

![UML sequence diagram](image)

**Figure 10.8:** *UML sequence diagram illustrating the uncertainty of a recipient agent about the exact timing of its response message.*

Even the sender agent cannot precisely determine the timeliness of reply messages. Its mailbox is decoupled from the agent, which can check the timeliness of the reply only after being restarted (e.g. upon the arrival of the reply) or as part of a global routine (e.g. the main loop of the agent) at time \( t_{\text{check}} \) (see Figure 10.9). In both cases, the time check may be delayed depending on whether the agent was busy performing some action or because restarting the agent was delayed due to high server load.

Hence the manual handling of message timeouts is imprecise and therefore inadequate. The above issues may lead to mistaken beliefs that trigger software agents to take wrong
actions and can result in interaction inconsistency for only temporal reasons.

FIPA recommends that these problems be solved on the protocol level. Thus the FIPA message timeout mechanism expresses recommendations rather than constraints. However, a solution on a protocol level, e.g. with additional acknowledge messages, is not possible. The timeliness of such acknowledge messages also can not be precisely determined. A recipient would not wait an indefinite time for an acknowledge message and set a message timeout to ensure its autonomy. Thus an acknowledge message for an acknowledge message becomes necessary resulting potentially in a livelock situation between sender and recipient agent. The robustness of agent interactions in general can not be guaranteed with FIPA standards only.

This thesis proposes a different concept for message timeout handling – message timeouts are actively managed by a software agent’s mailbox. The fundamental assumption underlying the implementation is that the decision about the timeliness of response messages should be performed in a centralised and precise fashion only by the software agent that asserted the temporal constraint in the first place – the sender of a request message, or more precisely its mailbox.

The general processing is shown in Figure 10.10. When a sender agent sends a message
with a timeout constraint, the mailbox creates a corresponding conversation session and starts a timer (in a separate thread) preset to the envisaged timeout deadline. The conversation session includes a status flag that is set to active. If a response message arrives on time, it is added to the agent’s message queue and the status flag is set to replied (see Figure 10.11). If the agent hibernates, the mailbox also restarts the agent in order to enable it to resume processing. If no response message arrives on time, the timer triggers an action that firstly sets the status flag to timedout and secondly attempts to restart the software agent, if required (see Figure 10.12).

Figure 10.10: UML sequence diagram depicting the general interactions between software agent and mailbox components during interactions with active message timeout.

The status flag of the conversation session indicates the timeliness of a response message. It can be communicated immediately after adding a message to the associated recipient agent because even though agent communication as a whole is asynchronous, the transmission of a single message is still based on a synchronous connection channel in which a remote procedure call is performed. This remote procedure call is also used to convey the timeliness of a response message to the recipient agent in the form of a
boolean return parameter: \(true\) denotes that a response was sent on time or that no message timeout was defined; \(false\) denotes the message was sent late.

![Figure 10.11: UML sequence diagram of alternative A – response sent on time.](image)

![Figure 10.12: UML sequence diagram of alternative B – message timeout elapsed.](image)

It is up to the sender agent to close a conversation session, when, for example, it has received all expected responses or the message timeout has expired. Messages that arrive late or after the termination of a conversation session are not added to the sender agent’s message queue and the respective return parameter of the remote procedure call is marked \(false\) (see Figure 10.13 and Figure 10.14).

The presented concept for message timeout handling is FIPA-compliant. Message objects carry timeout information as a recommendation. However, the reactive mailbox, together with return parameters for the remote procedure calls, determines the timeliness of a response message precisely and unambiguously for both sender and recipient agent.

A last issue concerning unicast messaging among software agents has to do with rel-
10.2. DESIGN AND IMPLEMENTATION ISSUES OF INTERACTION PROTOCOLS

Figure 10.13: UML sequence diagram of alternative C – late response with session object.

Figure 10.14: UML sequence diagram of alternative D – late response without session object.

ative message timeouts. Relative message timeouts are usually defined as the maximum duration in milliseconds allowable between sending a request and receiving a response. The sending of messages takes time because it depends on network load, network volatility and network distance between two software agents. Thus the question is when to start the timer object: just before sending a request or just after the focus has returned to the sending agent? Hence due to the network delay, the decision as to when to start the timer object may have a strong impact on the sender agent’s performance, particularly if this agent has to obey temporal constraints of its own. The current implementation allows programmers to choose between both options.

10.2.3 Implementation Issues of Multicast Message Timeouts

Multicast messages are sent to more than one recipient agent. Technically, when a sender agent sends a multicast message, the message is multiplied and a full copy is sent to every
recipient agent separately. As with unicast messages, multicast messages can also be sent with absolute or relative message timeouts. However, due to the number of recipient agents issues arise regarding fairness among recipient agents and the autonomy of the sender agent.

Fairness is guaranteed if all recipient agents have the same time to react to a request message. Fairness increases the probability of a higher quantity and quality of responses, since all recipient agents have an equal chance to create a response. Fairness depends on the message timeout type and value, as well as on the number and order of recipient agents. However, a high degree of fairness may compromise the autonomy of the sender agent.

The autonomy of the sender agent extends to its ability to comply with temporal constraints imposed on itself regarding to sending of a particular multicast request message. The more freedom the sender agent allows recipient agents for returning a response, the less freedom it has to comply with its own temporal constraints. This is a problem in time-critical domains.

