A Framework for Social Context-Aware
Pervasive Computing Applications

Faculty of Information and Communication Technologies
Swinburne University of Technology
Melbourne, Australia

Submitted for the degree of Doctor of Philosophy

Muhammad Ashad Kabir

2013
Abstract

Pervasive computing envisions “smart” places where people are assisted in their daily life by various technologies that move to the background, being invisible yet significantly helpful. To provide such technologies a rapidly growing research area called “context-awareness” has emerged, to make pervasive applications more intelligent and accessible. Humans, the centre of pervasive computing, however, are social beings. Hence, the notion of social context-awareness can advance the field further in achieving the vision of pervasive computing. As such, recently a vibrant research direction is introduced, entitled “Pervasive Social Computing”, which is a novel collective paradigm derived from pervasive computing, social media, social networking, social signal processing and multimodal human-computer interaction. It aims to take advantage of human social context to enable the fulfilment of users’ task on the move, ultimately promoting individual well being and social productivity. With the increasing prevalence and features of advanced mobile devices, interest has grown in socially-aware mobile applications, where the applications will be aware of users’ social context and be able to assist them in their daily activities.

Developing socially-aware applications is challenging. First, the socially-aware applications need different types of social context information such as social roles, social relationships, social interactions and situations. These different types of social context information need to be modelled, represented and acquired from various sources for applications’ use. Second, an application may need social context information that is not directly available from context sources but can be inferred from basic information. Thus, there is the need to define and obtain inferred social context information from the acquired basic information. Third, the users’ social context information is inherently sensitive and can be used to infer further information. The users should be allowed to specify their privacy preferences regarding access to their information. Fourth, much of users’ social context information is dynamic and continuously changes. The socially-aware applications need to support adaptation to cope with the changes in social context information.

This thesis introduces a novel framework to address the aforementioned challenges in developing socially-aware applications. The framework includes an approach to model, represent, reason about, and manage different types of social context, and a platform for acquiring, storing, provisioning and managing social context information to aid the development of socially-aware applications. This thesis makes several important research contributions. Firstly, the proposed framework provides an ontology-based social context model, including an upper social context ontology which can be customised and specialised to form domain-specific ontologies. Secondly, the
framework provides an ontology-based approach to define rules for deriving, deducing and abstracting social context information. It also propose an approach to inferring users’ situations by correlating historical interaction events. **Thirdly,** the framework provides an ontology-based socially-aware access control policy model and language for owners to control access to their information at different levels of granularity. **Fourthly,** the framework also provides a way to model and represent runtime social interactions from both the domain- and player-perspectives to support mediated social interactions, and manage their adaptation to cope with the changes in user requirements and environments. **Finally,** the framework provides a platform for acquiring, storing and provisioning social context information, executing social interaction models and managing their runtime adaptation.

This thesis presents two case studies that validate and evaluate the different functionalities of the framework and its underlying approach and platform. The first case study demonstrates the development of a socially-aware phone call application that validates the framework’s functionality of modelling and inferring social roles, connection-oriented social relationships and situations, acquiring such information from different online social networks (OSNs), and preserving privacy in accessing information. In this regard, we also evaluate the platform’s performance using real data from OSNs. The second case study demonstrates the development of a socially-aware telematics application that supports social interactions in a cooperative convoy based on interaction-oriented social relationships. This application validates the framework’s functionality of modelling runtime social interactions, managing their runtime adaptation, and inferring users’ situations from their interaction events. In this regard, we also evaluate the platform’s adaptation overhead and performance in inferring situations.

Overall, this research has provided a number of techniques and a platform for developing different kinds of innovative socially-aware applications. The techniques allow developers to represent social context, while the platform hides the complexity of managing social context, and thus provides support for the better development of socially-aware applications.
Acknowledgements

This thesis would have never been finished without the help of many, to whom I would like to express my sincere thanks. All praise be to Allah for providing me the strengths and blessings to complete this thesis.

I would like to express my sincere gratitude to my principal supervisor Professor Jun Han, for his exceptional support, patience and guidance to the completion of this thesis. My sincere thanks also go to my associate supervisors Dr. Alan Colman and Dr. Jian Yu for their generous guidance. Thank you all for the many hours spent on reviewing the papers and thesis drafts. I would also like to show my appreciation to my research progress review panels Professor Chengfei Liu, Dr. Rajesh Vasa and Dr. Jinjun Chen, and external reviewers for their feedbacks and comments. In addition, I thankfully acknowledge the many fruitful discussions with Dr. Adeel Talib and Dr. Minh Tran.

My thanks go to my fellow colleagues Dr. Malinda Kapuruge, Dr. Tharindu Patikirikorala, Dr. Tuan Nguyen, Ayman Amin, Mahmoud Hussein, Indika Kumara, ASM Kayes, Dr. Samiran Mahmud, Dr. Shimul Nath, Shibli Saleheen, Arup Kumar and Monir Uddin Ahmed for the numerous productive discussions that influenced the outcome of this thesis. Thank you all who had assisted in prototype implementation and volunteered in my experiments. I am also thankful to the awarding committee and the staff for their continuous assistance.

This project would not have been possible without the scholarships provided by the Commonwealth Government and Swinburne University of Technology. I would also like to acknowledge the top-up and travel funding scholarships provided by the AutoCRC.

I am indebted to my family members, especially to my mother for her never-ending compassion and every word of encouragement amidst numerous hardships. This thesis is all dedicated to you!

Finally, I thank everybody who was important to the successful completion of this thesis with an apology for not mentioning by name.
Declaration

This is to certify that,

- this thesis contains no material which has been accepted for the award of any other degree or diploma, except where due reference is made in the text of the examinable outcome; and

- to the best of my knowledge contains no material previously published or written by another person except where due reference is made in the text of the thesis; and

- where the work is based on joint research or publications, discloses the relative contributions of the respective workers or authors.

Muhammad Ashad Kabir
October, 2013
Melbourne, Australia
List of Publications

Journals


Conference Proceedings

• M. A. Kabir, J. Han, J. Yu, and A. Colman, SCaaS: A Platform for Managing Adaptation in Collaborative Pervasive Applications, 21st International Conference on Cooperative Information Systems (CoopIS 2013), Graz, Austria, Sep 11-13, 2013, LNCS, Springer, pp. 149-166. doi:10.1007/978-3-642-41030-7_10


Posters and Demos


# Contents

- **Abstract** iii
- **Acknowledgements** v
- **Declaration** vii
- **List of Publications** viii
- **List of Figures** xvii
- **List of Tables** xxi

## Part I

1. **Introduction** 3
   1.1 Pervasive Social Computing 3
   1.2 Defining Social Context 5
   1.3 Research Challenges 8
      1.3.1 Consistent Representation of Social Context Information 9
      1.3.2 Reasoning about Meaningful Social Context Information 10
      1.3.3 Managing Access Control and Adaptation 11
   1.4 Approach Overview 11
      1.4.1 Social Context Modelling 12
      1.4.2 Social Context Reasoning 13
      1.4.3 Social Context Management 13
   1.5 Thesis Statement and Contributions 14
   1.6 Thesis Structure 15

2. **Application Scenarios** 19
   2.1 A Data-centric Socially-Aware Application Scenario 20
      2.1.1 The Phone Call Interruptions 20
      2.1.2 Requirements Analysis 21
   2.2 An Interaction-centric Socially-Aware Application Scenario 22
      2.2.1 The Cooperative Convoy 23
      2.2.2 Requirements Analysis 24
   2.3 Requirements for Developing Socially-Aware Applications 26
   2.4 Summary 28

3. **Background and Related Work** 31
### Contents

6.1 Managing Access to Social Context Information ............................................. 113
   6.1.1 A Socially-Aware Access Control Policy Model .................................. 114
   6.1.2 Policy Specification and Inconsistency Detection ................................. 116
   6.1.3 Policy Enforcement in Accessing Social Context Information ............... 119
6.2 Modelling and Managing Social Interactions ................................................. 120
   6.2.1 Modelling Social Interactions ............................................................ 121
      6.2.1.1 Domain-centric Social Interaction Model ................................... 122
      6.2.1.2 Player-centric Social Interaction Model ...................................... 131
      6.2.1.3 Mediating Runtime Social Interactions ....................................... 136
   6.2.2 Managing Runtime Adaptation ............................................................. 136
      6.2.2.1 Types of Changes ................................................................. 137
      6.2.2.2 Adaptation in a Social Interaction Model ................................... 138
      6.2.2.3 Management Operations and Adaptation Rules .............................. 139
      6.2.2.4 Social Interaction State and Safe Change .................................... 142
   6.2.3 Adaptation across Social Interaction Models ......................................... 143
6.3 Summary ............................................................................................................ 147

Part III ................................................. 149

7 SocioPlatform Architecture and Prototype Implementation ......................... 151
   7.1 Architecture and Design of the SCIM ......................................................... 153
      7.1.1 Information Acquisition Interface ...................................................... 154
      7.1.2 Designing Adapters to Fetch Raw Data ........................................... 155
      7.1.3 Fetching and Processing Raw Data for Consistent Social Context Information ...................................................... 156
      7.1.4 Policy Interface .................................................................................... 157
      7.1.5 Query Interface .................................................................................... 158
      7.1.6 Rule Interface ....................................................................................... 160
   7.2 SCIM Prototype Implementation ................................................................. 161
      7.2.1 Implementing Ontologies ....................................................................... 161
      7.2.2 Implementing Adapters .......................................................................... 165
      7.2.3 Implementing Rule APIs and Reasoning Module ................................... 166
      7.2.4 Implementing Policy APIs and Privacy Policy Module ......................... 169
      7.2.5 Implementing Query APIs and Query Processing Module .................... 170
   7.3 Architecture and Design of the SIM ............................................................. 171
      7.3.1 The Model Execution Module ............................................................... 173
      7.3.2 The Management Module ..................................................................... 174
   7.4 SIM Prototype Implementation ................................................................. 174
      7.4.1 Implementing the Model Execution Module ......................................... 177
      7.4.2 Implementing the Management Module ................................................ 181
      7.4.3 The Deployment Layer ....................................................................... 182
      7.4.4 Composing Runtime Social Interaction Models using SIMbuilder .......... 184
      7.4.5 Generating and Providing Interaction Events ....................................... 185
   7.5 Summary ....................................................................................................... 187

8 Case Studies: Social Context-Aware Applications ........................................... 189
   8.1 Design and Implementation Process ............................................................. 191
   8.2 Case Study #1: The SPCell Application ....................................................... 192
      8.2.1 Design Goals and Constraint ............................................................... 193
## CONTENTS

### A Description Logic-based Reasoning Rules and Constraints
- A.1 Classification and Reasoning Rules ........................................... 275
- A.2 Constraints ............................................................................. 281
  - A.2.1 Constraints Defined in SACOnto ....................................... 282

### B Social Interactions in a Cooperative Convoy .......................... 283

### C Social Interaction Models ...................................................... 291
  - C.1 Domain-centric Social Interaction Models (DSIMs) ............... 291
    - C.1.1 ConvoyDSIM .................................................................. 291
    - C.1.2 BudgetDSIM .................................................................. 293
    - C.1.3 AVISDSIM .................................................................... 295
  - C.2 Player-centric Social Interaction Models (PSIMs) ................. 298
    - C.2.1 Car#1PSIM .................................................................. 298
    - C.2.2 Car#2PSIM .................................................................. 301

### D Rule Translation Script (*RuleTrans.dsl*) ............................... 305

### E Schema Definitions ............................................................... 309
  - E.1 DSIM Schema ...................................................................... 309
  - E.2 PSIM Schema ...................................................................... 313
List of Figures

1.1 SocioPlatform overview ........................................... 12

2.1 Illustrations of phone call interruptions [198] .......................... 20
2.2 An illustration of cooperative convoy ................................. 23

3.1 Towards a framework for social-context aware applications .......... 53

4.1 Illustration of granularity levels ...................................... 62
4.2 Hierarchies of the core concepts .................................... 65
4.3 Social context upper ontology (SCOnto) ............................... 66
4.4 An excerpt of domain-specific social role ontology .................. 69
4.5 A domain-specific people-centric relationship ontology .......... 71
4.6 Identifying family members in Facebook ................................ 72
4.7 A family relationship ontology ....................................... 73
4.8 A friend relationship ontology ........................................ 75
4.9 Facebook recognises close friend relationship ....................... 75
4.10 A work relationship ontology ......................................... 76
4.11 An object-centric relationship ontology .............................. 77
4.12 An interaction-oriented social relationships ontology for cooperative convoy 78
4.13 A social interaction ontology ....................................... 80
4.14 A situation ontology .................................................. 83

5.1 An inferred friend relationship ontology .............................. 89
5.2 An excerpt of situation ontology ....................................... 91
5.3 An excerpt of owl:Time ontology ....................................... 92
5.4 An inferred owl:Time ontology ....................................... 92
5.5 Classifying the time instant (May 20 2013 Monday 2:30pm) to domain-specific concepts ............................................. 94
5.6 Deducing the individual meeting situation from the time instant (May 20 2013 Monday 2:30pm) ........................................ 96
5.7 An inferred business situation ontology ............................... 97
5.8 Interaction event upper ontology (IntEO) .............................. 100
5.9 InteractionType, topic and Time ontologies for the cooperative convoy application .................................................. 102
5.10 An InteractionType ontology ......................................... 105

6.1 Access control policy model .......................................... 115
6.2 Policy specification steps ............................................... 117
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.3</td>
<td>Meta-model of DSIM</td>
<td>123</td>
</tr>
<tr>
<td>6.4</td>
<td>Graphical notations for representing social interaction model</td>
<td>123</td>
</tr>
<tr>
<td>6.5</td>
<td>Separation of functional and management aspects</td>
<td>124</td>
</tr>
<tr>
<td>6.6</td>
<td>A template for the domain-centric social interaction model (DSIM)</td>
<td>126</td>
</tr>
<tr>
<td>6.7</td>
<td>Budget domain-centric social interaction model (BudgetDSIM)</td>
<td>127</td>
</tr>
<tr>
<td>6.8</td>
<td>A partial description of R2 relationship in BudgetDSIM</td>
<td>128</td>
</tr>
<tr>
<td>6.9</td>
<td>AVIS domain-centric social interaction model (AVISDSIM)</td>
<td>129</td>
</tr>
<tr>
<td>6.10</td>
<td>Convoy domain-centric social interaction model (ConvoyDSIM)</td>
<td>129</td>
</tr>
<tr>
<td>6.11</td>
<td>A partial description of the convoy domain-centric social interaction model (ConvoyDSIM)</td>
<td>130</td>
</tr>
<tr>
<td>6.12</td>
<td>Meta-model of PSIM</td>
<td>132</td>
</tr>
<tr>
<td>6.13</td>
<td>Template for the player-centric social interaction model (PSIM)</td>
<td>133</td>
</tr>
<tr>
<td>6.14</td>
<td>Player-centric social interaction models of Car#1 and Car#2</td>
<td>135</td>
</tr>
<tr>
<td>6.15</td>
<td>Message based interaction via a runtime social interaction model</td>
<td>137</td>
</tr>
<tr>
<td>6.16</td>
<td>Externalisation of users’ interaction-relationships from applications</td>
<td>137</td>
</tr>
<tr>
<td>6.17</td>
<td>Organiser player and adaptation</td>
<td>140</td>
</tr>
<tr>
<td>6.18</td>
<td>State machine for Social Role, Social Relationship and SIM Composite</td>
<td>142</td>
</tr>
<tr>
<td>6.19</td>
<td>Cross-DSIM/PSIM adaptation propagation</td>
<td>143</td>
</tr>
<tr>
<td>7.1</td>
<td>SocioPlatform overview</td>
<td>151</td>
</tr>
<tr>
<td>7.2</td>
<td>Architecture of the social context information management (SCIM)</td>
<td>153</td>
</tr>
<tr>
<td>7.3</td>
<td>A screenshot of SCOnto in Protege</td>
<td>162</td>
</tr>
<tr>
<td>7.4</td>
<td>A screenshot of SACOnto and an example of formatting policy (Policy#1)</td>
<td>163</td>
</tr>
<tr>
<td>7.5</td>
<td>A screenshot of IntEO in Protege</td>
<td>164</td>
</tr>
<tr>
<td>7.6</td>
<td>Applications for getting authorisation from users to fetch their social data</td>
<td>165</td>
</tr>
<tr>
<td>7.7</td>
<td>Reasoning module implementation</td>
<td>166</td>
</tr>
<tr>
<td>7.8</td>
<td>The Drools executable format of the Watching game situation rule</td>
<td>168</td>
</tr>
<tr>
<td>7.9</td>
<td>Implementation of the semantic matching function</td>
<td>169</td>
</tr>
<tr>
<td>7.10</td>
<td>A Java code snapshot for the addPolicy function</td>
<td>170</td>
</tr>
<tr>
<td>7.11</td>
<td>A Java code snapshot in Eclipse that implements getSituationGranularity(A,gLevel) operation</td>
<td>172</td>
</tr>
<tr>
<td>7.12</td>
<td>Architecture of the social interaction management (SIM) platform</td>
<td>172</td>
</tr>
<tr>
<td>7.13</td>
<td>Layers of the social interaction management (SIM) prototype</td>
<td>175</td>
</tr>
<tr>
<td>7.14</td>
<td>Class Diagram: Model Execution Module implementation</td>
<td>178</td>
</tr>
<tr>
<td>7.15</td>
<td>Marshalling and Unmarshalling via JAXB 2.0 Bindings</td>
<td>179</td>
</tr>
<tr>
<td>7.16</td>
<td>A Drools rule for the obligation of LeadingCar</td>
<td>181</td>
</tr>
<tr>
<td>7.17</td>
<td>Class Diagram: Management Module implementation</td>
<td>182</td>
</tr>
<tr>
<td>7.18</td>
<td>LeadingCar social role (in ConvoyDSIM) related interactions are exposed</td>
<td>183</td>
</tr>
<tr>
<td>7.19</td>
<td>Organiser (management) role (in ConvoyDSIM) operations are exposed as</td>
<td>184</td>
</tr>
<tr>
<td>7.20</td>
<td>Steps for composing DSIMs and PSIMs</td>
<td>185</td>
</tr>
<tr>
<td>7.21</td>
<td>SIMbuilder tool for composing DSIMs and PSIMs</td>
<td>186</td>
</tr>
<tr>
<td>7.22</td>
<td>Interaction events flow from SIM to SCIM</td>
<td>187</td>
</tr>
<tr>
<td>8.1</td>
<td>Design and implementation process for socially-aware applications</td>
<td>191</td>
</tr>
<tr>
<td>8.2</td>
<td>A excerpt of the extended SCOnto for SPCall application</td>
<td>196</td>
</tr>
<tr>
<td>8.3</td>
<td>A excerpt of the extended IntEO for SPCall application</td>
<td>197</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

8.4 Screen shots of the SPCall client application for Android ............. 198
8.5 SocioPlatform acquires social context information from OSNs ......... 200
8.6 An excerpt of the extended SConTo for SocioTelematics case study ....... 208
8.7 An excerpt of the extended IntEO for the cooperative convoy .......... 211
8.8 Some screenshots of the SocioTelematics application for Android ....... 214
8.9 A screen-shot of a coordination rule specified in SocioTelematics application 215
8.10 An illustration of the runtime adaptation in the ConvoyDSIM and associated PSIMs to cope with changes in user requirements and environments 218