Fairness and autonomy depend on the choice of message timeout type. Absolute message timeouts guarantee sender agent autonomy because the sender agent sets the message timeout deadline. But absolute message timeouts do not provide fairness. The message timeout deadline is set for all recipient agents upfront. Thus the order in which the recipient agents receive the message determines how much time they have left for replying. Recipients at the tail of the order have significantly less time compared to recipients at the top because a multicast message is sent sequentially to all recipient agents. The maximum duration for replying to a request depends on both the length of the timeout and the number of recipients. Hence the response time \( \Delta t_{\text{response}} \) available to a recipient \( r_n \) is the difference between the specified message timeout \( \Delta t_{\text{timeout}} \) and the sum of time intervals \( \Delta t_{\text{send}}(r_m) \) taken to send a copy of the message to all recipients \( r_0 \ldots r_{n-1} \) plus the time interval \( \Delta t_{\text{send}}(r_n) \) taken to send a copy of the message to recipient \( r_n \) (see Equation 10.1).

\[
\Delta t_{\text{response}}(r_n) \leq \Delta t_{\text{timeout}} - \left( \sum_{m=0}^{n-1} \Delta t_{\text{send}}(r_m) - \Delta t_{\text{send}}(r_n) \right) \tag{10.1}
\]

The time necessary for sending each message copy depends on the network delay,
10.2. **DESIGN AND IMPLEMENTATION ISSUES OF INTERACTION PROTOCOLS**

volatility (including network outages and retry) and distance between the sender and every recipient agent, as well as the message size. In other words, $\Delta t_{send(r_m)}$ may differ significantly. Consequently, the network characteristics may compound the fairness problem by exacerbating the problems caused by sequential broadcasting (noted above).

Relative message timeouts also do not yield fairness. If the message timeout period begins set before sending the message (early base), the recipient agent at the tail of the order (as with absolute message timeouts) is disadvantaged (see Equation 10.1). If the message timeout period begins after sending the message (late base), the recipient agents at the top of the order are over-advantaged because they gain additional time on top of the original length of the message timeout $\Delta t_{timeout}$ (see Equation 10.2). The additional time includes the time interval $\Delta t_{send(r_n)}$ of the network delay that elapses before the return of the control flow back from recipient agent $r_n$ to the sender agent plus the sum of time spent sending a copy of the request message to all subsequent recipient agents in order.

$$\Delta t_{response(r_n)} \leq \Delta t_{timeout} + \Delta t_{send(r_n)} + \sum_{m=n+1}^{max_n} \Delta t_{send(r_m)} \quad (10.2)$$

Relative message timeouts support sender agent autonomy when the message timeout is set before sending out a multicast message (early base). This way, the sender agent determines precisely when it resumes processing. However, if the message timeout is started after regaining the focus from the last recipient agent (late base), the total time the sender agent needs to wait before resuming processing is variable: it depends on the total time needed to send out the message. This total time depends on network delay, volatility (including network outages and retry), distance between sender agent and every single recipient agent, the message size and the number of recipients. No autonomy is guaranteed. This is a problem in time-critical domains where high sender agent autonomy is needed.

The fairness issue of relative message timeouts can be remedied by simply splitting a multicast message into multiple unicast messages all with the same relative message timeout. Setting the base for the message timeout to after regaining focus from the recipient agent yields highest fairness, because no processing time is lost (from a recipient’s perspective) due to network conditions. In contrast, if the base is set to before sending out the request message, the response time available for a recipient agent is reduced by the
time required to transfer and add the message to its mailbox. Multiple unicast messages, however, do not improve sender agent autonomy.

In summary, characteristics of multicast message timeouts (see Table 10.2) and the uncertainty about network conditions and the behaviour of recipient agents makes it difficult to find a suitable message timeout setting. The choice of message timeout type must depend on the problem domain.

Table 10.2: Comparison of the support of sender agent autonomy and recipient agent fairness of available multicast message and message timeout settings.

<table>
<thead>
<tr>
<th>Multicast messages</th>
<th>Absolute timeout</th>
<th>Relative timeout</th>
<th>Multiple unicast messages</th>
<th>Relative timeout</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Early base</td>
<td>Late base</td>
<td>Early base</td>
<td>Late base</td>
</tr>
<tr>
<td>Autonomy</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Fairness</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
</tbody>
</table>

Furthermore, it is evident that asynchronous messaging among autonomous software agents significantly increases the complexity of the design and implementation of multi-agent systems (agent platforms and software agents). It requires additional measures such as conversation objects, multi-session management and rollback mechanisms for maintaining the integrity of an agent’s knowledgebase and internal state. Consequently, agent-based implementations are more complex than implementations based on other paradigms, e.g. object orientation or service orientation which rely on weaker real-time and concurrency assumptions.

10.2.4 Calculation of Message Timeouts

The use of message timeouts is necessary in order to ensure agent autonomy, as noted in the previous section. Thus it is necessary to consider how to calculate message timeouts efficiently. The calculation depends on a number of factors, such as:

- Temporal constraints imposed on the sender agent
- Minimum processing time needed for recipients to respond
10.2. DESIGN AND IMPLEMENTATION ISSUES OF INTERACTION PROTOCOLS

- Status of recipient agent (e.g. idle, busy, etc)

- Performance parameters of the recipient agent’s environment

- Network distance and likely delay between sender and recipient agent.

The calculation of a message timeout assumes that the recipient agent will respond. In general, the sender agent is interested in specifying a relatively short message timeout because it allows great freedom, particularly if the sender agent needs to meet hard time constraints itself (e.g. a message timeout from a different but relevant conversation). On the other hand, the recipient agent needs a certain amount of time to process a message and create a response. The time needed depends firstly on the complexity of its reasoning and decision-making capabilities. (Many artificial intelligence methods such as planning or matchmaking exhibit high computational complexity.) Secondly, it depends on the status of the recipient agent (which may be busy finalising the processing of another action or message). Thirdly, it depends on the conditions of the host of the recipient agent (e.g. high server load).

For the sender agent, the message timeout delay is a trade-off between its own autonomy and the likelihood of receiving a response. Unfortunately, due to agent autonomy the sender agent does not have any information about the behaviour of the recipient agent and its hosting environment. Moreover, the network distance and delay between sender and recipient agent have an impact on the response time of the recipient agent. Network conditions may be volatile and thus not subject to precise calculation.