9.1 Time taken by the Information acquisition module in SCIMS to acquire social context information from Facebook .................. 224
9.2 Reasoning time for various sizes of KBs using five different reasoners (i.e., HermiT, Pellet, RacerPro, TrOWL and Fact++) ......... 228
9.3 SCIMS’s performance on query processing (QP) – time taken to answer a set of queries (isAFamilyOfBE, getAllRelationshipsName and getSituAtGranularity) for various sizes of KBs using five different reasoners (i.e., HermiT, Pellet, RacerPro, TrOWL and Fact+++) ............... 228
9.4 Performance evaluation results for a fixed size (1000) of interaction event stream (IES) with increasing situation specification rule (SSR) set sizes and event firing occurrences ................. 230
9.5 Performance evaluation results for a fixed size (1000) SSR set and a fixed number (1000) of detected situations with increasing sizes of IESs ........ 231
9.6 Adaptation in a runtime social interaction model ....................... 232
9.7 Adaptation across social interaction models caused by binding and unbinding role player structural adaptation ....................... 234
9.8 SIM’s resource consumption and performance related to mediating social interactions .................................................. 235
9.9 SIM’s performance results are illustrated using the box plots - the left figure depicts the most extreme value in outliers, the right figure (zoom in view of the left figure) highlights the average processing time .......... 237
9.10 Client-server architecture of the SocioTelematics application .......... 237
9.11 Resource consumption of the SocioTelematics while it was fetching messages from the SIM in every three seconds, updating Google maps view in every five seconds, and sending messages (i.e., performing interactions) once in every second ........................................ 240
9.12 Communication latency ................................................. 241
LIST OF FIGURES
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>OWL DL descriptions, data ranges, properties, individuals and data values</td>
<td>38</td>
</tr>
<tr>
<td>3.2</td>
<td>OWL DL axioms and facts</td>
<td>39</td>
</tr>
<tr>
<td>3.3</td>
<td>Comparison of the different context modelling approaches</td>
<td>43</td>
</tr>
<tr>
<td>3.4</td>
<td>The summary of evaluation of the examined approaches</td>
<td>55</td>
</tr>
<tr>
<td>4.1</td>
<td>OWL-DL class constructors</td>
<td>60</td>
</tr>
<tr>
<td>4.2</td>
<td>OWL-DL class semantics</td>
<td>61</td>
</tr>
<tr>
<td>4.3</td>
<td>OWL object property characteristics</td>
<td>63</td>
</tr>
<tr>
<td>4.4</td>
<td>Interaction (i1): Leading car notifies its position to the following car</td>
<td>79</td>
</tr>
<tr>
<td>4.5</td>
<td>Interaction (i2): Following car notifies its position to the leading car</td>
<td>81</td>
</tr>
<tr>
<td>4.6</td>
<td>Obligation (o1) and related Operational parameter (p1): Both the leading</td>
<td>81</td>
</tr>
<tr>
<td></td>
<td>and following cars notify each other of their positions every 10 seconds</td>
<td></td>
</tr>
<tr>
<td>6.1</td>
<td>Operations provided by the <code>organiser role</code> to manipulate structure of the</td>
<td>140</td>
</tr>
<tr>
<td></td>
<td>runtime model</td>
<td></td>
</tr>
<tr>
<td>6.2</td>
<td>Operations provided by the <code>organiser role</code> to manipulate parameter of the</td>
<td>141</td>
</tr>
<tr>
<td></td>
<td>runtime model</td>
<td></td>
</tr>
<tr>
<td>6.3</td>
<td>Operations provided by the <code>organiser role</code> to manipulate <code>state</code></td>
<td>141</td>
</tr>
<tr>
<td>6.4</td>
<td>Operations provided by the <code>organiser role</code> to <code>monitor</code> a runtime model by</td>
<td>141</td>
</tr>
<tr>
<td></td>
<td>acquiring information</td>
<td></td>
</tr>
<tr>
<td>7.1</td>
<td>Information acquisition APIs for accumulating social context information</td>
<td>155</td>
</tr>
<tr>
<td>7.2</td>
<td>Policy APIs for managing users’ privacy policies</td>
<td>157</td>
</tr>
<tr>
<td>7.3</td>
<td>Query APIs for accessing social context information</td>
<td>159</td>
</tr>
<tr>
<td>7.4</td>
<td>Query APIs for accessing meta-context information</td>
<td>160</td>
</tr>
<tr>
<td>7.5</td>
<td>Rule APIs for managing reasoning rules</td>
<td>160</td>
</tr>
<tr>
<td>7.6</td>
<td>Platform level management APIs</td>
<td>175</td>
</tr>
<tr>
<td>8.1</td>
<td>Tom is Alice’s graduate school friend (Turtle syntax)</td>
<td>201</td>
</tr>
<tr>
<td>8.2</td>
<td>Jack is Alice’s Father (Turtle syntax)</td>
<td>201</td>
</tr>
<tr>
<td>8.3</td>
<td>Bob is Alice’s Colleague (Turtle syntax)</td>
<td>201</td>
</tr>
<tr>
<td>8.4</td>
<td>John is Alice’s Supervisor (Turtle syntax)</td>
<td>201</td>
</tr>
<tr>
<td>8.5</td>
<td>Tom is Alice’s friend as the education-based friend is a sub-class of friend</td>
<td>202</td>
</tr>
<tr>
<td></td>
<td>(Turtle syntax)</td>
<td></td>
</tr>
<tr>
<td>8.6</td>
<td>The relationship between Jack and Alice is Father-Daughter which is a kind of</td>
<td>202</td>
</tr>
<tr>
<td></td>
<td>Family relationship (Turtle syntax)</td>
<td></td>
</tr>
</tbody>
</table>
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.7</td>
<td>A Social Interaction (si1) that consists of two basic Interactions (i7 and i8), a Conversation (c1), a Obligation (o3) and an Operational Parameter (p3). The social interaction can be read as – upon receiving a route request from a driver of a rented car, a travel guide service should send a route notification in 15 seconds.</td>
<td>210</td>
</tr>
<tr>
<td>9.1</td>
<td>Social context information collected from users’ Facebook accounts</td>
<td>225</td>
</tr>
<tr>
<td>9.2</td>
<td>Knowledge bases (KBs) have built for experimental subjects</td>
<td>227</td>
</tr>
<tr>
<td>9.3</td>
<td>SocioTelematics vs. benchmark applications – CPU, Memory and Storage usage</td>
<td>239</td>
</tr>
<tr>
<td>9.4</td>
<td>Time required to perform adaptation operations</td>
<td>242</td>
</tr>
<tr>
<td>9.5</td>
<td>SPcall vs. benchmark applications – CPU, Memory and Storage usage</td>
<td>243</td>
</tr>
<tr>
<td>9.6</td>
<td>Call filtering performance of the SPcall application</td>
<td>245</td>
</tr>
</tbody>
</table>
Part I
Chapter 1

Introduction

“The most profound technologies are those that disappear. They wave themselves into the fabric of everyday life until they are indistinguishable from it” (Mark Weiser) [187].

After introducing the vision of ubiquitous computing, also called pervasive computing, by Mark Weiser in 1991, over the past two decades, this field evolved through roughly three generations of research challenges. Now, the scientific community articulates next-generation research directions as the quest to attain Weiser’s vision continues [77].

1.1 Pervasive Social Computing

Pervasive computing envisions environments where people are assisted in their everyday tasks by technologies that recede into the background, being invisible yet significantly helpful [163]. To enable such technologies a rapidly growing research area called “context-awareness” has emerged, as a technique to make pervasive applications more intelligent and accessible [26, 27, 68]. The term context is commonly understood by those working in pervasive computing, where it is felt that context is “key” in their efforts to disperse and enmesh computation into our lives [141]. The notion of context is much widely appreciated today, which usually refers to information about systems, entities, and their environments. Software applications that adapt their behaviour with the
changes of context information (e.g., location, temperature, and time) are called context-aware applications. A context-aware application uses context to provide relevant information and/or services to the user, where relevancy depends on the user’s task.

Humans, the centre of pervasive computing, however, are social beings. Hence, the notion of social context awareness (in short social awareness) can push the field further in achieving the vision of pervasive computing. Recently, it has been rising and attracting attentions of researchers from different communities, including pervasive computing and ambient intelligence. Nevertheless, pervasive computing is moving towards pervasive social computing with the pervasiveness of hand-held devices and enormous popularity of social networking websites.

Pervasive social computing (PSC) is a novel collective paradigm, derived from pervasive computing, social media, social networking, social signal processing and multimodal HCI. It aims to take advantage of human social relationships (a type of social context) to enable the fulfilment of users’ tasks on the move, ultimately promoting individual well being and social interactivity.

Despite the huge prospects of pervasive social computing and the extensive work in context-aware applications, so far very limited work has focused on socially-aware applications (in short socially-aware applications). An application is socially-aware if it uses social context information (e.g., social relationships, social roles, social interactions, situations) to adapt its behaviour. The advent of social media such as online social networks, blogs, and instant messaging, have radically changed the way people interact with each other and share information about their lives and works. Such use of social media platforms produces an unprecedented amount of social context information as people specify their relationships, update their status, share interests and contents. Thus, it is now possible to acquire users’ social context information from social media which may be rarely possible to acquire form physical sensors. Such social context information can be exploited to develop various data-centric socially-aware applications (where data is the basis of the applications’ behaviour). For example, SmartObject considers the users’ relationship information to turn on the audio player when friends are present; a socially-aware phone call application can reduce interruptions by allowing a user to filter incoming calls based on her social context information.
(e.g., relationships with the caller, current situation) [118].

On the other hand, the increased prevalence of advanced mobile devices (the so-called “smart” phones) with increasing features, interest has grown in interaction-centric socially-aware applications, where the collaborative interactions among persons are based on predefined agreements and constraints, and can be composed with the notion of interaction-relationships. Such applications can assist users in their daily activities and ultimately enrich their social interactions and well-being [131]. For instance, a socially-aware telematics application can make travel safe and convenient by allowing drivers to collaborate and interact with each other based on their interaction relationships [117].

Developing such socially-aware applications of distinct focus is a challenge for application developers as they must define social context information (e.g., social roles and interaction-oriented social relationships in a socially-aware telematics application) or collect social context information (e.g., social roles and connection-oriented social relationships in a socially-aware phone call application) from various sources (e.g., Facebook, LinkedIn, Google Calendar), represent and model such different social context information, infer more abstract meaningful context information, and manage them in a consistent manner. Therefore, the key requirements for developing socially-aware applications are modelling, managing and reasoning about users’ social context information [115].

1.2 Defining Social Context

The notion of context has been used in numerous areas, including Pervasive and Ubiquitous Computing, Human Computer Interaction, Computer-Supported Collaborative Work, and Ambient Intelligence [71]. In Ubiquitous and Pervasive Computing area, early works on context-awareness referred to context as primarily the location of people and objects [166]. In recent works, context has been extended to include a broader collection of factors, such as physical and social aspects of an entity, as well as the activities of users [71]. Having examined the definitions and categories of context given by the pervasive and ubiquitous computing community, this section seeks to define our view of social context within the scope of this thesis. As the definitions of context to pervasive and ubiquitous computing area are also broad, this discussion is intended to
Schmidt et al. [167] present a model of context with two distinct categories: human factors and physical environment. Human factors consist of three categories: information about the user (e.g., profile, emotional state), the user’s social environment (e.g., presence of other people, group dynamics), and the user’s tasks (e.g., current activity, goals). Physical environment also consists of three categories: location (e.g., absolute and relative position), infrastructure (e.g., computational resources), and physical conditions (e.g., noise, light). This model gives a classification according to specific contextual factors, but does not provide a formal definition. Dey [68] presents a survey of alternative view of context, which are largely imprecise and indirect, typically defining context by synonym or example. Finally, he offers the following definition of context, which is perhaps now the most widely accepted: “Context is any information that can be used to characterise the situation of an entity. An entity is person, place or object that is considered relevant to the interaction between a user and an application, including the user and the application themselves”. Henricksen [98] relates context to tasks, rather than with respect to interactions between users and applications, as in the definition of Dey. She separates the concepts of context, context modelling and context information. Henricksen argued that context represents a nebulous concept, is difficult to define and bound, where as context models and context information are well defined and understood, and are primary interest in constructing context-aware systems.

Several studies have attempted to define and represent social context from different perspectives. Han et al. [95] define social context as the user’s social surroundings, that is to say, the social relationships of the user. Eugster et al. [75] rely on a more restricted definition of social context. They consider distributed objects as peers and the social context of a peer represents its awareness of the existence of other peers. Zheng et al. [195] identify social context as one of the essential elements of the context space for online social interaction. They consider social context as social, cultural, psychological, and emotional influences on online social interactions. Wang et al. [184] analyse the role of the social group in a ubiquitous computing environment as a source of contextual information. They define social context as: “Information relevant to the characterisation of a situation that influences the interactions of one user with one or more other users”. Biamino [42] views
social context as social aggregations or social groups, and defines social context using
3-tuple expression (<number of nodes, number of connections between them, nature
of relations between the nodes>) that characterises a social network. Endler et al. [73]
introduce the term “situated social context” to enable location-based spontaneous inter-
action among people and define the term as: “Situated Social Context of an individual is
the set of people that share common spatio-temporal relationship with the individual, which turn
them into potential peers for information sharing or interacting in a specific situation”. Schuster
et al. [169] combines the concept of social context with pervasive context and introduce
the term “pervasive social context” which they define as: “Pervasive Social Context of an
individual is the set of information that arises our of direct or indirect interaction with people
carrying sensor-equipped pervasive devices connected to the same Social Network Service”.

The term “social context” can have many meanings or definitions, but most of the
above works view social context as possible forms of relationships and interactions among
people. Taking this insight, the following interpretations are adopted in this thesis:

- **Social Context** characterises social milieu\(^1\) of an individual with respect to an
  another individual or group of individuals.

- A **Social Context Model** represents a subset of the social milieu, which we define
  in terms of social roles, relationships, interactions and situations, of an individual
  with respect to another individual or a group. The social context model is
  employed by a given socially-aware application, is usually explicitly specified by
  the application developer but may evolve over time.

- **Social Context Information** is a set of data, gathered from various sources (e.g., social
  media) or explicitly specified by human, that conforms to a social context model.
  This provides a snapshot that approximates the state, at a given time, of the subset
  of the social context encompassed by the model.

- We distinguish social context from the concept of context used in previous research
  by calling the latter physical context. **Physical context** excludes **Social Context** and
  mainly refers to the contextual information about systems, entities, and their en-

\(^1\)refers to the social setting or environment in which people live or something happens
1.3 Research Challenges

Many of the research efforts in the area of context-aware applications are focused on modelling, reasoning and provisioning context information, so as to reduce the complexity of engineering such systems (see [41] for a survey). These efforts have achieved significant progress in formulating context models which are an important prerequisite for building systems that can respond to changes in their environment. In addition, software architectures, frameworks and platforms have also been proposed for developing and managing context-aware applications (see [152] for a survey).

Existing context models, software architectures and platforms for context-aware applications mostly address contexts of a physical nature such as location, time, activity, and so on. Comparatively, there have been only limited works investigating contexts of a social nature such as social relationships and social roles. Even though recently some works have attempted to model social context (e.g., [42]) and manage social context (e.g., [124]), they are limited in representing various aspects of social context. Furthermore, there is a lack of support for modelling and reasoning about different types of social context information and for managing the acquisition, changes and provision of such information (see Chapter 3 for a detail discussion).

This thesis explores the concept of social context as a means to represent and reason about social relationships, social roles, social interactions and situations. It presents a framework to provide high-level support for developing socially-aware applications by provisioning such different types of social context information and managing their runtime changes.

In order to aid the development of socially-aware applications, it is fundamental to understand

- what are the different types of social context and their use in developing socially-aware applications of distinct focus,

- how social context information can be acquired from different sources or can be composed,
how to model and represent such information,

- how to reason about more abstract and semantically rich social context information based on acquired information, and

- how to model runtime social interactions and manage their adaptation.

Towards this goal, we have formulated three key research challenges which will be discussed in the following three sections, respectively.

### 1.3.1 Consistent Representation of Social Context Information

There are different types of social context information. One type is the user’s social relationships which can be categorised into interaction- and connection-oriented relationships. The *interaction-oriented* relationships represent agreements and constraints regarding collaborative interactions among users, which use in developing interaction-centric socially-aware applications (e.g., the cooperative convoy application).

On the other hand, the *connection-oriented* relationships represent users’ relational ties which are used in developing data-centric socially-aware applications (e.g., the socially-aware phone call application). The connection-oriented relationships can be further categorised as “object-centric” and “people-centric” relationships [124]. The *object-centric* relationship is identified between people who have shown common interests (e.g., like/tag in Facebook) or participated in common activities or become members of similar groups. This type of relationship has been used in applications to infer preferences [94] and incentives of resource sharing [127]. The *people-centric* relationship is a formal and declarative definition of a direct relationship between people. For example, a person identifies other persons as father, supervisor, school friend, etc. This type of relationship can be used in a socially-aware phone call application as identified in [180], the SmartObject application [42], and the application to quantify review quality [130].

A user situation is his/her status or activity or state of affair with respect to surrounding people, which can be acquired from various sources (e.g., schedule description in Google Calendar, status update in OSNs). Such situation information is also impor-
tant for developing a socially-aware phone call application as identified in [180]. Thus, the challenge is to consistently model and represent different types of social context information such as social roles, social relationships, social interactions and situations, to facilitate applications use.

1.3.2 Reasoning about Meaningful Social Context Information

An application may need social context information that is not directly available from sources but can be derived from basic information. For instance, users may want to filter phone calls based on situation categories such as “busy” that may not be acquired from sources but can be inferred from collected data by specifying rules (e.g., meeting or seminar being in busy). Similarly, users may want to filter phone calls based on relationship categories such as “family” and “best-friends” that are not provided directly but can be inferred from the semantics of the relationship categories. Thus, there is the need to define and obtain derived relationships (at different abstraction levels) based on the basic relationships (e.g., mother and school-friend) and their semantics (e.g., mother being in family) and attributes (e.g., strength and trust).

To further enhance the services provided by the socially-aware applications, it might be required to infer situations based on users’ social interaction events. These events can be identified from users’ interaction activities in various social interaction platforms (e.g., Facebook, Email). The interaction events have temporal correlations. Furthermore, they are associated with users’ social context information such as their social relationships and social roles, and thus need to capture in the properties of interaction events. Many of these properties are ontological, i.e., their values have underlying semantics. As such, inferring situations from interaction events need to consider semantic matching of interaction event properties and temporal correlations among historical events. Thus, it is required to support inferring situations by observing and analysing current and past interaction events, and utilising ontological knowledge about such events.
1.3.3 Managing Access Control and Adaptation

The user’s social context information is inherently sensitive. The scenarios of emerging socially-aware applications require users to share their information for greater benefits but this may also compromise their privacy. For example, allowing a caller to know the status of the callee before calling might reduce interruptions, but may also raise serious concerns regarding the privacy and access control over users’ situation and other data. Thus, users should be able to retain control over who has access to their personal information under which conditions. In addition, a user may want to fine-tune the granularity of the answer provided to a given query, depending on the context of that query such as who is asking, what is asked for, and the user’s current situation. Therefore, it is necessary to provide a mechanism to support efficient access to this social context information while respecting information owners’ privacy.

Interaction-relationships based runtime social interaction models mediate social interactions between users of an interaction-centric socially-aware application. These runtime models typically evolve and many aspects of models such as topology, interaction constraints and non-functional quality properties need to modify frequently in response to changes in user requirements and environments. Thus, it is necessary to support adaptation in such runtime social interaction models.

1.4 Approach Overview

We propose a framework to address the aforementioned challenges in designing and implementing socially-aware applications of two different focuses such as data-centric and interaction-centric. The basis of the framework is an approach to modelling, reasoning about and managing social context information. The framework provides a platform, namely SocioPlatform (as illustrated in Fig. 1.1), to aid the development of socially-aware applications by acquiring, reasoning, storing and provisioning different types of social context information, and managing their runtime adaptation.

The SocioPlatform consists of social context information management (SCIM) and social interaction management (SIM) components. SCIM supports data-centric socially-aware
applications by acquiring, integrating, classifying and storing social context information from diverse sources such as Facebook, LinkedIn, Twitter and Google Calendar. It provides a number of functionalities including deriving social relationships, deducing situations from acquired information, and inferring situations from users’ interaction events. It also provides efficient access to social context information while preserving information owners’ privacy.

SIM provides the runtime environment and adaptation management for the interaction-centric socially-aware applications. The runtime environment instantiates runtime models and mediates social interactions among collaborative users. SIM supports the runtime adaptation of such model to cope with the changes in users’ requirements and environments.

1.4.1 Social Context Modelling

We adopt an ontology-based approach to modelling and representing social context. Ontology-based approaches have been evaluated as most promising for context modelling in pervasive computing [41]. In order to capture both the general and domain-specific concepts, we propose both the upper ontology and domain-specific ontologies for different types of social context information such as social role, social relationship and social interaction and situation. The upper ontology captures generic concepts, while the domain-specific ontology extends the upper ontology to capture the domain-specific concepts. One of the key characteristics of our modelling approach is that we model social context information as first-class entities rather than representing them as a generic link.
between people (as in existing approaches such as [182], discussed in Chapter 3). This way of modelling allows us to benefit from description logic (DL) in classifying and reasoning about social context information at different levels of abstraction based on the properties of social context.

1.4.2 Social Context Reasoning

We adopt an ontology-based approach and use OWL2 DL (a language for encoding ontology) for deriving, abstracting and deducing social context information (e.g., family, best friend, busy) from acquired information (e.g., father, school friend, meeting). OWL2 DL constructs allow us to specify reasoning rules by considering different aspects of social context information such as time-stamp. The main advantage of using OWL2 DL is that a DL reasoner can be used for executing reasoning rules and checking inconsistency in specification of such rules.

To address the temporal correlation aspect in situation reasoning, we propose a hybrid approach that combines ontology-based semantic matching and complex event processing in order to infer users’ situations from observed data, e.g., users’ interaction events. In this regard, we propose an ontology-based interaction event model to capture users’ interactions in various social interaction platforms (e.g., Facebook, cooperative convoy), which adopts and extends recent event models and incorporates social context information. We also propose a language to specify situations using atomic and complex interaction event patterns, which integrates their semantic aspect with temporal relations.

1.4.3 Social Context Management

We have taken policy-based approach to ensure users’ privacy preference in accessing their social context information. In this regard, we propose an ontology-based socially-aware access control policy model. Our policy model provides intuitive support by considering social relationship, social role and situation information when defining privacy preferences.

To manage social interactions among users of an interaction-centric socially-aware application, we propose an approach to modelling and managing runtime social inter-
actions. We adopt the view of an organisation being a composition structure of dynamic social roles in order to model runtime social interactions. The essences of our approach are the externalisation of the interaction relationships from the applications and the explicit modelling of such relationships from domain- and player-perspectives. The domain-centric social interaction model (DSIM) captures a collaborative view of the interaction relationships among the users/actors, whereas the player-centric social interaction model (PSIM) captures an individual’s coordinated view of all its interactions (across different domains), and thus supports coordination. We separate functional and management operations in such runtime models which facilitates their adaptation to cope with the continuous changes in users requirements and environments. We introduce states for the runtime models to enable adaptation in a safe manner. We also consider the inherent dependencies between DSIMs and PSIMs, and propose a protocol for adaptation propagation among these models.

1.5 Thesis Statement and Contributions

This thesis contends that existing context models, context reasoning and management approach have lacking to address the notion of social context to meet the challenges inherent in developing socially-aware pervasive computing applications. It proposes that the use of a specialised approach to modelling, reasoning about and managing social context, in conjunction with platform support for tasks such as social context gathering, management and provisioning, greatly simplifies the construction of socially-aware applications.

The contribution of this thesis lies in the development of a conceptual framework, and a corresponding software platform, that supports design and implementation tasks associated with social context-awareness. At the core of this framework is a novel approach to modelling, reasoning about and managing social context that allows context to be described at varying levels of abstraction with its semantics, deducing more meaningful and interesting social context information, and managing its acquisition, provisioning and runtime changes. In particular, followings are the original contributions of this thesis:
• An ontology-based social context model including the upper and domain-specific social context ontologies. The upper ontology defines and captures generic concepts such as social role, social relationship, social interaction and situation, while the domain-specific ontology extends the upper ontology to capture the domain-specific concepts.

• An ontology-based approach to deriving, deducing and abstracting social context information such as social relationships and situations, from basic acquired information such as social roles and users’ profile. A hybrid approach that combines semantic matching and complex event processing to inferring users’ situations from their interaction events. In this regard, we propose both the upper and domain-specific interaction event ontologies and a language to specify situations.

• An ontology-based socially-aware access control policy model and language to allow users to specify their privacy preference regarding access to their social context information.

• An approach to modelling runtime social interactions from both the domain- and player-perspectives, and managing their runtime adaptation.

• Finally, a platform, namely SocioPlatform for acquiring users’ social data from various sources, reasoning about more useful and interesting social context information, storing it into knowledge base and provisioning such information while preserving owners’ privacy preferences. It also supports to realise runtime social interaction models and to manage their runtime adaptation.

1.6 Thesis Structure

This thesis is structured into three main parts consisting of ten chapters. Part I consists of the introduction (Chapter 1), application scenarios (Chapter 2), and background and related work (Chapter 3) to provide the required background knowledge and themes for this thesis. Part II presents the approach to developing social context-awareness by presenting the ontology-based approach to modelling and representing different types of social context (Chapter 4), the approach to deriving, deducing, abstracting and infer-
Chapter 1. Introduction

ring social context information (Chapter 5), and the approach to managing privacy in accessing social context information and adaptation of runtime social interaction models (Chapter 6). Part III discusses the SocioPlatform with its implementation details (Chapter 7), case studies (Chapter 8), the evaluation results (Chapter 9), and finally a conclusion recapping the contributions of this thesis and related topics for future investigation (Chapter 10).

Chapter 2 presents two application scenarios that illustrate two socially-aware applications of distinct focus such as data-centric and interaction-centric, and their associated requirements. The key requirements for developing socially-aware applications are listed by summarising individual requirements of each of the application scenarios. These scenarios are also used as running examples by the remaining chapters of this thesis. Furthermore, these scenarios are used in case studies to show how the framework supports the design and implementation of different socially-aware applications.

Chapter 3 provides the related background and reviews the related work to analyse the existing approaches to modelling, reasoning about and managing context of physical nature. In addition, the lacking of the recent attempts to modelling, reasoning about and managing social context including runtime adaptation support are also introduced. Furthermore, the relevant work in existing literature is evaluated on the basis of the requirements of developing socially-aware applications as introduced in Chapter 2.

Chapter 4 presents our approach to modelling social context. In particular, it presents an upper and domain-specific ontologies for different types of social context information such as social role, social relationship, social interaction and situation. The upper ontology captures the basic concepts, abstracted from the analysis of application scenarios presented in Chapter 2 and the different sources of social context information such as Facebook, LinkedIn, Twitter and Google Calendar. A number of domain-specific ontologies are provided by specialising the concepts in the upper ontology.

Chapter 5 presents our approach to reasoning about social context information. An ontology-based approach is proposed to deriving social relationships, and deducing and abstracting situations where OWL2 DL is used to write reasoning rules. A hybrid
Chapter 1. Introduction

An approach that combines ontology-based semantic matching and complex event processing, is proposed to inferring users’ situations from their interaction events. In that regard, a language is proposed to specify situation inference rules.

Chapter 7 serves to integrate the theoretical contributions of Chapters 4, 5 and 6 into a system architecture and subsequently an implemented platform, namely SocioPlatform, incorporating the acquisition, management and querying of social context information, an environment for executing social interaction models and managing their runtime adaptation, and a set of APIs for developers to build socially-aware applications.

Chapter 8 describes two case studies involving the development of two prototypical social context-aware applications that have been discussed in Chapter 2, using our novel approach to modelling, reasoning about and managing social context information, and the associated SocioPlatform. This chapter demonstrates the use and applicability of our framework in developing socially-aware applications. At the same time, it highlights the benefits and strengths of the framework, and also validates the functionalities of the SocioPlatform.

Chapter 9 evaluates the functionality of the SocioPlatform by conducting a series of experiments using both real data from online social networks and simulated data, as appropriate. This chapter also evaluates the socially-aware applications that have been developed in Chapter 8 using real-life experiments. The results of these experiments demonstrate the real-world applicability and feasibility of developing socially-aware applications using our framework.

Chapter 10 concludes the thesis by summarising the key contributions of the thesis to the field of pervasive social computing, and then discussing some related topics for future investigation.
Chapter 2

Application Scenarios

In this chapter, we present two application scenarios that illustrate two different types of social context-aware applications (in short socially-aware applications) and their associated requirements. The objective of this chapter is twofold. The first objective is to present application scenarios which introduce different kinds of social context information and their use in developing various socially-aware applications. These application scenarios will be used throughout the next chapters to explain the framework for social context-aware applications by providing suitable examples. Furthermore, these scenarios will be used in case studies to show how the framework supports the development of various applications. The second objective is to motivate and identify key requirements of developing socially-aware applications by analysing these application scenarios.

The chapter is organized as follows. Section 2.1 presents a scenario related to phone call interruptions illustrating the needs and requirements of a data-centric socially-aware application which reduces phone call interruptions by exploiting users’ social context information, in particular, connection-oriented social relationships and situations. Section 2.2 presents a cooperative convoy scenario illustrating the needs and requirements of an interaction-centric socially-aware application which supports social interactions based on users’ interaction-oriented social relationships. Section 2.3 generalises the key requirements for developing socially-aware applications. Finally, Section 2.4 summarises this chapter.
2.1 A Data-centric Socially-Aware Application Scenario

In this section, we identify the requirements of developing data-centric socially-aware applications (where data drives the applications’ behaviour) by analysing the phone call interruptions scenario. Section 2.1.1 discusses the phone call interruptions scenario, while Section 2.1.2 analyses the requirements.

2.1.1 The Phone Call Interruptions

Mobile phones have the obvious benefit of at the moment communication. Irrespective of time and place, we can expect a phone to ring. But a ringing phone at an inopportune moment can be very disruptive to the current task or social situation. In a survey of 100 senior executives, it was reported that undesirable interruptions constitute 28% of the knowledge worker’s day, which translates to 28 billion wasted hours for companies in the United States alone. It results in a loss of 700 billion dollars per year, considering an average labor rate of $25 per hour for information workers [5]. Studies have also found that undesired disruption causes interrupted users to take up to 30% longer to complete and commit up to twice the number of errors. Judge Robert Restaino jailed 46 people when a mobile phone rang in his New York courtroom and no one would admit responsibility [1].

![Illustrations of phone call interruptions](image-url)

Figure 2.1: Illustrations of phone call interruptions [198]

We witness how often people are interrupted by incoming phone calls which not only disturb the user but also his/her surrounding people (Fig. 2.1 illustrates two scenarios). Mobile phone interruptions have sparked problems that have already been
CHAPTER 2. APPLICATION SCENARIOS

studied extensively, and they are many in number and can be serious in nature (e.g., they cause car accidents [13]). Meanwhile, the burden of managing social norms and patterns implied by mobile phone usage is left to the user. For instance, when one is in a meeting, he/she must remember to switch the phone to silent and decide whether to pick up any incoming call. Many of these interruptions could be avoided or adequately managed, only if the caller could be informed of the current status of the intended callee [123]. For example, if the caller could be informed that the callee is in a meeting, she might decide to call at a better time, thus avoiding or reducing harmful interruptions and possible socially embarrassing situations. On the other hand, the static nature of mobile phone configuration (e.g., reject all incoming phone calls while in a meeting) may lead to communication gaps and socially embarrassing situations [87]. For example, one may not want to miss any incoming call from his/her boss, even though he/she is in a meeting.

Let us consider that Alice, a university graduate student, wants to filter her incoming phone calls as follows: if my status/situation is meeting or seminar, and (i) a call comes from friend, the action is to reject, (ii) a call comes from family, the action is to reject and forward my status as Busy, (iii) a call comes from colleague, the action is to reject and forward status as meeting or seminar, (iv) a call comes from the supervisor, the action is to vibrate; (v) if my status is a non-business situation, the action is ring. Alice also wants to make her situation accessible by her colleague when she is in a meeting or seminar.

2.1.2 Requirements Analysis

A socially-aware phone call application is a way to minimize mobile phone interruptions that are mostly caused by (1) unawareness of each other’s (caller and callee) current situation, and (2) a lack of support to allow users to specify their dynamic filtering preferences over incoming phone calls. By observing how interruptions are currently handled by users, we might (at least partially) automate interruption management. Recently some studies (e.g., [34,65]) have tried to identify the context information (e.g., users’ situation) that is considered important in deciding whether to initiate phone call. In that regard, some other studies (e.g., [123,181]) have tried to identify factors that influence the users to share or disclose information about their situations. Some further stud-
ies also have focused on identifying actions that are performed currently by users to filter calls (e.g., [87]). Users’ social context, in particular, connection-oriented relationships (e.g., father-child, mother-child, husband-wife) and situations (e.g., meeting, seminar, watching a game in a stadium) have shown to have strong social underpinnings in both sharing current status information and filtering incoming phone calls [123, 180, 181].

A socially-aware phone call application that considers both the caller and callee perspectives, need to fulfill the following requirements:

First, the application should acquire its user’s current situation information and allow the user to share some of this information with other users. In this regard, the application should also allow its user to specify her privacy preferences to retain control over who has access to her situation information under which conditions.

Second, the application should allow its user to specify her phone call filtering preferences based on different social context information including social relationships and situations. When a phone call comes, the application needs to execute these filtering preferences and behaves accordingly. To check the filtering preferences, application needs its user’s social context information such as her current situations, her relationships with the caller, and so on. Studies have revealed that users want to define call filtering preferences that apply to all people with certain social relationships (e.g., family, best friends) or social roles rather than explicitly name each person [123, 181]. Thus, it is required to collect, classify, infer and manage different social context information.

2.2 An Interaction-centric Socially-Aware Application Scenario

In this section, we identify the requirements of developing interaction-centric socially-aware applications (where interactions drive the applications’ behaviour) by analysing the cooperative convoy scenario. Section 2.2.1 discusses the cooperative convoy scenario, while Section 2.2.2 analyses the requirements.
2.2.1 The Cooperative Convoy

Two groups of tourists in their two cars hired from two different rental companies want to drive together from Melbourne to Sydney. The car rental companies provide different levels of support to their customers based on the customers’ insurance policies, payment, service availability, and so on.

![Illustration of cooperative convoy](image)

Let us consider that the two cars, Car#1 and Car#2, as depicted in Fig. 2.2, are rented from Budget and AVIS respectively. The Budget Car Rental Company provides a car with a wet-sensor that can detect relevant road conditions and give an alert to reduce speed as necessary. The company also allows access to a Travel Guide Service (TGS) that suggests a possible route to a destination considering the shortest path. However, the service is unable to report real-time traffic information, e.g., road blocks due to road accidents. In contrast, the AVIS Company allows its car to access a Traffic Management Service (TMS) that provides information on traffic conditions ahead, e.g., the road is blocked due to accident or any emergency road works. However, both of these car rental companies provide a Roadside Assistance Service (RAS) to their customers. When a vehicle experiences a mechanical issue, its telematics system automatically informs the rental company and RAS, and requests a tow truck if the vehicle is no longer drivable.
Information about the vehicle location and the mechanical issue is also sent with the request. Upon receiving the request, the road-side service validates it with the car rental company and takes the necessary steps as defined in the agreement with that company.

Drivers of Car#1 and Car#2 want to form a cooperative convoy to make their travel safe and convenient by collaborating and interacting with each other. In a cooperative convoy, a vehicle interacts with other vehicles, service providers and infrastructure systems to make the travel safe and convenient. Through these interactions a vehicle’s driver can share information (acquired from the service providers and infrastructure systems) with other vehicles’ drivers in performing their tasks. Such interactions are subject to defined agreements and constraints among the entities (i.e., vehicle to vehicle (V2V), vehicle to service providers (V2SP), and vehicle to infrastructure (V2I)).

In the above scenario, we can see that Car#1 and Car#2 have access to different types of services: Car#1 has access to a Travel Guide Service (TGS), while Car#2 has access to a real time Traffic Management Service (TMS). In the cooperative convoy, they decide that Car#1 is the leading car (LC) whilst Car#2 is the following car (FC). The two cars follow the same route chosen by the leading car as it has access to a TGS. In addition, the drivers of both cars agree on a number of issues. For instance, they will always keep the distance between them less than 1 kilometre. Car#2 (the following car) will send road blocks information (obtained from its TMS) to Car#1 (the leading car) if there is any. Car#1 gets the updated route plan from its TGS by specifying its preferences (e.g., avoid the route with blocked road) and notifies that route information to Car#2. Both cars notify each other of their positions every 10 seconds. If either vehicle experiences mechanical problems (e.g., flat tyre, engine issue) it needs to notify the other vehicle.

2.2.2 Requirements Analysis

The socially-aware application facilitating the cooperative convoy needs to fulfill three major requirements:

First, the application should support interactions complying with the agreed interaction relationships (i.e., constraints and obligations). In the above scenario, these interaction relationships can be clustered into three groups by taking a collaborative view:
• **Interaction relationships between the driver of LC (Car#1) and the Budget Company with its associated services**: The interaction relationships such as getting route plan from the TGS, requesting road side assistance in case of any mechanical problem, and so on, fall into this category.

• **Interaction relationships between the driver of FC (Car#2) and the AVIS Company with its associated services**: The interaction relationships such as getting road blocks information from the TMS, and requesting road side assistance in case of any mechanical problem fall into this category.

• **Interaction relationship between the drivers of LC and FC**: The interaction-relationship between drivers of LC and FC (i.e., their mutual agreements) falls into this category. It includes that FC should send road blocks information to LC if there is any, LC should notify the updated route information to FC, FC follows the same route chosen by LC, both cars notify each other of their positions every 10 seconds, and in case of any mechanical problem both cars notify each other, and so on.

Thus, a model is required to capture the collaborative view of such interaction-oriented relationships.

*Second*, the application needs to provide a coordinated view of the interactions, and to allow drivers to specify their coordination preferences and to perform the coordination in an automated manner. According to the scenario, each driver is responsible for performing several tasks, for instance, the driver of LC is responsible for notifying route information, current positions, and mechanical problems (if any) to FC. Similarly, the driver of FC should send road blocks information, current positions, and mechanical problems to LC. For the drivers to perform such additional tasks (e.g., forwarding information) may cause distraction and have undesirable consequences. Thus, to facilitate collaboration with less distraction, it is required to provide a coordinated view of the interactions and to allow drivers to specify their coordination preferences.