Message timeouts can be: static or dynamic. Static message timeouts are hard-coded durations specified by a programmer. They are applied directly. Dynamic message timeouts adjust a programmer-specified duration to a specific situation with an estimation of network conditions and recipient agent behaviour. Static timeouts do not adapt to changing conditions and, therefore, increase the probability of interactions failing in highly volatile and dynamic environments. Besides the apparent advantage of adaptation, dynamic message timeouts also bear an intrinsic shortcoming. Due to agent autonomy, a sender agent can not determine why exactly a conversation has timed out. Thus an appropriate adjustment of a message timeout is difficult to achieve.
The effective calculation of message timeouts is an open issue. The impact of message timeouts on the effectiveness of agent communication has not been systematically studied.

10.2.5 Race Conditions

Multiagent systems based on the weak notion of agency may experience exceptional situations during interactions between three or more software agents because they exhibit the characteristics of hard real-time systems and concurrent systems in combination with the property of agent autonomy. As such, the order and timing of events and actions are particularly important for the correct processing of a multiagent system. A fatal error that occurs in the form of an unexpected sequence or timing of such events and actions is called a race condition.

Consider the example illustrated in Figure 10.15 in which three software agents $A$, $B$ and $C$ intend to collaborate. Software agents coordinate team actions by exchanging messages. In the given scenario, agent $A$ sends a request for proposal (RFP) to the agents $B$ and $C$. Assume that $A$ sends the RFP message first to $B$ and then to $C$ and that neither $B$ nor $C$ can on its own create proposals separately. Therefore they attempt to collaborate and exchange requests for quotes (RFQ) with each other.

![Figure 10.15: UML sequence diagram of a race condition caused by volatile network conditions and/or network distances between software agents.](image)

A race condition occurs in this scenario if the RFQ message from $B$ to $C$ reaches $C$ before the initial RFP message from $A$ reaches $C$, e.g. due to network conditions.
and/or the network distance between the software agents. Hence a race condition can occur if different messages from different software agents directly depend on each other. The emerging race condition prevents parts of the multiagent system from functioning correctly and, therefore, compromises the efficiency of the entire multiagent system.

A key problem of the agent paradigm is how to form well-defined and robust interaction protocols, especially if many different agent roles are involved. Race conditions can be caused by dependencies between different agent roles and messages as well as the uncertainty about network conditions and distances between software agents. Moreover, race conditions can occur as cross-cutting side effects across multiple interaction protocols.

One option to minimise the risk of race conditions is to incorporate additional messages into interaction protocols that facilitate the exchange of control and status information. However, this option has shortcomings. It increases network load and communication overhead. Moreover, it exposes interactions to a higher probability of being timed-out, as outlined in following sections.

An alternative approach is to establish well-defined structures and hierarchies for modelling the relationships and interactions between agent roles. This way, agent interactions are more robust and proceed in a more controlled fashion. But they are, then, less flexible and more likely to violate the autonomy property of the weak notion of agency. The property of agent autonomy requires software agents to be able to decide autonomously whether to join, leave or reject such structures.

10.3 Runtime Issues in Multiagent Systems

The complexity and uncertainty of multiagent systems is particularly seen at runtime. Similar issues can be observed in concurrent and real-time systems. However, multiagent systems are inherently more complex than the other systems because of the inherent property of agent autonomy. The runtime issues presented in the following influence the emergence of system behaviour in multiagent systems and, therefore, have a direct impact on the optimality and effectiveness of multiagent systems in general, and on the approach presented in this thesis (in chapters 4–7 in particular).
10.3.1 Ageing Information

Software agents cannot decide with certainty whether a piece of information is current or obsolete. There are a number of reasons for this problem. Firstly, the autonomy property of the weak notion of agency implies only partial observability of multiagent systems, and this results in uncertainty about the correctness, completeness and timeliness of the information available to a software agent. Software agents disclose only as much information as they believe is required to achieve their goals according to private preferences, especially in competitive environments.

Secondly, the autonomy property also requires software agents to exchange information by asynchronous message-passing. Since software agents decide autonomously if, when and how to process incoming messages, a certain amount of time will elapse between receiving and processing messages. The timeliness of information is uncertain.

Hence the information that a software agent possesses about the internal state of another software agent may not fully reflect the real situation. Moreover, that internal state may change rapidly through proactive operations and without noticing interaction partners.

There are two ways to improve the situation. Firstly, software agents must engage in constant information exchange in order to keep their knowledgebases as up to date as possible. Secondly, software agent must conduct decision-making based on uncertainty measures. Both procedures, however, result in extensive communication in comparison to synchronous communication in deterministic environments: the former directly and the latter indirectly through a higher probability of request-response failures.

10.3.2 Ageing of Messages

The ageing of messages can be interpreted as a special case of ageing of information caused by message timeouts. In the case of messages, ageing does not refer to the content of the message but rather to the message as it is subject to the temporal constraints during the execution of interaction protocols.

Firstly, the message timeout \( t_{\text{timeout}} \) specified by a message sender may be set too short in relation to the travel time \( \Delta t_{\text{delivery}} \) of the message through a multiagent system. The travel time \( \Delta t_{\text{delivery}} \) strongly depends on the size of the underlying network, the
relative position of communication partners and dynamic parameters such as network load, volatility and reliability that are difficult to predict.

Secondly, the autonomy property of the weak notion of agency requires software agents to exchange information by asynchronous message-passing. Since software agents decide autonomously if, when and how to process incoming messages, some time elapses between receiving and processing them. For example, a software agent may be occupied processing messages received earlier or it may be in the process of proactively performing some action – both of which delay the processing of a message and the creation of a reply. Thus the ageing of messages cannot be remedied in general by implementing particular message queue strategies such as first in first out (FIFO) or priority-based.