*Third*, the application needs to support runtime adaptation, as the interaction relationships evolve over time and thus need to adapt with the changes in requirements and environments. For instance, a third vehicle could join when the convoy is on the way; or the break-down of a following vehicle might result in its leaving the convoy before
reaching the destination. A mechanical problem of the leading car may require it to
handover the leading car role to one of the following cars. Because of heavy rain, the
maximum distance may need to be reduced from 1 kilometre to 600 metres.

2.3 Requirements for Developing Socially-Aware Applications

The above discussions reveal that there are a number of key requirements for developing socially-aware applications:

**Req1 Modelling Social Context Information:** There are different types of social context information such as interaction- and connection-oriented social relationships, social roles, social interactions and situations. Such social context information needs to be modelled and represented consistently with its semantics to facilitate application use. Moreover, a pervasive social computing application assumed to be highly dynamic in the sense that it might require new types of social context information or might introduce new information sources that yield new types of social context, so the social context model should be flexible extensive.

**Req2 Reasoning about Social Context Information:** Application may need social context information that is not directly available from context sources but can be derived from acquired information. For instance, users may want to filter phone calls based on relationship categories such as “family” and “best-friends”, and situations such as “busy” that may not be acquired directly from sources but can be inferred from other acquired basic information. Thus, there is the need to define and obtain inferred social context information based on the semantics, attributes and temporal correlations of the acquired basic information.

**Req3 Preserving Privacy in Accessing Social Context Information:** A socially-aware application may require users to share their social context information for greater benefits but this also comprise their privacy. For example, in a socially-aware phone call application, allowing a caller to know the situation of the callee before calling might reduce interruptions, but may also raise serious concerns regarding the privacy and access control over users’ status and other data. Therefore, users should
be able to retain control over who has access to their personal information under which conditions. In addition, a user may want to fine-tune the granularity of the answer. For instance, an employee may be happy that her boss knows her current status is “at desk”. However, she may not be happy for her boss to know that her status is “chatting to friends via instant messaging”. Thus, there is a need to provide a privacy policy that should allow users to control access to their information at different levels of abstraction.

**Req4 Acquiring and Managing Social Context Information**: Social media such as online social networks, blogs, and instant messaging, have radically changed the way people interact with each other and share information about their lives and works \[118\]. The popularity of online social networks such as Facebook, LinkedIn, Twitter and Google+ produce an unprecedented amount of social context information as people specify their relationships, update their status and share contents. Google Calendar also can be a possible source of situation information as people use it to maintain their schedule \[129\]. Furthermore, users’ interaction events in different platforms, e.g., social media, can be exploited to infer their situations. Thus, it is required to acquire users’ such information from different sources and manage its changes for up-to-date, accurate and real time provisioning of social context information.

**Req5 Modelling Runtime Social Interactions**: Socially-aware applications that support social interactions based on the interaction-oriented relationships (i.e., agreements and constraints) among collaborative actors, need runtime representation of such relationships, called social interaction model. Such runtime social interaction model mediates runtime social interactions between actors. For instance, in the cooperative convoy, to support social interactions between drivers of Car#1 and Car#2 according to their interaction-relationship, a runtime model of such interaction-relationship is required. However, an actor may involve in multiple interaction-relationships and it may have its own view of social interactions related to these relationships. Thus, it is also required to support an individual user to coordinate his/her social interactions by providing a coordinated view of runtime social interaction model.
**Req6** Managing Adaptation in Runtime Social Interaction Models: Runtime social interaction models typically evolves and many aspects of runtime models such as topology, interaction constraints and non-functional quality properties need to modify frequently in response to changes in user requirements and environments. Thus, it is necessary to manage the adaptation in such runtime social interaction models. For instance, a third vehicle could join when the convoy is on the way; or the break-down of a following vehicle might result in its leaving the convoy before reaching the destination. As a consequence, the runtime social interaction model of convoy needs to be updated.

### 2.4 Summary

In this chapter, we have presented two application scenarios associated with two socially-aware applications of distinct focus such as *data-centric* and *interaction-centric*. These applications need different types of social context information and requires various supports in their development.

*First*, we have presented a phone call interruptions scenario that demands a *data-centric* socially-aware application. This application allows a person to filter incoming phone calls based on her situation and the relationships with the caller. Thus, the application needs *data*, e.g., connection-oriented social relationships and situations, to provide service to its users. The application also allows a caller to know the status of the intended callee before making the call. In this regard, the application requires to access the callee’s situation information while preserving his/her privacy.

*Second*, we have presented a cooperative convoy scenario that demands an *interaction-centric* socially-aware application. This application enables a cooperative convoy by supporting social interactions among drivers and external services, based on their predefined *agreements* and *constraints*, which we refer as *interaction-oriented social relationships*. The application needs a technique to *model* such interaction-oriented relationships. It also requires support to *manage adaptation* in such runtime model to cope with the changes in requirements and environments.
Finally, we have identified the key requirements for developing socially-aware applications.
Chapter 3

Background and Related Work

This chapter presents a review and discussion of related work from various areas within the scope of this thesis to analyse and position this thesis work with respect to the existing literature. It covers context modelling techniques, context models, event models, situation specification and reasoning techniques, privacy policy models and access control, context management frameworks and platforms. The review also highlights the principal research areas in which the existing solutions fall short of requirements for social context-aware applications, and motivates the research presented in the following chapters.

The chapter is organised as follows. Section 3.1 gives background information on ontology. Section 3.2 discusses a number of existing context modelling approaches and justifies our decision of choosing the ontology-based approach to modelling social context information. Section 3.3 characterises the existing ontology-based context models, and discusses recent attempts to modelling social context information and their limitations. Section 3.4 explores ontology-based event models, and identifies the common concepts of those models as the basis for our ontology based interaction event model. Section 3.5 discusses existing situation specification and reasoning techniques and their limitations in addressing both the semantic and temporal aspects. Section 3.6 highlights the gaps of existing privacy policy models in fulfilling the requirements for access control over users’ social context information. Section 3.7 discusses recent attempts to managing social context information and their limitations in acquiring, reasoning and
supporting access to social context information. Section 3.9 motivates the need of a new platform to support collaborative social interactions among users by highlighting the limitations of existing platforms in supporting runtime adaptation. Section 3.10 characterises platforms that support runtime adaptation but are not aimed for collaborative applications, and thus can not be adopted to fulfill our requirements. Finally, Section 3.11 summarises the existing work in the field of social context-awareness, highlights a variety of shortcomings, consequently motivates the need of further research, and subsequently positions the contributions of this thesis.

3.1 Ontology Preliminaries

We chronicle the historical definitions of the term ontology and expound on the value of ontology use (Section 3.1.1). Later, we give an overview of the development of ontological technologies, especially the standard ontology languages (Section 3.1.2). Finally, we discuss the OWL-DL, an ontology encoding language, constructs to illustrate its potentiality in representing knowledge (Section 3.1.3).

3.1.1 Ontology Definitions

In the Oxford Dictionary \[15\], an ontology is defined as: “The science or study of being; that branch of metaphysics concerned with the nature or essence of being or existence; or a theory or conception relating to the nature of being”. This philosophical definition reflects the essence of ontology: capturing the natural features of realities and relations between realities. The term was borrowed from philosophy and introduced into the knowledge engineering field as a means of abstracting and representing knowledge. Ontologies are used to build consensual terminologies for the domain knowledge in a formal way so that they can be more easily shared and reused.

Ontologies have been applied in many fields such as the semantic Web, information systems, service-oriented computing, and artificial intelligence. Accordingly, the definitions of ontologies have evolved with different and complementary points of view. Some of these definitions stress the general and intrinsic properties of ontologies, while others are influenced by the technical means of developing ontologies. We will illustrate
several typical definitions in both of the perspectives.

Ontology is defined as “a shared and common understanding of a domain that can be communicated between people and heterogeneous and distributed systems” \cite{76}, and more generally “a formal, explicit specification of a shared conceptualization” \cite{90}. Fensel \cite{76} elaborated on the definition of Gruber \cite{90} by saying that conceptualization refers to an abstract model of some phenomenon in the world by having identified the relevant concepts of that phenomenon. Explicit means that the type of concepts used and the constraints on their use are explicitly defined. Formal refers to the fact that the ontology describes what each concept is meant to denote, and specifies formal axioms that constrain the interpretation and well-formed use of these concepts. Shared reflects the notion that an ontology captures consensual knowledge; that is, it is not private of some individual, but accepted by a group. This definition describes how the philosophical nature of ontologies could be incorporated into computer science.

Guarino \cite{92} refined Gruber’s definition by distinguishing ontologies from conceptualizations formally and logically: “a logical theory accounting for the intended meaning of a formal vocabulary, i.e., its ontological commitment to a particular conceptualization of the world”. When this logical theory is used to model a particular aspect of reality - an intended domain - an ontological commitment is specified to capture the very basic ontological assumptions (such as identities and internal structures) about the intended domain. The ontological commitment gives explicit information about the intended nature of the modelling primitives and their a priori relationships for this logical theory. Thus, it constrains a subset of all the possible models of the logical theory by specifying the intended meaning of its vocabulary. These models being constrained are called intended models, which only describe those states of affairs that are compatible with the underlying ontological commitment \cite{93}. Furthermore, Guarino \cite{92} focused on the application side to offer a systematic account of the central role that ontologies played in information systems, leading to the ontology-driven information systems; that is, ontologies ‘drive’ all aspects and all components of an information system.

Taking the technical aspect, Huhns et al. \cite{109} define ontology as “a computer model of some portion of the world” \cite{109}. Recently, Tom Gruber \cite{89} again defines ontology as, “a set of representational primitives with which to model a domain of knowledge or discourse, where
the representational primitives are typically classes (or sets), attributes (or properties), and relationships (or relations among class members)”. This definition provides the representational primitives for modelling information and developing a knowledge base to managing such information. Thus, in this thesis context, we adopt this definition.

3.1.2 Languages for Encoding Ontology

In this section, we give an overview of a number of ontology encoding languages. In particular, we provide a brief introduction of Web Ontology Language (OWL) which is the broadly accepted ontology language, and discuss its strengths over other ontology languages (compiled from [31, 108, 137, 140, 142]).

The Web Ontology Language (OWL) [140] created based on the DAML+OIL [63], and extends Resource Description Framework (RDF) [39] and Resource Description Framework Schema (RDFS) [45]. Its primary aim is to bring the expressive and reasoning power of description logic (DL) [35] to the semantic web.

OWL takes the basic fact-stating ability of RDF and the class- and property-structuring capabilities of RDFS and extends them in important ways. OWL can declare classes, and organise these classes in a subsumption (‘subclass’) hierarchy, as can RDFS. OWL classes can be specified as logical combinations (intersections, unions or complements) of other classes, or as enumerations of specified objects, going beyond the capabilities of RDFS. OWL can also declare properties, organise these properties into a ‘subproperty’ hierarchy, and provide domains and ranges for these properties, again as in RDFS. The domains of OWL properties are OWL classes, and ranges can be either OWL classes or externally-defined datatypes such as string or integer. OWL can state that a property is transitive, symmetric, functional, or is the inverse of another property, here again extending RDFS.

OWL can express which objects (also called ‘individuals’) belong to which classes, and what the property values are of specific individuals. Equivalence statements can be made on classes and on properties, disjointness statements can be made on classes, and equality and inequality can be asserted between individuals.
However, the major extension over RDFS is the ability in OWL to provide restrictions on how properties behave that are local to a class. OWL can define classes where a particular property is restricted so that all the values for the property in instances of the class must belong to a certain class (or datatype); at least one value must come from a certain class (or datatype); there must be at least certain specific values and there must be at least or at most a certain number of distinct values. For example, using RDFS, we can

- declare classes like Country, Person, Student and Australian;
- state that Student is a subclass of Person;
- state that Australia and England are both instances of the class Country;
- declare Nationality as a property relating to the classes Person (its domain) and Country (its range);
- state that age is a property with Person as its domain and integer as its range; and
- state that Peter is an instance of the class Australian, and that his age has value 48.

With OWL, we can additionally

- state that Country and Person are disjoint classes;
- state that Australia and England are distinct individuals;
- declare HasCitizen as the inverse property of Nationality;
- state that the class Stateless is defined precisely as those members of the class Person that have no values for the property Nationality;
- state that the class MultipleNationals is defined precisely as those members of the class Person that have at least two values for the property Nationality;
- state that the class Australian is defined precisely as those members of the class Person that have Australia as a value of the property Nationality; and
- state that age is a functional property.
Description Logics (DL) [35], and insights from Description Logic research, had a strong influence on the design of OWL, particularly on the formalisation of the semantics, the choice of language constructors and the integration of datatypes and data values. A key feature of Description Logics is that they are logics, i.e., formal languages with well-defined semantics. As such, an OWL ontology may include logical descriptions of classes, properties and their instances. Given such an ontology, the OWL formal semantics (see Section 3.1.3) specifies how to derive its logical consequences, i.e., facts not literally present in the ontology, but entailed by the semantics. One advantage of OWL ontologies is the availability of tools that can reason about them.

The expressive power of a language like OWL is determined by the class (and property) constructors supported, and by the kinds of axioms that can occur in an ontology. Of course increased expressive power inevitably means increased computational complexity for key reasoning problems such as entailment. To address this issue, three species of OWL are defined [140]: OWL Lite, OWL DL and OWL Full.

**OWL Lite** can be used to express taxonomy and simple constraints, such as 0 and 1 cardinality. It is the simplest OWL language and corresponds to description logic \(SHIF(D)\). **OWL DL** supports maximum expressiveness while retaining computational completeness and decidability. The DL in the name shows that it is intended to support description logic capabilities. **OWL DL** corresponds to description logic \(SHOIN(D)\). **OWL Full** has no expressiveness constraints, but also does not guarantee any computational properties. It is formed by the full OWL vocabulary, but does not impose any syntactic constrains, so that the full syntactic freedom of RDF can be used.

These three languages are layered in a sense that every legal OWL Lite ontology is a legal OWL DL ontology, every legal OWL DL ontology is a legal OWL Full ontology, every valid OWL Lite conclusion is a valid OWL DL conclusion, and every valid OWL DL conclusion a valid OWL Full conclusion. The inverses of these relations generally do not hold. Also, every OWL ontology is a valid RDF document (i.e., DL expressions are mapped to triples), but not all RDF documents are valid OWL Lite or OWL DL documents. Thus, in this thesis, we use OWL DL for modelling and reasoning about social context information.
3.1.3 OWL DL Constructs for Representing Knowledge

OWL DL can form descriptions of classes, datatypes, individuals and data values using the constructs shown in Table 3.1. In this table the first column gives the OWL abstract syntax for the construction, while the second column gives the standard Description Logic syntax. OWL DL uses these description-forming constructs in axioms that provide information about classes, properties and individuals, as shown in Table 3.2. Again, the frame-like abstract syntax is given in the first column, and the standard Description Logic syntax is given in the second column.

A Description Logic knowledge base consists of two parts, namely the TBox and the ABox. The TBox consists of a number of class and property axioms; the ABox consists of a number of individual assertions (see Table 3.2). Here, C refers to a description, T refers to a concrete datatype; D refers to either a description or a datatype. R refers to an object property name, R_k refers to datatype property; Q refers to an object or datatype property where several appearances of R_i, R_j in one statement always refer to either both object or both datatype properties; o and t refer to object and concrete values, respectively. A class axiom in the TBox consists of two class descriptions, separated with the GCI (General Class Inclusion, or subsumption; \( \sqsubseteq \)) symbol or the equivalence symbol (\( \equiv \)), which is equivalent to GCI in both direction (i.e., \( \sqsubseteq \) and \( \sqsupseteq \)). Similarly, a property axiom consists of two property names, separated with the subsumption (\( \sqsubseteq \)) or the equivalence (\( \equiv \)) symbol. A description in the TBox is either a named class (A), an enumeration (\{o_1,...,o_n\}), a property restriction (\( \exists R \cdot D, \forall R \cdot D, \exists R \cdot o, \geq nR, \leq nR \)), analogously for datatype property restrictions), or an intersection (\( C \cap D \)), union (\( C \cup D \)) or complement (\( \neg C \)) of such descriptions (Table 3.1). Individual assertions in the ABox are either class membership (\( o \in C_i \)), property value (\( < o_1, o_2 > \in R_i, < o_1, t_1 > \in U_i \)), or individual (in)equality (\( o_1 = o_2, o_1 \neq o_2 \)) assertions (Table 3.2).

Description Logics have a set-based model-theoretic semantics. In an interpretation \( I \), a description C (Table 3.1) is mapped to a subset of the domain \( \Delta^I \) and an individual o is mapped to an object of \( \Delta^I \) using the mapping function \( \cdot^I \). Similarly, a datatype T is mapped to a subset of the concrete domain \( \Delta^I_D \) and a literal is mapped to a value in \( \Delta^I_T \). An abstract role R is mapped to a binary relation over the abstract domain:
Similarly, a concrete role \( R \) is mapped to a binary relation between the abstract and the concrete domain: \( \Delta^A \times \Delta^D \). Equivalence of descriptions is interpreted as set equivalence (\( C \equiv D \) is interpreted as \( C^A = D^A \)), subsumption is interpreted as set inclusion (\( C \subseteq D \) is interpreted as \( C^A \subseteq D^A \)), and so on. We refer the interested reader to [35] and [146] for a more exhaustive treatment of Description Logic and OWL DL semantics, respectively.