The loose coupling between software agent and mailbox prevents a solution to this problem. To handle message ageing consistently, two measures must be taken. Firstly, message timeouts must be created dynamically based on an estimation of the expected network characteristics. Secondly, software agents must implement conversation tracking and rollback mechanisms in order to keep their knowledgebase and internal state consistent. Message timeouts support the decision whether a message has been processed on time or not. If processing was late, sender and recipient agents may have already changed states or adopted new roles. In such a case, software agents roll back to the last state prior to processing a message.

10.3.3 Uncertainty about the Status of Communication Partners

A software agent cannot fully predict the internal state of a communication partner. Based on the property of agent autonomy, software agents do not reveal private information, particularly not in competitive environments (partial observability of multiagent systems). Moreover, asynchronous messaging decouples request from response messages. A response message is sent via a separate network connection. Thus a sender agent will never know why a response arrives delayed or not at all. Effective exception handling is impossible. This means that uncertainty remains even if software agents agree to follow a certain interaction protocol prior to interaction. A response message may be delayed or not arrive at all for logical, software, hardware and network reasons, such as the

- Recipient agent is busy performing some action or processing other messages
10. LESSONS LEARNT

- Recipient agent needs too much time to calculate a response
- Recipient agent is in a state in which it does not reply to certain requests
- Recipient agent deliberately delays the response (which may yield some advantage)
- Exceptions and errors in the recipient agent code disturb smooth execution
- Recipient agent ceases to exist after receiving the message
- Server hosting the recipient agent fails
- Network between sender and recipient experiences delays or high network load
- Network between sender and recipient fails.

10.3.4 Limited Exception Handling

Multiagent systems offer only limited exception handling capabilities. Communication in multiagent systems is conducted with asynchronous messaging among autonomous software agents. The first issue is that agent autonomy results in software agents that do not reveal full status information if it is not in their interests. Thus it cannot be assumed without limiting generality that software agents exchange status information with their communication partners. Secondly, asynchronous message-handling introduces a loose relationship between communication partners. In comparison, in synchronous communication (e.g. according to the client–server paradigm), sender and recipient fully synchronise their actions and exchange status and exception-handling information through an active connection when one software component calls an operation of another software component. This information is passed with a return parameter.

In multiagent systems the situation is different. A sender agent can discover whether its message has been added to the mailbox of a recipient agent. After all, this is a synchronous communication act between the sender and recipient agents’ mailboxes. Information about the success or failure of adding a message on time can be passed on with the return parameter of a remote procedure call. However, the sender agent has no means of handling exceptions with respect to the message content. First of all, there is no parameterisation of messages in agent orientation. No signature specifies exactly what
arguments a recipient agent expects and what concept, class or domain these arguments should be a member of. Moreover, the processing of a message is completely decoupled from adding the message to a software agent’s mailbox. Thus no information about the correctness of the content or exceptions during the processing of the message is available at the time of message delivery. Agents embody information hiding. Status information, exception messages, return parameters and results must be expressed and exchanged with additional messages on the interaction protocol level.

The agent-oriented paradigm does not provide for fully reliable exception handling. In the event of failure, a sender agent may be uncertain about the cause of the exception. The consequence is limited control at the system level and no guarantee of optimal processing in multiagent systems. Moreover, additional messages to enrich interaction protocols with some high level of exception handling result in an increased network load.

10.3.5 Adaptation to Failure

The flexibility of software agents and their ability to adapt to changes in their environment is often highlighted as one of the unique selling points of agent orientation. The basis for adaptation is formed by reasoning and decision-making capabilities. Based on the autonomy property, software agents independently react and act according to their local decisions. This decision-making is typically based on certain knowledge a software agent has about the behaviour of its communication partners.

A special case of adaptability – the adaptation to failure of collaboration partners and the resulting uncertainty – is typically not modelled. Failure of communication partners in short-term collaboration – limited to a particular interaction protocol – can be dealt with using message timeouts. After a timeout, a software agent can readjust itself to the new situation and decide on appropriate actions.

Failure in longer term collaborations is harder to deal with. If a relationship between two or more software agents reaches beyond a single interaction protocol, it may not be possible for communication partners to detect the liveliness of their collaboration partners. Consider the example depicted in Figure 10.16. A leader agent represents a group of member agents in negotiations with a service requester. Assume that the leader has informed the members that it has successfully placed a proposal with the requester (and
hence the members are waiting for the final contract confirmation). The leader then receives an acceptance message from the requester and replies with a confirm message before the associated message timeout. The leader must then pass on the agreed contract to the member agents. However, what happens if the leader agent fails to send the final message (perhaps for a reason listed in Section 10.3.3)? The member agents have no means of verifying whether a contract has been successfully created.

![UML sequence diagram of an example of adaptation to the failure of communication partners across long-term interaction protocols.](image)

**Figure 10.16:** UML sequence diagram of an example of adaptation to the failure of communication partners across long-term interaction protocols.

In general, a software agent cannot check the availability of another software agent. The only contact point of a software agent is its mailbox. But the availability of the mailbox does not allow any conclusions about the availability of the associated agent due to the loose coupling between an agent and its mailbox. If a software agent fails, e.g. due to an unhandled runtime exception, a deadlock situation or a server failure is completely transparent to the members.

The problem can be avoided by using a different set of interaction protocols that include the member agents in all interactions with the service requester. However, doing so may increase the network load and communication overhead significantly. It also increases the risk of concurrency and other temporal effects, as discussed in Section 10.2. Finally, it may not be possible to change the interaction protocols because of the organisational relationships and dependencies between different entities of the modelled problem domain.

In this context, viewing a software agent as a representative that acts on the behalf
of someone else is inadequate because there is no guarantee, without limiting generality, that a software agent can fulfill this role in a reliable manner. The impact of this problem is clear. It affects the robustness, reliability and efficiency of the agent paradigm. The coordination of multiple software agents in long-term relationships is a difficult problem without a general solution. Software agents can adapt to changes in their environments (events or failure of action). However, they have difficulties detecting failure in long-term relationships among multiple agents and reacting appropriately.

10.3.6 Balanced Behaviour

Padgham and Winikoff [130, p.2] stress that balanced execution of software agent behaviours is difficult but of utmost importance. If software agents are too reactive and adjust too easily to changes in environment, they may never achieve their goals. If software agents are not reactive enough and alter their beliefs only slowly, they may pursue plans that are no longer current.

In this thesis, a software agent reactively first processes all messages until its mailbox is empty before it performs proactive behaviour in order to trigger its own actions. In that sense this approach is settled at one extreme of the spectrum between very proactive (competitive) and very reactive (collaborative). Experiments with a balanced execution of proactive and reactive behaviours (iteratively always only one message followed by one proactive behaviour) have shown significant changes in the performance of the on-demand coalition formation process of this thesis. It notably reduced the performance and completeness of the approach by a factor of 4.

Padgham and Winikoff [130, p.9] suggest modelling the reactivity of software agents based on “significant changes in the environment” expressed with events. Unfortunately, in reasonably complex problem domains it is difficult to identify all relevant events. Moreover, agent self-adjustment does not seem to be a universally appropriate means. Its impact on system behaviour is unknown a priori. Therefore, it is difficult to control and it may lead to uncoordinated adjustments and unwanted system states. Determining a good balance between reactive and proactive behaviour is a very important issue for the success of multiagent systems. It has not been investigated sufficiently so far.
10. LESSONS LEARNT

10.4 Summary

This chapter discusses a number of issues that have been encountered during the development and execution of a multiagent system which implements the concept of on-demand service composition. The reason for choosing an agent-based approach in this thesis is the belief that agent orientation establishes “a genuine advance over the current state of the art for engineering complex systems” (Jennings [80], Jennings and Wooldridge [81]). However, this chapter highlights that the inherent complexity of multiagent systems (due to agent autonomy, real-time behaviour and concurrency) introduces problems with respect to their completeness, reliability and effectiveness. These issues counterbalance the natural agent-oriented modelling abilities and the relative simplicity of resulting static system designs (see Section 6.1). Further, this Chapter discusses a set of design, implementation and run-time issues beyond the empirical evaluation of the concept of on-demand service composition presented in Chapter 9. This chapter demonstrates that the software agent paradigm does not constitute a universal software engineering paradigm. Its application must be carefully chosen depending on the requirements of a specific problem domain. Even though some literature already explores the benefits and limitations of agent-based systems (e.g. Jennings [78, 79], Jennings and Wooldridge [82]), the development and characteristics of multiagent systems are yet not comprehensively understood (see Wooldridge and Jennings [170]). Accordingly, open issues of the software agent paradigm affect the development and performance of the on-demand service composition approach in open environments presented in this thesis, in sometimes unexpected ways, despite the inherent similarities between multiagent systems and the problem space of open environments (see Section 2.4.3).
Chapter 11

Conclusions

This chapter concludes this thesis by presenting its summary, an outline of main contributions and a discussion of prospective future work.

11.1 Summary

This thesis is an exploration of aspects of service-oriented computing (SOC). The core concept in SOC is service composition, that is, the combining of several services into a new composite service. This concept is central to this thesis. Even though it is the core concept in SOC, more research is required to instantiate it in open environments.

Current service composition solutions in industry are typically static, hard-coded and workflow-driven. They ensure full control over business processes at predictable costs, exhibited by a single organisational body. However, two ongoing trends – pervasive networking and the proliferation of computer systems in almost every aspect of modern life – drive the creation of new complex distributed software systems in open environments in which multiple organisations congregate to explore new business opportunities and to integrate and automate their business processes. These open environments are non-deterministic, partially observable and dynamic. Different organisations act in them as autonomous peers. Organisations do not share all information equally and they perform actions according to local decision-making that is based on potentially conflicting goals and preferences. No single peer can exhibit full control or gain full knowledge. Hence existing industry solutions are vulnerable in such environments to unforeseen dy-
dynamic changes largely due to their static and hard-coded nature. Thus more flexible and dynamic approaches are required. The central problem addressed in this thesis is the dynamic composition of services in open environments. The relevant notions of SOC, service composition and open environments are introduced in Chapter 2.

Dynamic service composition has already been intensively studied by the software agent community. Chapter 2 provides the most common definition of the term *software agent*. Based on a concrete interpretation of this definition, Chapter 2 further discusses a uniform software agent paradigm that firstly defines multiagent systems as systems that exhibit the characteristics of open environments, and secondly, utilises service-oriented design principles in support of a successful mapping of the problem of dynamic service composition onto the domain of collaborative problem-solving in multiagent systems. The hypothesis of this thesis is that the software agent paradigm provides appropriate means for developing an approach to dynamic service composition in open environments. However, a review of related agent-based approaches to dynamic service composition in Chapter 3 reveals several shortcomings. The motivation of this thesis is to develop an agent-based service composition approach that matches the characteristics of open environments.

Accordingly, Chapter 4 develops the novel concept of on-demand service composition that is tailored to the characteristics of open environments. Following this concept, Chapter 5 details a novel approach of on-demand coalition formation in open environments with a concrete technique for collaborative problem-solving in multiagent systems. Chapters 6 and 7 outline the static and dynamic design of a multiagent system and the two software agent types that implement the approach of on-demand coalition formation in open environments.

The last part of this thesis provides an evaluation of the presented approach, documented in three ways. Firstly, a demo application validates the application of the approach of this thesis in the domain of travel planning (Chapter 8). Secondly, an experimental evaluation analyses a number of key properties of the approach of on-demand coalition formation in open environments (Chapter 9). Thirdly, a discussion reports on selected issues that occurred during the design, implementation and operation of the approach of this thesis that affect the properties of the approach and also reflect on the software agent paradigm in general (Chapter 10).
11.2 Contributions

This thesis presents a novel approach to the problem of dynamic service composition in open environments in the form of on-demand coalition formation amongst autonomous self-interested software agents. It overcomes the shortcomings of existing agent-based approaches by offering a model of agent adaptation suited to open environments.