<table>
<thead>
<tr>
<th>Abstract Syntax</th>
<th>DL Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A ) (URI reference)</td>
<td>( A )</td>
<td>( \Delta^A \subseteq \Delta^A )</td>
</tr>
<tr>
<td>\texttt{owl:Thing}</td>
<td>( \top )</td>
<td>( \texttt{owl:Thing}^A = \Delta^A )</td>
</tr>
<tr>
<td>\texttt{owl:Nothing}</td>
<td>( \bot )</td>
<td>( \texttt{owl:Nothing}^A = { } )</td>
</tr>
<tr>
<td>\texttt{intersectionOf}(( C_1, C_2, \ldots ))</td>
<td>( C_1 \sqcap C_2 )</td>
<td>( (C_1 \sqcap D_1)^A = C_1^A \cap D_1^A )</td>
</tr>
<tr>
<td>\texttt{unionOf}(( C_1, C_2, \ldots ))</td>
<td>( C_1 \sqcup C_2 )</td>
<td>( (C_1 \sqcup C_2)^A = C_1^A \cup C_2^A )</td>
</tr>
<tr>
<td>\texttt{complementOf}(( C ))</td>
<td>( \neg C )</td>
<td>( \neg C^A = \Delta^A \setminus C^A )</td>
</tr>
<tr>
<td>\texttt{oneOf}(( o_1, \ldots ))</td>
<td>( {o_1, \ldots } )</td>
<td>( {o_1, \ldots }^A = {o_1^A, \ldots } )</td>
</tr>
<tr>
<td>\texttt{restriction}(( R ) someValuesFrom(( C )))</td>
<td>( \exists R.C )</td>
<td>( (\exists R.C)^A = { x \mid \exists y.(x, y) \in R^A \text{ and } y \in C^A } )</td>
</tr>
<tr>
<td>\texttt{restriction}(( R ) allValuesFrom(( C )))</td>
<td>( \forall R.C )</td>
<td>( (\forall R.C)^A = { x \mid \forall y.(x, y) \in R^A \to y \in C^A } )</td>
</tr>
<tr>
<td>\texttt{restriction}(( R ) hasValue(( o )))</td>
<td>( R : o )</td>
<td>( (R : o)^A = { x \mid \langle x, o^A \rangle \in R^A } )</td>
</tr>
<tr>
<td>\texttt{restriction}(( R ) minCardinality(( n )))</td>
<td>( \geq n R )</td>
<td>( (\geq n R)^A = { x \mid #{y.(x, y) \in R^A} \geq n } )</td>
</tr>
<tr>
<td>\texttt{restriction}(( R ) minCardinality(( n )))</td>
<td>( \leq n R )</td>
<td>( (\leq n R)^A = { x \mid #{y.(x, y) \in R^A} \leq n } )</td>
</tr>
<tr>
<td>\texttt{restriction}(( U ) someValuesFrom(( D )))</td>
<td>( \exists U.D )</td>
<td>( (\exists U.D)^A = { x \mid \exists y.(x, y) \in U^A \text{ and } y \in D^A } )</td>
</tr>
<tr>
<td>\texttt{restriction}(( U ) allValuesFrom(( D )))</td>
<td>( \forall U.D )</td>
<td>( (\forall U.D)^A = { x \mid \forall y.(x, y) \in U^A \to y \in D^A } )</td>
</tr>
<tr>
<td>\texttt{restriction}(( U ) hasValue(( v )))</td>
<td>( U : v )</td>
<td>( (U : v)^A = { x \mid \langle x, v^A \rangle \in U^A } )</td>
</tr>
<tr>
<td>\texttt{restriction}(( U ) minCardinality(( n )))</td>
<td>( \geq n U )</td>
<td>( (\geq n U)^A = { x \mid #{y.(x, y) \in U^A} \geq n } )</td>
</tr>
<tr>
<td>\texttt{restriction}(( U ) maxCardinality(( n )))</td>
<td>( \leq n U )</td>
<td>( (\leq n U)^A = { x \mid #{y.(x, y) \in U^A} \leq n } )</td>
</tr>
</tbody>
</table>

| Data Ranges (\( D \)) | \( D \) (URI reference) | \( D \) | \( D^P \subseteq \Delta^D \)
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{oneOf}(( v_1, \ldots ))</td>
<td>( {v_1, \ldots } )</td>
<td>( {v_1^A, \ldots } )</td>
<td></td>
</tr>
</tbody>
</table>

| Object Properties (\( R \)) | \( R \) (URI reference) | \( R \) | \( R^A \subseteq \Delta^A \times \Delta^A \)
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{inverseOf}(( R ))</td>
<td>( R^\preceq )</td>
<td>( (R^-)^A = (R^A)^\preceq )</td>
<td></td>
</tr>
</tbody>
</table>

| Datatype Properties (\( U \)) | \( U \) (URI reference) | \( U \) | \( U^A \subseteq \Delta^A \times \Delta^D \)
|-----------------------------|------------------------|--------|-----------------------------|

| Individuals (\( o \)) | \( o \) (URI reference) | \( o \) | \( o^A \in \Delta^A \)

| Data Values (\( v \)) | \( v \) (URI reference) | \( v \) | \( v^A = v^D \)

Table 3.1: OWL DL descriptions, data ranges, properties, individuals and data values [108]
### Table 3.2: OWL DL axioms and facts

<table>
<thead>
<tr>
<th>Abstract Syntax</th>
<th>DL Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Class</strong> (A \text{ partial } C_1 \ldots C_n)</td>
<td>(A \sqsubseteq C_1 \sqcap \ldots \sqcap C_n)</td>
<td>(A^I \subseteq C_1^I \sqcap \ldots \sqcap C_n^I)</td>
</tr>
<tr>
<td><strong>Class</strong> (A \text{ complete } C_1 \ldots C_n)</td>
<td>(A = C_1 \sqcap \ldots \sqcap C_n)</td>
<td>(A^I = C_1^I \sqcap \ldots \sqcap C_n^I)</td>
</tr>
<tr>
<td><strong>EnumeratedClass</strong> (A \ o_1 \ldots o_n)</td>
<td>(A = {o_1, \ldots, o_n})</td>
<td>(A^I = {o_1^I, \ldots, o_n^I})</td>
</tr>
<tr>
<td><strong>SubClassof</strong> (C_1 \ C_2)</td>
<td>(C_1 \sqsubseteq C_2)</td>
<td>(C_1^I \subseteq C_2^I)</td>
</tr>
<tr>
<td><strong>EquivalentClasses</strong> (C_1 \ldots C_n)</td>
<td>(C_1 = \ldots = C_n)</td>
<td>(C_1^I = \ldots = C_n^I)</td>
</tr>
<tr>
<td><strong>DisjointClasses</strong> (C_1 \ldots C_n)</td>
<td>(C_i \cap C_j = \bot, i \neq j)</td>
<td>(C_i^I \cap C_j^I = {}, i \neq j)</td>
</tr>
<tr>
<td><strong>Datatype</strong> (D)</td>
<td>(D^I \subseteq \Delta_D)</td>
<td>(D^I \subseteq \Delta_D^I)</td>
</tr>
<tr>
<td><strong>DatatypeProperty</strong> (U \ super(U_1) \ldots super(U_n))</td>
<td>(U \sqsubseteq U_i)</td>
<td>(U^I \subseteq U_i^I)</td>
</tr>
<tr>
<td>domain (C_1) \ldots domain (C_m)</td>
<td>(\exists U \sqcap D_i)</td>
<td>(U^I \subseteq C_i^I \times \Delta_D^I)</td>
</tr>
<tr>
<td>range (D_i) \ldots range (D_l)</td>
<td>(\forall U \sqsubseteq D_i)</td>
<td>(U^I \subseteq \Delta^I \times D_l^I)</td>
</tr>
<tr>
<td><strong>SubPropertyOf</strong> (U_1 U_2)</td>
<td>(U_1 \sqsubseteq U_2)</td>
<td>(U_1^I \subseteq U_2^I)</td>
</tr>
<tr>
<td><strong>EquivalentProperties</strong> (U_1 \ldots U_n)</td>
<td>(U_1 = \ldots = U_n)</td>
<td>(U_1^I = \ldots = U_n^I)</td>
</tr>
<tr>
<td><strong>ObjectProperty</strong> (R \ super(R_1) \ldots super(R_n))</td>
<td>(R \sqsubseteq R_i)</td>
<td>(R^I \subseteq R_i^I)</td>
</tr>
<tr>
<td>domain (C_1) \ldots domain (C_m)</td>
<td>(\exists R \sqcap C_i)</td>
<td>(R^I \subseteq C_i^I \times \Delta^I)</td>
</tr>
<tr>
<td>range (C_i) \ldots range (C_l)</td>
<td>(\forall R \sqsubseteq C_i)</td>
<td>(R^I \subseteq \Delta^I \times C_l^I)</td>
</tr>
<tr>
<td>[inverseOf (R_0)]</td>
<td>(R = \sim R_0)</td>
<td>(R^I = (R_0^I)^\sim)</td>
</tr>
<tr>
<td>[Symmetric]</td>
<td>(R = \sim R)</td>
<td>(R^I = (R^I)^\sim)</td>
</tr>
<tr>
<td>[Functional]</td>
<td>(\exists \sim \sqsubseteq R)</td>
<td>(R^I = (R^I)^\sim)</td>
</tr>
<tr>
<td>[InverseFunctional]</td>
<td>(\exists \sim \sqsubseteq R^-)</td>
<td>(R^I = (R^I)^\sim)</td>
</tr>
<tr>
<td>[Transitive])</td>
<td>(Tr(R))</td>
<td>((R^I)^\sim) is functional</td>
</tr>
<tr>
<td><strong>SubPropertyOf</strong> (R_1 R_2)</td>
<td>(R_1 \sqsubseteq R_2)</td>
<td>(R_1^I \subseteq R_2^I)</td>
</tr>
<tr>
<td><strong>EquivalentProperties</strong> (R_1 \ldots R_n)</td>
<td>(R_1 = \ldots = R_n)</td>
<td>(R_1^I = \ldots = R_n^I)</td>
</tr>
<tr>
<td><strong>AnnotationProperty</strong> (S)</td>
<td>(S^I \subseteq D^I \times \Delta^I)</td>
<td>(S^I \subseteq \Delta_S^I)</td>
</tr>
<tr>
<td><strong>Individual</strong> (o \ type(C_1) \ldots type(C_n))</td>
<td>(o \in C_i)</td>
<td>(o^I \in C_i^I)</td>
</tr>
<tr>
<td>value (R_1 o_1) \ldots value (R_n o_n)</td>
<td>(\langle o, o_i \rangle \in R_i)</td>
<td>(\langle o^I, o_i^I \rangle \in R_i^I)</td>
</tr>
<tr>
<td>value (R_1 v_1) \ldots value (R_n v_n)</td>
<td>(\langle o, v_i \rangle \in U_i)</td>
<td>(\langle o^I, v_i^I \rangle \in U_i^I)</td>
</tr>
<tr>
<td><strong>SameIndividual</strong> (o_1 \ldots o_n)</td>
<td>(o_1 = \ldots = o_n)</td>
<td>(o_1^I = \ldots = o_n^I)</td>
</tr>
<tr>
<td><strong>DifferentIndividuals</strong> (o_1 \ldots o_n)</td>
<td>(o_i \neq o_j, i \neq j)</td>
<td>(o_1^I \neq o_j^I, i \neq j)</td>
</tr>
</tbody>
</table>
3.2 Context Modelling Approaches

One of the major research foci in context-aware computing is on context modelling to reduce the complexity of engineering context-aware systems. A number of modelling techniques, such as the key-value based modelling [165], mark-up schema [97], graphical approach [101], object-oriented modelling [168], logic-based modelling [150], and ontology-based modelling [185] [56], have been developed. Strang and Linnhoff-Popien [176] described these modelling approaches, which are based on the data structures used for representing and exchanging contextual information in the respective systems.

**Key-Value based modelling.** The model of key-value pairs is the simplest data structure for context modelling. A key (or identifier) in this type of model corresponds to an attribute of the environment that has a value. This value is usually acquired from sensors or data sources. For example, <Room 31 temperature, 24> is such a <key, value> pair. Key-value models were used by Schilit et al. [165] to manage location information. The model of key-value pairs is easy to manage, but has problems with its expressiveness in representing complex structures.

**Markup scheme based modelling.** Markup-based models use a hierarchical data structure consisting of markup tags with attributes and content. Profiles represent typical markup scheme models. Typical examples are the Composite Capabilities/Preference Profile (CC/PP) and User Agent Profile (UAProf) which are encoded in RDF/S with XML notation [21]. Various other examples can be found in [176].

**Graphical modelling** The Unified Modelling Language (UML) is also used for modelling context. Various approaches exist where contextual aspects are modelled by using UML, e.g., [172]. Another modelling approach includes an extension to Object-Role Modelling (ORM) by context information presented in [101].

**Object-oriented modelling** This is based on the concept of objects and relationships between them as in the object-oriented programming paradigm. Rather than using implementation-based concepts like records, object-oriented models provide
flexible structuring capabilities by using fully-fledged object-orientation mechanisms such as encapsulation, reusability, and inheritance. Objects are used to represent various types of contextual information (e.g., temperature and location); they encapsulate the operations used for context processing. For example, Hydrogen [106] uses such an object-oriented model.

**Logic based modelling** Logic-based models have a high degree of formality. Typically, facts, expressions and rules are used to define a context model. A logic based system is then used to manage these terms and allows to add, update or remove new facts. An inference (also called reasoning) process can be used to derive new facts based on existing facts and rules in the systems. The contextual information needs to be represented in a formal way as facts. One of the first approaches was published by McCarthy and Buvac [136], Ranganathan et al. [151] and Katsiri and Mycroft [122] applied first-order logic to reason with contextual information. Loke [128] took rule-based programming further by representing situations for triggering context-aware actions.

**Ontology based modelling** Ontologies represent a description of the concepts and relationships. Therefore, ontologies are a very promising instrument for modelling contextual information due to their high and formal expressiveness and the possibilities for applying ontology reasoning techniques. Ontology-based techniques support a vocabulary for context predicates [103], so that the data representations in a system can be shared across different systems, or even retrieved from a storage over the Web. Various context-aware frameworks use ontologies as underlying representations for context models such as CoBrA [19], Gaia [157] and SOCAM [91]. They use ontologies and the tools from the “Semantic Web” to represent and reason about context. A discussion and thorough review of these ontology-based context models can be found in [192].

Strang and Linnhoff-Popien further specify six high level requirements for context-aware systems to evaluate the aforementioned context modelling approaches [176]. These requirements are summarised by Christian [103] as follows:

- **Distributed Composition (dc)** — Context-aware systems are distributed, and thus do
not have any central instance responsible for the creation, deployment, or maintenance of data and services, particularly context descriptions. Instead, the composition and administration of a context model and its data varies with notably high dynamics in terms of time, network topology, and source.

- **Partial Validation (pv)** — On the structural as well as on the instance level, there is no single place or point in time where the contextual knowledge is available on one node as a result of distributed composition. This is particularly important because of the complexity of contextual interrelationships, which make any modelling intention error-prone.

- **Richness and Quality of Information (qua)** — The quality of information delivered by sensors varies over time, and the richness of information provided by different kinds of sensors characterizing an entity in an ubiquitous computing environment, may differ. Thus, a context model appropriate for use in ubiquitous computing should inherently support both quality and richness.

- **Incompleteness and Ambiguity (inc)** — The set of contextual information available at any point in time characterizing relevant entities in ubiquitous computing environments is usually incomplete and/or ambiguous, particularly if this information is gathered from sensor networks. This should be covered by the model, for instance by interpolating incomplete data on the instance level.

- **Level of Formalism (for)** — It is always a challenge to describe contextual facts and interrelationships in a precise and traceable way. For instance, carrying out the task “print document on the nearest printer” requires a precise definition of terms used in the task, for instance what “nearest” means. It is extremely important that all the entities share the same interpretation of the data that are exchanged, as well as their underlying meaning.

- **Application to Existing Environments (app)** — From the implementation perspective, it is important for a context model to be applied within an existing infrastructure of ubiquitous computing components.
### Table 3.3: Comparison of the different context modelling approaches

<table>
<thead>
<tr>
<th>Approach</th>
<th>dc</th>
<th>pv</th>
<th>qua</th>
<th>inc</th>
<th>for</th>
<th>app</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key-Value based modelling</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Markup Schema based modelling</td>
<td>+</td>
<td>++</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>++</td>
</tr>
<tr>
<td>Graphical modelling</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Object oriented modelling</td>
<td>++</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Logic based modelling</td>
<td>++</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>++</td>
<td>-</td>
</tr>
<tr>
<td>Ontology based modelling</td>
<td>++</td>
<td>++</td>
<td>+</td>
<td>+</td>
<td>++</td>
<td>+</td>
</tr>
</tbody>
</table>

Table 3.3 shows the summary of the evaluation of the context modelling approaches based on the above six requirements. The evaluation summary shows that ontology is the most expressive context modelling approach and fulfill most of the requirements of context-aware systems. Furthermore, the ontology-based approach exhibits *four* main technical strengths over other modelling approaches [154, 176, 192]:

i. Ontology promotes knowledge sharing and reuse across different applications and services interacting in a pervasive computing environment. Subsequently, this promotes interoperability between different applications and devices.

ii. Ontological representation allows context-aware systems to use existing logic reasoning mechanism to deduce high level, conceptual context from low level, raw context and handle uncertainty and inconsistency in context.

iii. Ontologies can be combined to form a more complex ontology and hence this saves the effort of starting from scratch.

iv. Finally, a number of graphical tools, rich languages and inference engines are available to support designing, representing and processing ontologies using computer.

Thus, we adopt the *ontology-based approach for modelling and reasoning* about social context information.
3.3 Ontology-based Context Models

In the field of general context-aware systems, there has been significant amount of research effort to building domain ontologies for pervasive computing, including CoBrA [19, 55], Gaia [157], GLOSS [66], ASC [177, 178], and SOCAM [91]. However, all of these research aim to model context of a physical nature such as location, time, activity, and so on. None of them attempts to model social context which is the focus of this thesis. As less relevant, here we do not discuss these context models in details, and refer reader to [192] for a survey and detail discussion.

Recently some research efforts such as [16] and [67], have attempted to adopt and extend the Friend of a Friend (FOAF) [9] ontology to represent social relationships. They extend the foaf:knows object property, the only option offered by the FOAF ontology, with sub-properties such as colleagueOf and friendOf. However, representing relationships using such object property suffers from a lack of generality for two main reasons:

1. it does not allow the specification of different attributes such as strength and trust associated with a relationship; and
2. as a consequence more abstract and rich context information cannot be derived, such as bosom friend, best friend, and so on.

To overcome these limitations, in our approach, we model social context information, as such social relationships, as first-class entities rather than representing them as a generic link between people (like aforementioned approaches). This way of modelling allows us to benefit from description logic (DL) [35] in classifying and reasoning about relationships at different levels of abstraction based on the properties of these relationships. Furthermore, in addition to social relationships, we also model other types of social context information such as social roles, social interactions and situations (see Chapter 4).

3.4 Ontology-based Event Models

An event is something of interest that happens within a particular system or domain [74]. Events can be compound of other events that build the complex events. An event type is
a specification for a set of event objects that have the same semantic intent and same structure; every event object is considered to be an instance of an event type. A simple event pattern is an event type with or without logical constraints on its attributes/properties/contents, e.g., $E1(\text{attr1} = \text{"zzz"} \text{ and attr2} > 10 \text{ or attr3} \neq 6.5)$. A simple event pattern $F$ is a stateless truth valued function that is applied to an event object. An event object $e$ matches a simple event pattern $F$ if and only if it satisfies all attributes of pattern $F$ and logical relationships among those attributes. A complex event pattern is a temporal/logical relationships among two or more simple event patterns. Allen [28] proposed thirteen temporal relationships among events such as $E1 \text{ during } E2$, $E1 \text{ after } E2$, and $E1 \text{ coincides } E2$.

A number of ontology-based models have been proposed for representing events in the Semantic Web. Shaw et al. [171] proposed the Linking Open Descriptions of Events (LODE) model for publishing records of events as linked data and gave a comparison of some existing event models including Event Ontology (EO) [148], Event-Model-F [164], DOLCE+DnS Ultralite (DUL) [84], CIDOC-CRM [70] and ABC ontology [125], based on their main constituent properties like type, time, space, participation, causality, and composition. Hage et al. [24] proposed the Simple Event Model (SEM) to model events in various domains such as history, cultural heritage and geography. SEM is designed with a minimum of semantic commitment to guarantee maximal interoperability. The authors also classified the EO, LODE and CIDOC-CRM event models and highlighted their limitations on the basis of four design choices: domain specificity, focus on classes or properties, scope, and the level of formalisation.

Even though the above existing event models differ significantly in purpose and content, they share a set of common background concepts such as event, time, and place. We generalise and relate those concepts to form a core event ontology which we extend further to establish our interaction event ontology by incorporating social context information and other concepts related to interaction events (see Section 5.4.1).
3.5 Situation Specification and Reasoning

A substantial amount of work has been carried out on defining, modelling, specifying and reasoning about situations. Earlier work on situations from Dey [68], Henricksen et al. [99] and Loke [128] focused on the concept of situations as combinations of logical constraints on context information. More recent approaches from Jakkula and Cook [111], and Augusto et al. [32] consider temporal aspect in situation recognition. Here, we discuss the most relevant works highlighting their focus and approach in contrast to ours. A survey of different situation reasoning approaches with detailed discussions can be found in [193].