The contributions of this thesis are:

- A detailed analysis of a concrete interpretation of the weak notion of agency and the subsequent derivation of a software agent component definition that lays the foundation for a uniform agent-based software engineering approach. The resulting fundamental building blocks establish multiagent systems as open systems that exhibit the characteristics of open environments. They also utilise service-oriented design principles in support of a successful mapping of the problem of service composition in open environments into the software agent domain.

- The concept and design of a multiagent coalition formation framework that models on-demand service composition in open environments as collaborative problem-solving in an open market. In this open market, supply and demand confront each other as equal-righted autonomous peers in the form of service consumer agents and service provider agents. The design of the respective multiagent system is influenced by such concepts such as decentralisation, on-demand service provision, self-organisation, emergence and market-based control.

- The implementation of the multiagent coalition formation framework facilitates coalition formation among service provider agents with lightweight interaction protocols and decentralised multi-attribute utility-based decision-making based on client-defined quality-of-service (QoS) requirements and private preferences. It shapes an on-demand approach for dynamic service composition in open environments.

- A validation of the multiagent coalition formation framework is provided with a demo application for demonstrating dynamic service composition in the domain of travel planning.
11. CONCLUSIONS

- The experimental evaluation of the dynamic behaviour of the multiagent coalition formation process investigates non-functional properties such as termination, completeness, stability and scalability. It further identifies the conditions under which an optimal solution can be achieved despite the non-deterministic behaviour of interacting autonomous self-interested software agents.

- A critical analysis of design and implementation issues relating to the multiagent coalition formation framework documents inherent difficulties and limitations of the software agent paradigm when based on the weak notion of agency. These shortcomings stem from the weak notion’s complexity that is implied by the combination of characteristics of concurrent systems, real-time systems, asynchronous messaging, efficient message multicasting and local decision-making under uncertainty.

The main contribution of this thesis is the novel concept of on-demand service composition in open environments and its instantiation with a novel collaborative problem-solving technique. Composite services are created based on the decomposition of a user input using automated planning and the subsequent call for proposals that is broadcasted by a service consumer agent into the open market environment. Service provider agents then engage in interactions with the aim of forming coalitions in which service provider agents with complementary capabilities congregate to provide the requested composite service. The interactions between the autonomous self-interested service provider agents follow a set of principles in order to provide a model of agent adaptation to the characteristics of open environments: bottom-up decentralisation, on-demand structural organisation, self-organisation, emergence and market-based control. The decision-making of service provider agents is constrained by private preferences and by the client-defined QoS requirements of a composite service request.

The approach of this thesis clearly distinguishes it from existing agent-based approaches by its on-demand nature and the emergent behaviour of interacting autonomous self-interested software agents. The approach presented is fully tailored to the complex characteristics of open environments. Hence it allows the composition of services in such environments. Its strong adaptation to the characteristics of open environments also bears shortcomings due to the intrinsic complexity of such environments. In this respect, the
presented approach takes a similarly opportunistic approach, like comparable coalition formation approaches presented by Klusch and Gerber [93] and Soh and Tsatsoulis [154].

Table 11.1: Overview of the advantages and shortcomings of the approach of on-demand coalition formation in open environments.

<table>
<thead>
<tr>
<th>Advantage</th>
<th>Shortcoming</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adaptation to open environments</td>
<td>Limited control</td>
</tr>
<tr>
<td>Flexibility, extensibility</td>
<td>Incompleteness</td>
</tr>
<tr>
<td>Robustness</td>
<td>Redundancy, high communication costs</td>
</tr>
<tr>
<td>Local decision-making</td>
<td>No guarantee of optimality</td>
</tr>
<tr>
<td>Implications of the notion of agency</td>
<td></td>
</tr>
</tbody>
</table>

The approach of this thesis has been implemented and evaluated with a software prototype. The evaluation shows that the approach presented is adequate to facilitate dynamic service composition in open environments. It should be noted, however, that this approach is highly specialised and only addresses situations where the characteristics of a problem space do not allow for a deterministic solution. The key advantages and shortcomings of the approach are shown in Table 11.1 and summarised below.

**Adaptation to Open Environments**

The approach presented reflects the characteristics of open environments. It exhibits emergent behaviour that results in the dynamic creation of a composite service under the uncertain conditions of an open environment.

**Flexibility and Extensibility**

The approach presented does *not focus on service selection only*. It is able to adapt to a number of dynamic changes resulting from automated planning and dynamic decision-making under uncertainty. The flexible combination of sets of small composable plans enables a certain degree of adaptation to varying user-defined functional and QoS preferences. Local on-demand decision-making supports the adaptation to variations in the number and nature of available service providers and their services. Moreover, the set of
small composable plans offers opportunities for the flexible adaptation of the functionality of a solution by extension and reuse of existing plans.

Robustness

The approach presented reduces the risk of single points of failure. It does not incorporate a central repository, central decision-making and central data and control flows. Moreover, it does not require complete knowledge to be available to actors in the system. The approach presented reflects the partial observability, dynamics and non-determinism of open environments. The system behaviour is not based on pre-defined structures and dependencies. It emerges dynamically at run-time based on local decision-making performed under uncertainty.

Local Decision-making

The approach presented does not neglect private information and preferences. Service providers have full control over their involvement in the on-demand service composition process. They act according to local decision-making that is based on their private preferences as well as their knowledge about their environment. The concept of agent autonomy enables the protection of the actor’s interests.