Dey [68] took the initiative in considering situations as the abstraction of context that was more influential in triggering applications. He proposed a simple logic formula (only using AND to connect context constraints). Henricksen et al. [99] propose to represent a situation specification in a restricted form of quantifications on predicate logic, which serves to restrict possible values for free variables. Loke presents a declarative approach to representing and reasoning with situations at a high level of abstraction where a situation is characterised by imposing constraints on the output or readings returned by sensors [128].

Wang et al. [185] model low-level context in an ontological model and derived high-level context using both ontology reasoning with description logic and user-defined reasoning that is defined in specific rules in first-order logic. As ontology is a popular approach in modelling context, it is also applied by other researchers such as Ranganathan et al. [149] and Chen et al. [58]. Like those approaches, we use ontology to model interaction events. But unlike us, none of these approaches have considered temporal aspect in their situation recognition.

Temporal information has been incorporated into a few situation specification approaches. Jakkula and Cook [111] use Allen’s temporal relations [28] as the basis for defining temporal rules across activities. The rules allow the sequence and overlap of activities to be encoded into a rule base. They compare the predictive accuracy of activities with and without the temporal rules, noticing an improvement when temporal rules are applied. This indicates that temporal information will help with situation
recognition, and is desirable for us to include into our interaction event-based situation reasoning approach.

Matheus et al. [135] also consider the temporal factor. In their work, a situation is defined as a set of relations with other objects. Augusto et al. [32] introduce the temporal operators ‘ANDlater’ and ‘ANDsim’ in event-condition-action rules, upon which temporal knowledge on human activities can be specified. This allows events that co-occur (ANDsim) and events that happen in a sequence (ANDlater) to be used in reasoning. Therefore, their work indicates that co-occurrence and sequence temporal relations improve situation recognition.

Even though some of the above existing works have considered temporal aspects in situation recognition. None of them support all types of temporal relations excepting Jakkula and Cook [111]. None of the existing works have considered both semantic and temporal aspects in situation specification. Moreover, none of them have considered real-time situation recognition over interaction event streams, even though they all have considered inferring situations from context information acquired from sensors and/or focused on “at the moment” situation recognition over static sensor data. To the best of our knowledge, our work is the first systematic attempt at inferring users’ situations considering both semantic and temporal aspects of interaction events coming from social media. These two aspects of our approach are discussed in the following two sections.

### 3.6 Privacy Policy Models and Access Control

Policy-based access control has been the subject of extensive research over the past decade. Recent research efforts have tried to integrate semantic web technologies both in access control models and policy specifications, thus enabling automated reasoning and policy enforcement over expressive access control specifications. In particular, the ROWLBA [80] and ReBAC [81] models have provided us with useful insight for our socially-aware access control framework.

Most policy models for social networking applications [52], and even those designed for pervasive computing applications [110] consider either role or relationship in
their access control but not both. While the work [181] is very close to ours in protecting user’s social context information, it does not consider role in its access control model and therefore is not able to offer the advantages of role based access control. Moreover, the work does not consider obfuscation in supporting the access control at different granularity levels, which is an important aspect for access control [189].

3.7 Managing Social Context Information

Socially-aware applications are often designed from scratch by embedding all management functionalities into the application logic, providing an application-specific data representation models, and acquiring data from one or a few specific external sources (e.g., [42], [38]). In all of these cases, social knowledge has been mined in the context of a single application. Some efforts, such as [124] and [182], have already recognized the need to externalize the social context management functionalities and have taken steps towards systematically managing users’ social context information.

Prometheus [124] collects user’s social data from different OSNs and represents it as multi-edged graphs, where vertices correspond to users and edges correspond to interactions between users. The interactions are described with a label (e.g., “football”, “music”) and a weight specifies the intensity of an interaction, and essentially represents an “object-centric” relationship. Prometheus implements a set of inference functions to answer queries like social strength, proximity, and so on, while enforcing user-defined access control policies. Like Prometheus, PocketSocial [191] also collects social data from different sources. But unlike Prometheus, it represents social data in JSON objects and supports only REST based APIs like Facebook, and does not provide any inference functions. Neither Prometheus nor PocketSocial represent both the object- and people-centric relationships with their semantics, and as a consequence they are not able to infer richer information or fine-tune the granularity of information access.

On the other hand, Yarta [182] adopts the FOAF ontology in their relationship representation. But Yarta only considers people-centric relationships and does not capture the object-centric ones. Even though it does consider social context in its policy model, its access control policy does not take into account granularity in information access.
Moreover, its policy model is RDF-based which lacks in the generality of OWL DL-based models.

Online social networks (OSNs) like Facebook, LinkedIn and so on, offer their native APIs to access simple and unprocessed social data. These APIs do not provide access to derived relationships like “Best friend” or “Father-Son”. Instead, an application must explicitly crawl through the graph to obtain them.

Our work significantly differs from the above noted approaches in that it not only collects users’ social relationship information (both object- and people-centric) from multiple sources and stores it in richer ontologies, but also considers the owners’ status information and their semantics, allowing information representation and derivation at different levels of abstraction and consequently facilitating fine-grained access control and query processing.

3.8 Modelling and Managing Runtime Social Interactions

According to the sociologist Max Weber, “an action is social if the acting individual takes account of the behaviour of others and is thereby oriented in its course” [186]. Social interaction refers to the sequence of social action between individuals (or groups) by considering their relationships.

Social interaction has been studied in multi-agent systems. In a multi-agent system, agents usually interact in order to achieve better goals of them or the system in which they exist. Kalenka et al. [120] incorporate social concepts within the computer system level paradigm to enhance the agent’s interaction performance. Rather than defining social relationships, the work proposes multi-furious social interaction attitudes ranging from self interested to the purely altruistic.

Role-based design has been extensively researched in last few decades and adopted in multiple disciplines (e.g., agent-oriented software engineering (AOSE), distributed systems (DS), object-oriented methodologies (OOM), and so on) as a key concept for modelling and managing interactions. In AOSE, many frameworks (e.g., BRAIN [50], Gaia [194], and so on) have been proposed to design role-based software systems us-
ing agent paradigm. In their survey paper, Cabri et al. \cite{49} compare the strength of existing AOSE proposals based on some evaluation criteria and showed BRAIN is more expressive compared to other frameworks. The existing proposals in AOSE (including BRAIN), DS (\textit{e.g.}, \cite{132}) and OOM (\textit{e.g.}, \cite{153}), however, mainly consider up to design phase of software engineering and give no indication of how these roles are to be realized. Even though some of the works (\textit{e.g.}, ROADMAP \cite{114}) further extended to consider runtime/implementation phase by incorporating different interaction language such as AUML \cite{36}, they do not consider roles as an implementation entity and provide very limited support for managing runtime adaptation. In such approaches roles are used in the modelling and to inform the design, but disappear as entities during implementation. Thus, they are not suitable for managing adaptations in runtime model.

Compared to the above mentioned frameworks, Role-Oriented Adaptive Design (ROAD) \cite{59} brings a number of design principles that support flexible management and runtime adaptations, which has also been recognized and appreciated in \cite{54}. One of such principles is separation of functional and management operations. This principle is important for facilitating application development and runtime adaptation as the functional operations performed by the actors enable normal interactions while the management operations realize adaptation to the structure and parameters of the model. Thus, we adopt ROAD architectural style for modelling and managing \textit{social interactions} at runtime (see Section \ref{sec:runtime-adaptation}).

### 3.9 Platform for Collaborative (Social) Interactions

Most of the approaches in the pervasive computing literature mainly focus on tasks of individual users, and provide very limited support to collaborative tasks of a group of users \cite{161}. Furthermore, existing approaches in developing collaborative pervasive applications (\textit{e.g.}, \cite{44,47,53,72,160}), and middleware architectures and frameworks for pervasive computing \cite{152} are limited in supporting the dynamic \textit{relationships} between users, which themselves are an important aspect of context. In particular, there is a lack of support for managing the \textit{adaptation} in interaction-relationships in response to \textit{changes} in the requirements and environments.
The need for supporting collaboration in pervasive computing environments has emerged in recent years (e.g., [53], [44]). Such research has focused on collaborative interactions between different types of actors such as user-user and device-device, for various purposes. The SAPERE [53] middleware exploits social network graph to establish collaboration for sharing data among spatially collocated users devices. CoCA [72] is an ontology-based context-aware service platform for sharing of computational resources among devices in a neighbourhood. Both SAPERE and CoCA focus on collaboration among devices, where the social interactions in this thesis primarily target on collaboration among users. Similar to us, MoCA [160], a middleware architecture for developing context-aware collaborative applications, focuses on collaboration among users. But unlike us, the collaborations among users in MoCA are not based on predefined goals or tasks, rather driven by spontaneous and occasional initiatives. CASMAS [47] and UseNet [44] focus on collaborative activities among users to achieve a common goal like us. But none of them explicitly model the interactions among users. Moreover, all of the above approaches lack support for managing the dynamicity and complexity of the social interactions as highlighted in this thesis. The interaction relationships between actors and their adaptations are not modelled explicitly, and instead are often hard-coded directly into the applications. To the best of our knowledge, there is no work to date that addresses the runtime adaptation of social interaction models in response to the changes in requirements and environments.

### 3.10 Platform for Runtime Adaptation

Much research has been carried out into middleware support for runtime adaptation in context-aware systems (e.g., MADAM [85] and 3PC [96]) and service-oriented systems (e.g., MUSIC [159] and MOSES [51]). These middleware solutions mainly target the tasks of individual users/applications and have focused on reconfiguring applications’ settings (rather than interaction relationships) based on physical context information (e.g., place, time)/quality of service requirements (e.g., performance, reliability), rather than interaction relationships. Moreover, their proposed runtime models are application-specific and cannot be used to model interaction relationships among collaborative users.
In contrast to these solutions, our adaptation management approach (see Section 6.2.2) targets interaction-centric socially-aware applications, and focuses on executing adaptation by explicitly modelling and realising interaction relationships using a social interaction model and providing an organiser interface to change such model. On the other hand, in this thesis, we do not address the monitoring of environment changes (i.e., physical context information), acquiring and analysing such physical context information to make adaptation decisions. In that sense, our adaptation management approach is not a substitute for existing middleware solutions that manages physical context information, rather can be built on top of those solutions as appropriate, in order to manage social interactions and runtime adaptation in interaction-centric socially-aware applications.

Some of these existing middleware solutions have a degree of similarity to our approach to managing adaptation. Like MADAM [85] and PCOM in 3PC [96], we have employed both structural and parametric adaptations and has adopted an external adaptation approach where adaptation capabilities of applications are realised by an application-independent middleware. Besides, we provide a management interface to trigger adaptations. Thus, any existing middleware solutions (e.g., MADAM) can be used to detect the changes in environmental context, make adaptation decisions and trigger adaptations in platform. Furthermore, users can explicitly trigger the adaptations as required. Like MUSIC [159] and MADAM [85], our approach provides a model-driven development and middleware support to build applications. Like us, MOSES [51] is a service-oriented architecture centric approach, but its adaptation logic is limited to a predefined set of attributes (i.e., response time, expected execution cost, and expected reliability). However, all of these approaches lack an explicit model of social interactions.

3.11 Summary

Despite extensive research on context-aware computing, so far very limited work has focused on social context-aware applications. Research on context-aware computing has largely focused on the development of software frameworks and platforms that perform task such as the acquisition of context information from the sensors, persistent storage
of the information within servers, processing applications queries to access such context information and support adaptation. These infrastructures serve an important role, shifting much of the complex functionality onto the middleware, and thereby simplifying the construction of robust applications \[107\] \[152\] \[155\]. As such, this type of framework support is also desirable for developing social context-aware applications.

Recently some research work have attempted to provide the functionalities, as described above, to aid the development of socially-aware applications. However, the diverse characteristics and application domain of such socially-aware applications demand for different types of social context information and functionality of the platform/middleware. In the previous chapter, we have described two socially-aware applications with two different focuses (i.e., data-centric and application-centric) and application domains (i.e., business and travel), and identified a set of requirements for developing such types of applications. In this chapter, we have analysed, examined and reviewed the existing approaches that aims to simplify the development of either social context-aware applications or context-aware applications in general, and provide one or more aforementioned functionalities such as context acquisition, modelling, reasoning and management. A summary evaluation of the examined approaches in this chapter against the requirements for developing socially-aware applications from Chapter 2 is given in Table 3.4. The information in the table show that none of the existing approaches explicitly supports either a singe requirement or collectively all the requirements.

Thus, in this thesis, we have attempted to explicitly support all the requirements. In a nutshell, there are three essential aspects towards a framework for social context-
aware applications, i.e., social context information *modelling*, *reasoning* and *management*, where the management includes preserving *privacy* on accessing social context information, *modelling* runtime social interactions and managing their *adaptation*, as shown in Fig. 3.1. Each aspect has its own challenges to address. These challenges and corresponding solutions are discussed in Chapters 4, 5 and 6 respectively.
Table 3.4: The summary of evaluation of the examined approaches

<table>
<thead>
<tr>
<th>Approach</th>
<th>Modelling social context information</th>
<th>Reasoning</th>
<th>Preserving Privacy</th>
<th>Acquiring and Managing</th>
<th>Modelling social interaction</th>
<th>Managing Adaptation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prometheus [124]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Yarta [182]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PocketSocial [191]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Toninelli et al. [181]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Jagtap et al. [110]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>BRAIN [50]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gaia [194]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROADMAP [114]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MoCA [160]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UseNet [44]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CASMAS [47]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MADAM [85]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3PC [96]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MUSIC [159]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOSES [51]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

+ Explicitly supported, * Supported to a certain extent, x Not supported, [blank] Not aimed or applicable
CHAPTER 3. BACKGROUND AND RELATED WORK
Part II
Chapter 4

Social Context Modelling

The social context model provides a basis on which the socially-aware application is developed and executed. This chapter presents an ontology-based approach to modelling social context information. We present ontologies for different types of social context such as social role, social relationship, social interaction and situation. These ontologies allow application developers to specify, represent and interpret users’ social context information which drives the behaviour of applications.

The chapter is organised as follows. Section 4.1 presents our modelling approach with its strengths and key features. Section 4.2 presents our social context upper ontology. A number of domain-specific ontologies are presented in section 4.3 by extending the concepts in the upper-ontology and analysing the application scenarios presented in Chapter 2.

4.1 Modelling Approach

We adopt an ontology-based approach to modelling and representing different types of social context information. In computer science, the term ontology has numerous descriptions and definitions as we discussed in section 3.1.1. In this thesis context, we adopt the recent operational definition given by Tom Gruber, “an ontology defines a set of representational primitives with which to model a domain of knowledge or discourse, where the
Chapter 4. Social Context Modelling

Representational primitives are typically classes (or sets), attributes (or properties), and relationships (or relations among class members) [89]. This definition provides us the representational primitives for modelling social context information and developing a knowledge base to managing such information.

Ontology-based approaches have been evaluated as most promising for context modelling in pervasive computing [41]. It has two main advantages:

i. It creates a common understanding and knowledge base of context information, thus, promotes re-usability and sharing; and

ii. It allows context reasoning.

We use OWL2-DL [88], a sub-language of OWL 2, to model social context information. To date, OWL2-DL has been the most practical choice for most ontological applications as it supports maximum expressiveness while retaining computational completeness and decidability [154].

Table 4.1: OWL-DL class constructors

<table>
<thead>
<tr>
<th>No.</th>
<th>Constructor</th>
<th>DL Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>intersectionOf</td>
<td>$C_1 \cap \cdots \cap C_n$</td>
<td>CollageFriend $\cap$ UniversityFriend</td>
</tr>
<tr>
<td>2</td>
<td>unionOf</td>
<td>$C_1 \cup \cdots \cup C_n$</td>
<td>Seminar $\cup$ Meeting</td>
</tr>
<tr>
<td>3</td>
<td>complementOf</td>
<td>$\neg C$</td>
<td>$\neg$AtDesk</td>
</tr>
<tr>
<td>4</td>
<td>oneOf</td>
<td>${x_1} \cup \cdots \cup {x_n}$</td>
<td>${father} \cup {mother}$</td>
</tr>
<tr>
<td>5</td>
<td>allValuesFrom</td>
<td>$\forall P \cdot C$</td>
<td>$\forall$plays $\cdot$ SocialRole</td>
</tr>
<tr>
<td>6</td>
<td>someValuesFrom</td>
<td>$\exists P \cdot C$</td>
<td>$\exists$relates $\cdot$ Person</td>
</tr>
<tr>
<td>7</td>
<td>maxCardinality</td>
<td>$\leq n P \cdot C$</td>
<td>$\leq 2$plays $\cdot$ SocialRole</td>
</tr>
<tr>
<td>8</td>
<td>minCardinality</td>
<td>$\geq n P \cdot C$</td>
<td>$\geq$strength $\cdot$ {0.8}</td>
</tr>
<tr>
<td>9</td>
<td>exactCardinality</td>
<td>$= n P \cdot C$</td>
<td>$= 2$relates $\cdot$ Person</td>
</tr>
<tr>
<td>10</td>
<td>hasValue</td>
<td>$\exists P \cdot I$</td>
<td>$\exists$plays $\cdot$ father</td>
</tr>
</tbody>
</table>

Table 4.1 presents OWL-DL class constructors and its use in representing knowledge about social context information. In particular, the table shows how we can construct new classes from existing named classes, properties and individuals by:

- applying set operators such as intersection, union, and complement on classes (#1,
CHAPTER 4. SOCIAL CONTEXT MODELLING

#2 and #3);

- explicitly and exhaustively enumerating the individuals that are members of the new class (#4);

- restricting a property of the class: the range of the property either has all the individuals from a specific class (universal restriction) (#5) or has some individuals in a specific class (existential restriction) (#6);

- restricting the maximum (#7) or minimum (#8) or exact (#9) cardinality of a property;

- specifying a particular value of the property (#10).

Furthermore, we can write a complex expression, namely reasoning rule, to represent a new class based on the above basic class constructor (see Chapter 5 for details).

Table 4.2: OWL-DL class semantics

<table>
<thead>
<tr>
<th>No.</th>
<th>Name</th>
<th>DL Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>EquivalentTo</td>
<td>$C_i \equiv C_j$</td>
<td>Husband (\equiv) WifeHusband</td>
</tr>
<tr>
<td>2</td>
<td>SubClassOf</td>
<td>$C_i \sqsubseteq C_j$</td>
<td>MotherChild (\sqsubseteq) ParentChild</td>
</tr>
<tr>
<td>3</td>
<td>DisjointWith</td>
<td>$C_i \cap C_j \sqsubseteq \bot$</td>
<td>MotherChild (\cap) FatherChild (\sqsubseteq) \bot</td>
</tr>
</tbody>
</table>

Using OWL-DL class semantics, as presented in Table 4.2, we can specify association between classes (named or new class) such as:

**Equivalence** – Individuals of a class are same as individuals to its equivalent class. We also can define a new class using DL expression as an equivalence class which usually referred to as a necessary and sufficient condition. Thus, all reasoning rules should be specified as equivalence. If a class $C_j$ is defined using necessary and sufficient condition(s) then we can say that an individual of the class $C_j$ must satisfies the condition(s) and we also can say that any (random) individual satisfies the condition(s) must be a member of the class $C_j$;

**Subclass/Superclass** – Individuals of a subclass are considered as individuals of its superclass. A subclass is more specialised concept of its superclass (is-a relation-
ship). We use this association to represent the **granularity** of concepts in ontology, i.e., a subclass is a more fine-grained concept of its superclass (as illustrated in Fig. [4.1]). We define an annotation property, called **granularityLevel** for explicitly specifying the **granularity level** of each concept in an ontology. Section 5.3.2 presents the granularity levels of different situations.

![Granularity levels illustration](image)

**Figure 4.1:** Illustration of granularity levels

We also can define a new class using DL expression as a subclass which usually imposes a *restriction/constraint* and is referred to as a necessary condition for the superclass. If a class $C_j$ is described using necessary condition(s), then we can say that an individual of the class $C_j$ must satisfy the condition(s); and

**Disjointness** – Individuals of a class specifically exclude individuals to its disjoint class. Like above, this class could be a primitive class or derived class (i.e., defined using DL expression)

In addition to OWL-DL class constructors and semantics, we can use OWL object property characteristics, to specify further association among classes based on their individuals. Properties link two individuals together, i.e., establish binary relations on individuals. As shown in Table 4.3, the OWL property characteristics enrich the meaning of the properties as follows:

- If a property is *transitive*, and the property relates individual a to individual b, and
also individual $b$ to individual $c$, then we can infer that individual $a$ is related to individual $c$ via property $P$.

- If a property $P$ is symmetric, and the property relates individual $a$ to individual $b$ then individual $b$ is also related to individual $a$ via property $P$.

- If a property $P$ is asymmetric, and the property relates individual $a$ to individual $b$ then individual $b$ cannot be related to individual $a$ via property $P$.

- A property $P$ is said to be reflexive when the property must relate individual $a$ to itself.

- If a property $P$ is irreflexive, it can be described as a property that relates an individual $a$ to individual $b$, where individual $a$ and individual $b$ are not the same.

- If a property is functional, for a given individual, there can be at most one individual that is related to the individual via the property. Functional properties are also known as single valued properties. If the property relates individual $a$ to individual $b$, and also individual $a$ to individual $c$, then we can infer that individual $b$ and $c$ must be the same individual. It should be noted however, that if $b$ and $c$ were explicitly stated to be two different individuals then the above statements would lead to an inconsistency.

- If a property is inverse functional then it means that the inverse property is func-
tional. For a given individual, there can be at most one individual related to that individual via the property

- If a property links individual a to individual b then its inverse property will link individual b to individual a

Thus, for modelling social context information, we use the OWL2-DL class constructors, semantics and property characteristics to represent concept hierarchies (i.e., super-class/sub-class relationships), and impose constraints on concepts (e.g., disjoint classes) and their properties (e.g., minimum cardinality). The main advantages of their use is that a DL reasoner can be used for (1) checking constraints and (2) detecting automatically inconsistencies in the social context information of the knowledge base.

### 4.2 Social Context Upper Ontology

Despite its intuitive anchoring in everyday life, social context can be very challenging to represent in a formal model. In section 1.2, we define social context model as a representation of users’ social roles, social relationships, social interactions and situations. In this section, we use OWL2-DL constructs (as introduced in the previous section) to model such different types of social context.

In order to capture the general concepts and incorporate domain-specific concepts, we propose both the upper and domain-specific ontologies. The upper ontology captures the basic concepts, abstracted from the analysis of: (i) different types of social context required in various real-world use case scenarios, such as socially-aware phone call application (illustrated in section 2.1.1) and cooperative convoy application (illustrated in section 2.2.1); and (ii) different sources of social context information such as Facebook, LinkedIn, Twitter and Google Calendar. This upper ontology can be shared, reused, and adapted to specific domains. It can be customized to represent social context information related concepts in different forms for different domains, applications, even users. It is also extensible to allow for the incorporation of new concepts and the specialisation of concepts and constraints for a particular domain or application, forming a domain-specific ontology.
Our upper ontology, called Social Context Ontology (SCOnto), defines four first-class entities, namely, Social Role, Social Relationship, Social Interaction, and Situation. These entities can be organised into a hierarchy where the root of the hierarchy is the term SCI, i.e., social context information, which is the superclass of those four entities and a subclass of the root concept owl:Thing (see Fig. 4.2). The concepts Person and Time are also incorporated as a subclass of owl:Thing, as they are associated with social context information.

To define Person, we adopt and extend the friend of a friend (FOAF) ontology which describes persons, their activities and their relations to other people and objects. In the FOAF ontology, a literal “status” is used to capture a person’s current activity, mood, location, etc. We use the term “situation” which is more general than the term “status” and most frequently used in context-aware computing domain to represent a person’s state of affairs. Moreover, to support reasoning over a person’s status at different granularity levels we change the type of “status” from data property to class and rename the term Status with Situation. We define an object property hasSituation which links the Person to the Situation class (see Fig. 4.3).

To capture a person’s situations over different time we incorporate Time ontology which connects with the Situation using the hasTimeStamp object property. The Time
ontology allows us to capture both the \textit{instant} and \textit{interval} time. We view a person’s (social) \textit{situation} as his/her condition or activity or state of affair with respect to surrounding people. For instance, a person’s ‘meeting’, ‘seminar’, or ‘watching game in stadium’ situation represents his/her status with respect to surrounding people.

In real-world, a person may have different types of relationships as she plays various roles. To identify a person in such circumstances, we introduce the concept of \textit{Social Role} which \textit{isPlayedBy} the \textit{Person}. The object property \textit{isPlayedBy} is InverseOf the property \textit{plays}, i.e.,

\[
isPlayedBy \equiv (\text{plays}^{-})
\]

The core concept of our model is the \textit{Social Relationship} which is a subclass of SCI and \textit{relates} exactly two persons that can be defined in OWL2 using the DL \cite{35} syntax as follows:

\[
\text{SocialRelationship} \sqsubseteq SCI \sqcap = 2 \text{relates} \cdot \text{Person}
\]

where those persons should be different individuals and linked with each other through \textit{relatedTo} property. That is, the \textit{relatedTo} object property type should be set to \textit{Irreflexive}, i.e.,

\[
\{o_i, o_j\} \in \text{relatedTo, } o_i \neq o_j, i \neq j
\]

and \textit{Symmetric}, i.e.,

\[
\text{relatedTo} \equiv (\text{relatedTo}^{-})
\]
The social relationship can have different properties which can be used to infer finer-grained relationships. For instance, trust in relationships can be measured through mining online social networks, which is used to accept recommendations, make purchase decisions and select transaction partners in the online community [86]. In [190], authors estimate relationship strength from users’ interaction activity to improve the range and performance of various aspects of online social networks, including link prediction, item recommendation, news-feeds, people search and visualization. As such, we incorporate strength and trust in our model as two data properties of Social Relationship, which can have a single value. Therefore, the type of these properties has been set to Functional, i.e.,

\[ \top \leq 1 \text{strength} \quad \text{and} \quad \top \leq 1 \text{trust} \]

Interaction-oriented and Connection-oriented are two social relationship types, defined as two subclasses of the Social Relationship, i.e.,

\[ \text{Interaction-oriented} \subseteq \text{Social Relationship} \]

\[ \text{Connection-oriented} \subseteq \text{Social Relationship} \]

The Interaction-oriented social relationships are used in socially-aware applications that support social interactions based on predefined agreements and constraints, for example, the cooperative convoy application. An interesting concept for describing and understanding such interaction-oriented relationship is social interaction as it mediates the interactions between social roles by defining their sequences (called conversations) and associated obligations (e.g., time constraints). A social interaction is an interaction between exactly two social roles, i.e.,

\[ \text{Social Interaction} \subseteq 2 \text{between} \cdot \text{Social Role} \]

On the other hand, The Connection-oriented social relationships describe users’ relational ties which can be further categorized as People-centric and Object-centric relationships (i.e., People-centric \( \subseteq \) Connection-oriented and Object-centric \( \subseteq \) Connection-oriented).
The \textit{Object-centric} relationship is identified between people who have shown common interests (e.g., like/tag in Facebook) or participated in common activities or become member of similar groups. The \textit{People-centric} relationship is a formal and declarative definition of a direct relationship between people. For example, a person identifies other persons as father, supervisor, school friend, etc. Such people-centric and object-centric relationships can be acquired from users’ online social networks (OSNs) and can be used by socially-aware applications that exploit social context information to adapt its behaviour, for example, the socially-aware phone call application.

The SCONto model (Fig. 4.3) can be read as follows. A Person may have Social Relationship(s); each Social Relationship relates exactly two Persons, can be classified as Interaction-oriented or Connection-oriented that can be further classified as People-centric or Object-centric; thus through a Social Relationship that may have a strength and a trust value, a Person is relatedTo another Person; a Person plays a particular Social Role; also a Person may have a Situation that hasTimeStamp Time. An Interaction-oriented relationship defines a set of permitted Social interaction between two Social Roles.

\section*{4.3 Domain-specific Social Context Ontologies}

We propose a number of domain-specific ontologies by specialising the upper ontology concepts such as Social Role, Interaction-oriented relationship, People-centric relationship, Object-centric relationship and Situation. We specialise the social role for family and work (educational organisation) domains, the interaction-oriented relationship for a cooperative convoy domain, the people-centric relationship for family, work and friend domains, the object-centric relationship for Common interest, Living address and Following-Follower domains, and the Situation for business and non-business domains.

\subsection*{4.3.1 The Social Role Ontology}

We consider domain-specific social role ontology for three domains such as family, work and cooperative convoy, as these domain are related to the application scenarios described in Chapter 2. We specialise the Social Role class/concept as \textit{Family Social Role}, \textit{Work Social Role} and \textit{Convoy Social Role} and define them as enumerations/nominals
We define the concept *Family Social Role* by enumerating its instances: father, mother, husband, wife, partner, son, daughter, sister, brother, cousin, uncle, aunt, nephew, niece, grandfather, grandmother, grandson, granddaughter, father-in-law, mother-in-law, brother-in-law, and son-in-law. Combining nominals using the *oneOf* OWL2-DL constructor, we can define the *Family Social Role* enumeration class as follows:

\[
{\text{FamilySocialRole}} \equiv \{ \text{father, mother, husband, wife, partner, son, daughter,} \\
\text{sister, brother, cousin, uncle, aunt, nephew, niece,} \\
\text{grandfather, grandmother, grandson, granddaughter,} \\
\text{father-in-law, mother-in-law, brother-in-law,} \\
\text{son-in-law} \}
\]

We define those instances based on our understanding of social role in the family domain and the available family role types in Facebook. Facebook allows people to maintain a family list, where a user can annotate a person in his family list as one of the
instances of the *Family Social Role* listed above.

Similarly, we define the *Work Social Role* class for a university domain by enumerating its instances: student, research-assistant, post-doctoral-fellow, lecturer, senior-lecturer, associate-professor, professor, faculty-director, centre-director, manager, dean, deputy-dean, vice-chancellor and chancellor, which can be expressed as follows:

\[
\text{WorkSocialRole} \equiv \{ \text{student, research-assistant, post-doctoral-fellow, lecturer, senior-lecturer, associate-professor, professor, faculty-director, centre-director, manager, deputy-dean, dean, vice-chancellor, chancellor} \}
\]

We define the *Convoy Social Role* by enumerating its instances: leading-car, following-car, rented-car, traffic-management, travel-guide, rental-company, and roadside-assistance, which can be expressed as follows:

\[
\text{ConvoySocialRole} \equiv \{ \text{leading-car, following-car, rented-car, traffic-management, travel-guide, rental-company, roadside-assistance} \}
\]

We also define the constraint related to these social roles, i.e., all those social roles are different individuals:

\[
\text{father} \neq \text{mother}, \ldots, \text{student} \neq \text{professor}, \ldots, \text{leading-car} \neq \text{following-car}, \ldots
\]

### 4.3.2 The Connection-oriented Social Relationship Ontology

The *Connection-oriented* relationships describe users’ relational ties which we further categorised into *People-centric* (Section 4.3.2.1) and *Object-centric* (Section 4.3.2.2) relationships.
4.3.2.1 **The People-centric Social Relationship Ontology**

The *People-centric* relationship is a formal and declarative definition of a direct relationship between people. We specialise the *people-centric* relationship (see Fig. 4.5) to propose a fine-grained relationship model for *Family*, *Work* and *Friend* relationships.

![Figure 4.5: A domain-specific people-centric relationship ontology](image)

### Family

There are a number of online social networks, such as Facebook and Google+, that allow users to specify family related connections. We use Facebook to deduce family relationships as it is the most popular and provides the richest functionality to maintain a family list. In general, such family relationships can also be extracted and consolidated from different sources.

Facebook allows users to maintain a family list, where a user can annotate a person in his family list as brother, father, uncle and so on, which ultimately indicates the role played by the person being annotated from the user’s point of view rather than the specific relationship between the user and that person (as shown in the screenshot of Fig. 4.6). Based on these list of social roles and our understanding of family domain, we classify the *Family* relationship (see Fig. 4.7) as *Parent-Child*, *Parent_in_law-Child_in_law*, *Grandparent-Grandchild*, *Uncle-Nephew*, *Aunt-Nephew*, *Uncle-Niece*, *Aunt-Niece*, *Sibling*, *Cousin* and *Partner*. To provide a fine-grain family relationship model, we further classify the *Parent-Child* relationship as *Father-Child* and *Mother-Child*. The *Father-Child* relationship is further classified to the *Father-Son* and *Father-Daughter* relationships. Similarly, the *Mother-Child* relationship is classified to the *Mother-Son* and *Mother-Daughter* relationships.
Figure 4.6: Identifying family members in Facebook
Figure 4.7: A *family* relationship ontology
Like the Parent-Child relationship, we also classify the Grandparent-Grandchild and Parent_in_law-Child_in_law relationships. We classify the Sibling as Brother-Brother, Brother-Sister and Sister-Sister relationships.

To keep the ontology consistent, it is required to define constraints related to the associated concepts in the ontology. For instance, a relationship between two persons can not be both the Father-Son and Father-Daughter, i.e., the Father-Son and Father-Daughter concepts should be disjoint; a person can have only one father, i.e., a person can have only one type of Father-Son relationship with another person who plays the father role. We define those constraints using the OWL2-DL as follows:

\[
\text{Father-Son} \sqcap \text{Father-Daughter} = \bot \\
\text{Father-Son} \sqsubseteq = 1 \text{relates(Person \sqcap \text{plays \cdot father})}
\]

Similarly, we define associated constraints related to the other concepts in the ontology. A complete constraint specification list can be found in Appendix A.2.

Friend. We classify the friend relationship (see Fig. 4.8) as CloseFriend and Education-BasedFriend. Facebook recognises users’ CloseFriend by allowing users to explicitly add friends in the close friend list (as shown in the screenshot of Fig. 4.9). A person is an EducationBasedFriend if he/she is studied with the user. We further categorize the EducationBasedFriend as PrimarySchoolFriend, HighSchoolFriend, CollegeFriend and GraduateSchoolFriend. From LinkedIn and Facebook, we can get studied institutions of a user and the persons connected with that user. Based on the institution name/id and type, we can deduce their specific type of friend relationship.

Two persons can be studied in a same school, college and university. Also they can be close friends. Thus, we do not define any disjoint constraints for the concepts of friend ontology.

Work. We classify the work relationship (see Fig. 4.10) for an educational organisation as Student-Staff and Colleague (i.e., the relationship between staff) relationships (the relationship between students is already captured in EducationBasedFriend relationship).
CHAPTER 4. SOCIAL CONTEXT MODELLING

Figure 4.8: A friend relationship ontology

Figure 4.9: Facebook recognises close friend relationship
From LinkedIn and Facebook, we can get the current and the past employment organisations of a user and all the persons connected with that user. Based on the organisation name or id and the positions held by them, we can deduce their work relationships. For instance, if two persons hold any of the staff roles in an organisation, we can deduce their relationship as being Colleague. Based on the specific types of staff roles, we can further classify the relationship as: AcademicStaff-AdminStaff, AcademicStaff-AcademicStaff, AdminStaff-AdminStaff. Similarly, we can classify the Student-Staff relationship as Student-AcademicStaff and Student-AdminStaff. A student can be supervised by an academic staff in relation to his/her project or thesis. To capture such specific relationship, we further classify the Student-AcademicStaff as Student-Supervisor and Student-Teacher.

![Figure 4.10: A work relationship ontology](image)

In an educational organisation, a person can play both an academic staff and administrative staff roles. Also a student can play an academic staff role. For instance, a person can be a professor as well as a faculty dean; a student can be involved in tutoring and/or work as a research assistant. Thus, there is no disjoint constraint on the relevant concepts in the work ontology.
4.3.2.2 The Object-centric Social Relationship Ontology

The Object-centric relationship (see Fig. 4.11) is identified between people who have shown common interests or participated in common activities or become member of same groups. We classify the Object-centric relationship as Common-Interest, Following-Follower and Living-Address relationships.

![Object-centric relationship ontology](image)

Figure 4.11: An object-centric relationship ontology

CommonInterest includes the relationships with person(s) who like the same Digital-Content (e.g., those people use “like” to express in Facebook), are members of the same Group (e.g., soccer, cricket), are interested in the same Event (e.g., conference), or have CommonResearchInterest which can be collected from LinkedIn.

From Twitter we can obtain the Following-Follower relationship, where the Following includes the person(s) whom the user is interested and following, and the Follower includes person(s) who are following the user.

Living-Address based relationship can be acquired from Facebook or LinkedIn, and further categorized at Living-Country and Living-City levels.
4.3.3 The Interaction-oriented Social Relationship Ontology

The interaction-oriented relationship is formed between collaborative individuals who are obliged to interact with each other. In some cases, such a relationship is based on an explicit agreement; in other cases, participants behave as though they agree to the relationship without a formal agreement. For example, in the cooperative convoy scenario (discussed in Section 2.2.1), when the group of tourists rent the cars from Budget and AVIS, they are often required to sign a rental contract. The contract defines key aspects of the relationship between the customer and the rental company. In another case, the convoy agreements between the drivers of these rented cars is less formal. Fig. 4.12 shows the interaction-oriented relationships ontology for the cooperative convoy scenario. The concepts hierarchy is represented as follows:

![Interaction-oriented Social Relationships Ontology](image)

**Figure 4.12:** An interaction-oriented social relationships ontology for cooperative convoy

**Social Interaction Ontology** The social interactions are defined by the interaction-oriented relationships between collaborative individuals playing social roles. These interaction-relationships are used to construct runtime social interaction models that mediate runtime interactions (see Section 6.2). Social interaction captures social norms, rules and obligations between social roles, which we represent in terms of interactions, conversations and obligations (see Fig. 4.13).
Interactions define permitted atomic message exchange between the social roles. The properties of an interaction include an identifying message signature, a direction of the message (i.e., fromRole and toRole) and the interaction type.

Conversation defines an acceptable sequence of interactions defined by temporal constraints in the interaction rule specification language [113]. For instance, a conversation, say $c_1$, relates the interactions $i_1$ and $i_2$ linked with the intrac1 and intrac2 object properties respectively, using the sequence type precedes (i.e., before); it means that the $i_1$ should occur before $i_2$.