Incompleteness

The approach presented may terminate without a result (see Section 9.2). The cause of this incompleteness is the inherent uncertainty in open environments implied by the autonomy property of the weak notion of agency. Software agents perform local decision-making based on private, potentially conflicting, preferences. Hence there is no guarantee that a set of autonomous agents will agree on a course of action, particularly because they typically do not fully reveal private information and thus perform decision-making under uncertainty. Even if the software agents agree, the emergence of appropriate structures depends on the interactions that are required for coordinating team actions (see Section 9.2). The number and nature of interactions is not known a priori (see Section 9.4).
11.2. CONTRIBUTIONS

No Guarantee of Optimal Outcomes

The approach presented cannot guarantee optimal outcomes: for service consumers or for service providers. The inherent uncertainty of open environments cannot be overcome. As shown in Section 9.3, local decision-making based on private preferences and information hinders the creation of a global optimum in many cases. However, Section 9.3 highlights that the approach presented delivers, on average, outcomes with a high probability of being better than outcomes of randomly formed coalitions. Moreover, outcomes in the vicinity of the optimal outcome are possible under specific conditions.

High Communication Costs

The approach presented creates a significant amount of network load due, firstly, to the use of broadcasting and multicasting in some interaction protocols, and secondly, to the extensive exchange of messages for creating coalition structures. Section 9.4 documents a quadratic dependency between communication costs and the number of communication partners. The market-based control mechanism of the open market concept has a positive impact on the stability of the presented approach (see Section 9.5). This means that local decision-making does not trigger software provider agents to switch between coalitions, which would significantly increase the communication costs. However, the quadratic trend holds only on average, which means that there may be extreme cases with significantly higher communication costs.

Redundancy

The approach presented introduces redundancy. This is due to local decision-making that results in concurrent computational efforts and communication activities of different software agents. Redundancy is a direct consequence of the complex characteristics of open environments and the autonomy property of the weak notion of agency.

Limited Control

The approach presented does not provide high levels of control for particular actors (e.g. service consumer). It does not guarantee cost and process control in comparison to static workflow-driven approaches. Cost and process control in this approach can be established
11. CONCLUSIONS

to a degree with SLAs imposed on contracted winning coalitions. However, contracts cannot be enforced on a technical level. Limited control is a direct result of the non-determinism of open environments in general and the emergent behaviour of the presented approach in particular.

Implications of the Notion of Agency

The shortcomings of the presented approach stem from the complex characteristics of open environments. They are further amplified by the complex properties of the weak notion of agency. This notion provides a natural means for modelling actors in an open environment (see Sections 2.4.2 and 2.4.3). It also introduces a number of difficulties that occur during the design, implementation and execution of a multiagent system. Firstly, the agent paradigm introduces problems, discussed in Section 10.1, that complicate the design of actors and the integration of legacy applications into multiagent systems. Secondly, the property of agent autonomy turns multiagent system into very complex software systems that combine characteristics and problems of the following concepts and principles:

- Concurrency (e.g. Sections 10.2.1, 10.2.5)
- Real-time constraints (e.g. Section 10.2.2)
- Asynchronous messaging (e.g. Section 10.2.2)
- Efficient message multicasting (e.g. Section 10.2.3)
- Local decision-making under uncertainty (e.g. Section 10.3).

The implications of the complexity of the notion of agency affect the performance of the approach of this thesis in an unforeseeable manner.

11.3 Future Work

This thesis focuses on the problem of dynamic service composition in open environments. Underlying this topic is the fundamental question of how software systems can be programmed in general so that they adapt to the complex characteristics of non-deterministic, partially observable, dynamic and thus uncertain problem spaces in which no single entity
can exhibit full control. Accordingly, prospective future research includes the design and implementation of the minimal – necessary and sufficient – set of rules and controls that must be in place to ensure results that are, with a certain probability, in a required range. Areas of future research include the:

- Design, implementation and verification of interaction protocols (e.g. considering concurrency and real-time issues)
- Design and implementation of message timeouts (e.g. static vs dynamic) and their impact on agent and multiagent system behaviour
- Balanced execution of proactive and reactive components of a software agent and its impact on agent and multiagent system behaviour
- Influence of network characteristics (e.g. delays, outages, topologies) on the local decision-making and global multiagent system behaviour
- Adaptation of software agents to failure and uncertainty about the internal states of other software agents
- Improvement of practical reasoning and decision-making techniques.

Advances in any of the above areas may improve the performance of the presented approach.

Further, future research focusing on relaxing some of the assumptions specified in Section 4.4 could improve and extend the approach presented in this thesis. A number of key issues are:

- Further experimentation in order to better understand the full effect of the characteristics of open environments and the corresponding influencing factors of the presented approach may lead to the identification of further means for improving its performance (e.g. facilitate the creation of more complete outcomes, achieve outcomes with higher utility for the service consumer agent and reduce the number of exchanged messages).
- An area of research is the extension of the current approach to incorporate trust and reputation measures that help to detect and prevent malicious agent behaviour.
11. CONCLUSIONS

Long-term trust and reputation records (over many composite service requests) in combination with automated learning techniques could be explored, and especially their impact on overall system performance and the optimality of outcomes.

- The current approach assumes single stateless services. However, in reality services typically exhibit complex message-exchange patterns. Accordingly, a conversational agent-based approach could be researched that would enable two or more software agents to determine whether they are interoperable in this regard by exchanging and verifying relevant information.

- The current approach assumes that software agents will only join one coalition at a time. However, this assumption is difficult to enforce in open environments. Thus the investigation of software agents joining multiple coalitions would be useful to understand their impact on the system behaviour.

- The current Composite Service Request interaction protocol (see Section 6.4.1) may be extended into an iterative Contract Net Protocol that facilitates a negotiation process between a service consumer agent and coalition leaders. This extension would also imply additional coalition-internal negotiations.

- The impact that variable service advertisements have during a single coalition formation process should be investigated. In the approach presented, service provider agents disseminate only a fixed service advertisement at the beginning of a service composition process.
References


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


REFERENCES


[81] Nicholas R. Jennings and Michael J. Wooldridge. Agent-Oriented Software Engineering. The article was planned to appear in the Handbook of Agent Technology that has never been published, 2000. available online at http://citeseer.ist.psu.edu/242172.html. last visited January 2009.


REFERENCES


REFERENCES


REFERENCES


[116] Erick Martínez and Yves Lespérance. Web Service Composition as a Planning
REFERENCES


REFERENCES


[140] Chris Preist, Claudio Bartolini, and Andrew Byde. Agent-Based Service Composition Through Simultaneous Negotiation in Forward and Reverse Auctions. In
REFERENCES


REFERENCES


REFERENCES


REFERENCES

Appendix
Appendix A

AUML Quick Reference

This section provides a quick reference of the AUML interaction diagram notation. Figure A.1 presents the structure and key elements of AUML interaction diagrams that are used in this thesis. The reader is referred to Huget et al. [74] for the complete specification of the notation.

AUML interaction diagrams bases on the notation of UML sequence diagrams. One interaction diagram specifies exactly one interaction protocol. An AUML interaction diagram is presented in a framed box that has an inlet in the top left corner which states the name of the interaction protocol prefixed by the keyword “sd” (compare Section I in Figure A.1).

The definition of a software agent is shown in Section II of Figure A.1. A software agent is defined with a box that contains a unique identifier, a particular role the software agent may act in, and an agent type. All of these attributes are optional. That means, an agent box may also represent multiple software agent instances or a generic class of a software agent. The associated cardinality is provided with a number in the top right corner of the agent box. An agent box has a lifeline attached. A lifeline is a vertical dashed line.

Only asynchronous messages are used within this thesis. Their specification is described in Section III of Figure A.1. A message is represented by a solid line with an open arrow head (a). The line is annotated with a message id and an ACL performative ([52]) (b). A message always has exactly one sender. It may have various recipients, expressed with a cardinality tag next to the arrow head (b). A cardinality in AUML interaction
A. AUML QUICK REFERENCE

Diagrams is defined either with a numeral, a variable, or an expression including numerals, variables, and relation symbols such as $<,>,\leq,\geq,=$. This convention applies for messages as well as agents. If no cardinality is given, the default value of 1 is assumed. Messages may also be specified with a timeout $(c)$. Message timeouts are always used in blocking mode. That means an agent stalls processing until a reply is received or the timeout is due. The timeout delay can be optionally specified in curly brackets.

<table>
<thead>
<tr>
<th>I) diagram definition</th>
<th>II) agent definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Diagram" /></td>
<td><img src="image" alt="Agent" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>III) message definition</th>
<th>IV) control construct definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>a) Asynchronous message</td>
<td>d) synchronous</td>
</tr>
<tr>
<td>b) Message identifier</td>
<td>e) alternative</td>
</tr>
<tr>
<td>c) Asynchronous message</td>
<td>f) parallel</td>
</tr>
<tr>
<td>d) with blocking mode timeout</td>
<td>g) coalition formation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>V) additional definitions</th>
<th>l) termination symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>h) Action name</td>
<td></td>
</tr>
<tr>
<td>i) Condition</td>
<td></td>
</tr>
</tbody>
</table>

**Figure A.1:** Overview of the key elements of the AUML interaction diagram notation.
Section IV of Figure A.1 depicts the control constructs that are used in this thesis.
The option construct (d) contains a block of messages, actions, etc. that are executed if
an initial condition is fulfilled. The initial condition is stated in square brackets at the top
of the box. The alternative construct (e) compares to a choice or case control construct.
A number of mutually excluding conditions guard the execution of different blocks. Two
different blocks are separated by a dashed line. The conditions are stated in square
brackets at the top of every block. The parallel construct (f) separates different blocks
that are executed concurrently. Different blocks are separated by dashed lines. AUML also
enables the modelling of nested interaction protocols. An embedded interaction protocol
(g) is specified with a box that includes an inlet (carrying the keyword “ref”) in the top
left corner and that states the name of the embedded interaction protocol in the centre.

There is a number of additional definitions. (h) shows the graphical specification
of modelling a software agent that changes roles. A dashed line with open arrow head
connects two lifelines of two different agent roles pointing to the role to be assumed. The
arrow is annotated with the stereotype $<< \text{change role} >>$. (i) presents the graphical
notation of an action. It is depicted with an oval box with the action’s name inside. An action can be attached to a lifeline of an agent with a solid line. A sequence of
actions may be modelled with a sequence of concatenated action boxes. (k) shows how
adjacent interaction protocols can be aligned. A message arrow is drawn from a lifeline
to outside the box of an AUML interaction diagram. The adjacent sequence diagram has
a complementary incoming message arrow that connects from outside its box to a lifeline.
(l) refers to the termination symbol which marks the end of any activities with regard to
an interaction protocol for a single lifeline. Hence, it does not terminate the lifeline itself.
(j) presents the modelling of conditions. A condition is modelled with square brackets.
A condition is a statement in natural language. Multiple of them can be combined into
logical sentences using the following relations: negation $\neg$, conjunction $\&$, disjunction $\mid$, and parentheses $(\). Conditions are applied as following. Firstly, a condition specifies the
initial condition for executing all elements in a block of a control construct (Section IV of
Figure A.1). If a condition is used not in the context of a control construct, it refers to
exactly one single message, action, etc. In this case, the condition is placed right on top
of the element it refers to.
Appendix B

List of Publications

The following list presents all main/co-authored publications as well as edited volumes prepared by the candidate that have been published during the PhD candidature with respect to the research project and related topics.


- Ingo Müller, Ryszard Kowalczyk: Service Composition through Agent-based Coalition Formation. In Proceedings of the Workshop on WWW Service Composition with Semantic Web Services (WSCOMP 2005), held in conjunction with the


• Ingo Müller, Peter Braun, Ryszard Kowalczyk. A Classification Scheme for the Integration of Software Agent and Service-oriented Paradigms. In Proceedings of the Workshop on Service-Oriented Computing and Agent-Based Engineering (SOCABE 2005), held in conjunction with the Fourth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2005), pages 57–60, Utrecht, the Netherlands, 2005.