Obligation defines time constraints in a conversation using an operational parameter. The operational parameter includes a parameter name, a time unit or distance unit and a value. The obligation with a timer “duration” and a specific operational parameter value, constraint the maximum allowed time duration between the interactions in the associated conversation. For example, an operational parameter value “2min” means that the maximum allowed delay between two interactions is two minutes. On the other hand, the obligation with a timer “period” and a specific operational parameter value, constraint the frequency of the associated conversation. For example, a obligation with a timer “period” and “2min” operational parameter value means that the associated conversation should occur at least once in every two minutes.

In the cooperative convoy scenario, both the leading car and following car need to send their positions to each other in every 10 seconds. Table 4.4 represents (in Turtle syntax [64]) the $i_1$ interaction, i.e., leading car notifies its position to the following car, while Table 4.5 represents the $i_2$ interaction, i.e., following car notifies its position to the leading car. Table 4.6 represents the obligation associated with these two interactions, i.e., the position update should be every 10 seconds.

**Table 4.4:** Interaction ($i_1$): Leading car notifies its position to the following car

<table>
<thead>
<tr>
<th>:i1</th>
<th>rdf:type</th>
<th>:Interaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>:i1</td>
<td>:msgSignature</td>
<td>&quot;notifyPosition&quot;</td>
</tr>
<tr>
<td>:i1</td>
<td>:fromRole</td>
<td>:LeadingCar</td>
</tr>
<tr>
<td>:i1</td>
<td>:toRole</td>
<td>:FollowingCar</td>
</tr>
</tbody>
</table>
Figure 4.13: A social interaction ontology
There are a number constraints related to the specification of social interaction ontology. A social interaction consists of at least one interaction, occurs between exactly two social roles. A conversation should impose a sequence - either precedes or leadsTo[113] - between exactly two interactions linked by the intrac1 and intrac2 object properties. An interaction should have exactly one toRole, one fromRole and one msgSignature properties. An obligation should have exactly one timer (either duration or period) and one relatedOp property. An operational parameter should have exactly one paramName, one value and one timUnit ("MS" or "SEC" or "MIN" or "HOUR" or "DAY") or one distanceUnit ("MM"
or “CM” or ‘M’ or “KM”). These constraints are represented as follows:

\[
\begin{align*}
\text{Social-Interaction} & \sqsupseteq \geq 1 \text{ hasIntrac \cdot Interaction} \\
\text{Social-Interaction} & \sqsubseteq = 2 \text{ between \cdot Social-Role} \\
\text{Conversation} & \equiv \{\text{precedes, leadsTo}\} \\
\text{Conversation} & \sqsubseteq = 1 \text{ intrac1 \cdot Interaction} \\
\text{Interaction} & \sqsubseteq = 1 \text{ toRole \cdot Social-Role} \\
\text{Interaction} & \sqsubseteq = 1 \text{ fromRole \cdot Social-Role} \\
\text{Interaction} & \sqsubseteq = 1 \text{ msgSignature} \\
\text{timer} & \in \{\text{DURATION, PERIOD}\} \\
\text{Obligation} & \sqsubseteq = 1 \text{ timer} \\
\text{Obligation} & \sqsubseteq = 1 \text{ relatedOp \cdot Operational-Parameter} \\
\text{timeUnit} & \in \{\text{MS, SEC, MIN, HOUR, DAY}\} \\
\text{distanceUnit} & \in \{\text{MM, CM, M, KM}\} \\
\text{Operational-Parameter} & \sqsubseteq \leq 1 \text{ timeUnit} \\
\text{Operational-Parameter} & \sqsubseteq \leq 1 \text{ distanceUnit} \\
\text{Operational-Parameter} & \sqsubseteq = 1 \text{ paramName} \\
\text{Operational-Parameter} & \sqsubseteq = 1 \text{ value}
\end{align*}
\]

### 4.3.4 The Situation Ontology

A situation ontology is a logical specification and representation of situation related concepts that are of application interest \[193\]. In this thesis context, we view situation as a user status with respect to his/her surrounding people. For example, a user’s situation is ‘meeting’, ‘seminar’ or ‘watching game in a stadium’ describes his/her status with respect to the surrounding people.

We categorise situation as business and non-business situations considering their use in application scenarios (described in Chapter \[2\]). We further categorise business situation as ‘meeting’, ‘seminar’ and ‘Lecture’, and non-business situation as ‘watching game in a stadium’ and ‘driving in convoy’ (see Fig. \[4.14\]).
Most of the business situations are very formal and scheduled. Such situations of users can be acquired from online digital calendar, e.g., Google Calendar - assuming that the users maintain their schedules through the Google calendar, or can be deduced from time instant (see Section 5.3). This type of situation is very useful for many socially-aware applications. For example, in the socially-aware phone call application, a ‘meeting’ situation is an important factor for a user to filter an incoming phone call; also this situation is a vital factor for a caller to make decision about the suitable time of making call.

On the other hand, non-business situations are usually occasional. Many of such situations can be inferred from users’ interaction activities in online social networks. For instance, ‘watching game in a stadium’ can be inferred from user interactions in Facebook (see Section 5.4 for details). This situation can be used to automate mobile settings, e.g., automatically change the ring tone and ring volume in mobile phones, as the stadium environment usually very noisy and a person may prefer to set their phone ring tone volume in a highest level.

We specify a number of constraints in the situation ontology. At the same time, a person can not be in any of the two non business situations. Also, it is not possible to watch game in a stadium and drive, at the same time. These constraints are represented
as follows:

\[
\begin{align*}
\text{Meeting} & \sqcap \text{Seminar} \not\sqsubseteq \\
\text{Meeting} & \sqcap \text{Lecture} \not\sqsubseteq \\
\text{Seminar} & \sqcap \text{Lecture} \not\sqsubseteq \\
\text{Watching-Game-in-Stadium} & \sqcap \text{Driving-in-Convoy} \not\sqsubseteq 
\end{align*}
\]

4.4 Summary

In this chapter, we have presented an ontology based approach to modelling social context information. We have presented both the upper and domain-specific ontologies for different types of social context information such as social role, social relationship, social interaction and situation. The upper ontology specifies general concepts, their semantics and related constraints, while the domain-specific ontologies extend and classify general concepts to specify domain-specific concepts, their semantics and related constraints. We use the OWL2-DL, an ontology representation language, for encoding the concepts hierarchies, their disjointness and constraints on the properties of those concepts. One of the benefits of using OWL2-DL is that existing DL-based reasoners can be used for checking inconsistency and integrity of knowledge.

The key advantages of the ontology-based model are that it is reusable and extensible, and provides a basis for developing a knowledge base to store and manage context information. Furthermore, the ontology-based modelling and OWL-DL based representation provide a basis for composing new concepts and inferring new social context information (see Chapter 5).
Chapter 5

Social Context Reasoning

In the previous chapter, we have introduced different types of social context information and present ontologies that represent them. However, some useful social context information in the ontologies (e.g., Father-Daughter, best friend) cannot be acquired directly from the possible sources, but can be derived from the acquired data. Moreover, new concepts (e.g., Group Meeting, Busy), as required by applications, can be deduced and abstracted from the existing concepts. Furthermore, users’ social interactions can be exploited to infer their situations which can be used to enhance applications’ functionalities.

To address the above requirements, in this chapter, we present our approach to deriving different types of social relationships (section 5.2), and deducing and abstracting situations from acquired information. We also present our approach to inferring situations from users’ interaction events (section 5.3).

5.1 Reasoning Approach

Reasoning is the process of identifying meaningful facts from observed data [41,193]. There are four aspects of reasoning about social context information:

Structure and Property Social context information can be derived from a set of other social context information and their properties. For example, a ‘best friend’ re-
A person’s relationship with another person can be derived from their social roles and gender, for example, a ‘Father-Daughter’ relationship between two persons can be derived from a person’s corresponding social role, e.g., ‘Father’ and the other person’s gender information, here ‘female’.

**Composition/Abstraction** Social context information can be composed of or abstracted from a set of other social context information. For example, a ‘busy’ situation abstracts the ‘seminar’, ‘lecture’ and ‘meeting’ situations.

**Time-stamp** Social context information might be related with a time-stamp, i.e., time of day, duration, frequency. A situation can be deduced from the current time-stamp as - it may only happen at a particular time of a day, it may only last a certain length of time, it may only happen a certain times per week. For example, a person have a lecture on every Tuesday 9:30am to 11:30am till May 30 except public holidays.

**Temporal correlation** Social context information can be inferred from the temporal correlation of observed facts. For example, a ‘watching game’ situation can be inferred from a person’s location and consequent message updates in social media such as posting current location as stadium, posting a game related picture and/or commenting on a game performance.

In this thesis, we address the above four aspects of social context information reasoning. As such, we view reasoning as a process of deriving, deducing, abstracting and inferring social context information that is of interest to applications. We adopt a specification based technique in our reasoning approach. In the specification based technique, domain knowledge is represented in rules and reasoning engine is applied to infer facts from observed data. There are number of approaches proposed for the specification based technique. These approaches have developed from earlier attempts in first-order logic \[149, 185\] to a OWL-DL based model \[154\] that aims to support efficient reasoning while keeping expressive power, to support formal analysis, and to maintain the soundness and completeness of a logical system.
Ontologies have been widely applied \[41, 57, 151, 192\] for its powerful representation and reasoning capabilities. Ontologies can provide a standard vocabulary of concepts to represent domain knowledge, specifications and semantic relationships of context information defined in formal logic approaches. They can also provide fully fledged reasoning engines to reason on them following rules specified in formal logic approaches. Thus, we adopt an ontology based approach to deriving, abstracting and deducing social context information considering their structural/property, compositional/abstraction and time-stamp aspects. In particular, we use OWL 2 DL \[88\] to represent and specify reasoning rules.

To date, OWL 2 DL\(^1\) has been the most practical choice for most ontological applications as it supports maximum expressiveness while retaining computational completeness and decidability \[154\]. OWL-DL class constructors such as \(\text{intersectionOf}\ (\cap)\), \(\text{unionOf}\ (\cup)\), \(\text{complementOf}\ (\neg)\), \(\text{oneOf}\ (\{\})\), \(\text{allValuesFrom}\ (\forall)\), \(\text{someValuesFrom}\ (\exists)\), \(\text{maxCardinality}\ (\leq)\), \(\text{minCardinality}\ (\geq)\), \(\text{exactCardinality}\ (\equiv)\) and \(\text{hasValue}\ (\ni)\) can be used to construct new classes/concepts from existing named classes, properties and individuals. OWL-DL class semantics such as \(\text{equivalentTo}\ (\equiv)\), \(\text{subclassOf}\ (\sqsubseteq)\) and \(\text{disjointWith}\ (\sqcap = \bot)\) can be used to specify associations between classes (named or a new class). In addition, OWL property characteristics such as \(\text{Transitive}\), \(\text{Symmetric}\), \(\text{Asymmetric}\), \(\text{Reflexive}\), \(\text{Irreflexive}\), \(\text{Functional}\), \(\text{Inverse functional}\) and \(\text{InverseOf}\) can be used to specify further associations among classes based on their individuals. Details of these constructors are discussed in Sections 3.1.3 and 4.1.

As such, we use the OWL-DL class constructors, semantics and property characteristics to impose constraints on named classes in the ontologies and to specify reasoning/inference rules for defined classes. The main advantages of their use is that a DL reasoner can be used for (1) executing inference rules, and (2) detecting automatically inconsistencies in the specification of those inference rules.

To address the temporal correlation aspect in situation reasoning, we propose a hybrid approach that combines ontology-based semantic matching and complex event processing in order to infer users’ situations from observed data, i.e., users’ interaction events. In this regard, we have developed an ontology-based interaction event model to capture users’

\(^1\)for the sake of brevity we use OWL-DL later on
interactions in various social interaction platforms, which adopts and extends recent event models and incorporates social context information. We also propose a technique to specify situations using atomic and complex interaction event patterns, which integrates their semantic aspect with temporal relations.

5.2 Deriving Social Relationships

In this section, we present DL-based reasoning rules to derive different types of family relationship from acquired social context information (Section 5.2.1), and to derive new types of friend relationship based on the existing friend relationship types and their properties (Section 5.2.2).

5.2.1 Deriving Family Relationships

In Section 4.3.2.1, we have presented a family relationship ontology that captures different types of family relationships. However, such relationships information cannot be acquired directly from sources, but can be derived based on acquired social roles and gender information. We rely on users’ Facebook information to deduce Family relationships. Facebook allows people to maintain a family list, where a user can annotate a person in his family list as brother, father, uncle and so on, which ultimately indicates the role played by the person being annotated from the user’s point of view rather than the specific relationship between the user and that person. For instance, if a user annotates a person in his/her family list as “father”, the relationship between them could be a “Father-Son” or “Father-Daughter” relationship, which can be inferred and differentiated by utilising the user’s gender information. Some examples are given below:

\[
\text{Father-Daughter} \equiv ( \exists \text{relates}(\text{Person} \cap \text{plays} \cdot \{\text{father}\}) \sqcap \\
\exists \text{relates}(\text{Me} \cap \text{gender} \cdot \{\text{female}\})) \\
\sqcup ( \\
\exists \text{relates}(\text{Person} \cap \text{plays} \cdot \{\text{daughter}\}) \sqcap \\
\exists \text{relates}(\text{Me} \cap \text{gender} \cdot \{\text{male}\}))
\] (5.1)
Uncle-Nephew \equiv ( \exists \text{relates}(\text{Person} \sqcap \text{plays} \cdot \{\text{uncle}\}) \sqcap \\
\exists \text{relates}(\text{Me} \sqcap \text{gender} \cdot \{\text{male}\})) \\
\sqcup ( \\
\exists \text{relates}(\text{Person} \sqcap \text{plays} \cdot \{\text{nephew}\}) \sqcap \\
\exists \text{relates}(\text{Me} \sqcap \text{gender} \cdot \{\text{male}\}))

Here, Father-Daughter and Uncle-Nephew relationships between a user and another person are inferred from the user’s gender information (a data property of foaf:Person) and the related person’s social role. Me is a subclass of Person that specifies the user from whose perspective the relationship is computed.

A complete list of the reasoning rules can be found in Appendix A.1.

5.2.2 Deriving Friend Relationships

In section 4.3.2.1 we have presented a friend ontology that captures different types of friend relationship. Such basic friendship concepts can be used to further derive friendship relationships such as BestFriend, BosomFriend, ChildhoodFriend and LongtimeFriend. Fig. 5.1 shows the inferred friend relationship ontology. The specialised friend relationships might be of interest to different socially-aware applications. For example, in the socially-aware phone call application, a user might want to filter incoming phone calls or specify her privacy preference based on the “best friend” relationship type.

Figure 5.1: An inferred friend relationship ontology
According to an application requirements, developers can specify reasoning rules for deriving such relationships. For instance, a best friend could be specified as a specially close and trusted friend [22]; a bosom friend is a very close friend [22], i.e., a close friend with a high relationship strength, for example, greater than or equal to 0.8; a person is a childhood friend if he/she is primary school or high school friend; a person can be identified as a long time friend if he/she is a PrimarySchoolFriend, HighSchoolFriend, CollegeFriend and GraduateSchoolFriend. We specify these reasoning rules using OWL-DL constructs as follows:

\[
\text{BestFriend} \equiv \text{CloseFriend} \cap (\text{trust} > \{0.0\}) \\
\text{BosomFriend} \equiv \text{CloseFriend} \cap (\text{strength} \geq \{0.8\}) \\
\text{ChildhoodFriend} \equiv \text{PrimarySchoolFriend} \sqcup \text{HighSchoolFriend} \\
\text{LongtimeFriend} \equiv \text{PrimarySchoolFriend} \cap \text{HighSchoolFriend} \cap \\
\text{CollegeFriend} \cap \text{GraduateSchoolFriend}
\]

5.3 Deducing and Abstracting Situations

Situation reasoning is the process of identifying situations from acquired information. In this section, we present DL-based reasoning rules to deduce situations from acquired information by considering their time-stamp aspect (Section 5.3.1). The deduced situations are further used to define more abstract situations (Section 5.3.2).

5.3.1 Deducing Situations

In section 4.3.4, we have presented ‘meeting’, ‘seminar’ and ‘lecture’ situations that are related to the business/office environments. We have argued that most of these situations occur periodically and follow a fixed schedule. For instance, in a university, course lectures are scheduled before starting of each semester, e.g., a teaching staff have a lecture on every Tuesday 9:30am to 11:30am; a faculty may run a weekly seminar in a specific time and week day, e.g., every Wednesday 10:30am to 11:30am; a research staff might need to attend to a weekly research group meeting which is also usually scheduled for whole semester, e.g., every Monday 10am to 11am during the semester; a research staff
also may have a weekly individual meeting with his/her research students, e.g., every Monday 2pm to 3pm; research group members may go for a group lunch every Thursday. Such scheduled situations of a user can be defined using time-stamp property of the corresponding situations, and be deduced at runtime.

Our Situation ontology is linked with owl:Time ontology using hasTimeStamp object property, as shown in Fig. 5.2. Fig. 5.3 shows an excerpt of owl:Time ontology [104, 105]. We extend the owl:Time ontology by incorporating and defining a number domain-specific concepts such as Weekends, Weekdays, NonBusinessHours, BusinessHours, WorkingHours, LunchHours and Holidays. These concepts will be used in inference rules to define the aforementioned scheduled situations. We define these concepts based on the general rules in Australia [4]. To define Holiday we use the Victorian public holidays 2013 [6]. However, these can be modified according to the application domains. We define Saturday and Sunday as Weekends; Monday to Friday as Weekdays; 9:00 to 16:59 is BusinessHours while 9:00 to 11:59 and 13:00 to 16:59 are WorkingHours, and 12:00 to 12:59 is LunchHours; 17:00 to 23:59 and 00:00 to 8:59 are NonBusinessHours. We further categorize Holidays as NewYearDay (Tuesday 1 January), EasterMonday (Monday 1 April), ChristmasDay (Wednesday 25 December), and so on.

Fig. 5.4 shows the inferred concepts hierarchy in the time ontology. The inference rules of these concepts are represented using OWL-DL as follows:

\[
\text{Weekends} \equiv \exists \text{inDateTime} (\text{DateTimeDescription} \sqcap (\text{dayOfWeek} \cdot \{\text{Saturday}\} \sqcup \text{dayOfWeek} \cdot \{\text{Sunday}\}))
\]
CHAPTER 5. SOCIAL CONTEXT REASONING

Figure 5.3: An excerpt of *owl:Time* ontology

Figure 5.4: An inferred *owl:Time* ontology
Weekdays $\equiv \exists inDateTime(DateTimeDescription \cap$
\(\{\text{dayOfWeek} \cdot \{\text{Monday}\}\} \cup \{\text{dayOfWeek} \cdot \{\text{Tuesday}\}\} \cup \{\text{dayOfWeek} \cdot \{\text{Wednesday}\}\} \cup \{\text{dayOfWeek} \cdot \{\text{Thursday}\}\} \cup \{\text{dayOfWeek} \cdot \{\text{Friday}\}\})
\]
(5.2)

Holidays $\equiv \text{NewYearDay} \cup \text{ChristmasDay} \cup \text{BoxingDay}$
\(\cup \text{GoodFriday} \cup \text{LabourDay} \cup \text{AustraliaDay} \cup \text{QueenBrithDay} \cup \text{ANZACDay} \cup \text{EsaterMonday} \cup \text{MelbourneCupDay} \cup \text{SaturdayBeforeEasterSunday}$

NewYearDay $\equiv \exists inDateTime(DateTimeDescription \cap$
\(\{\text{year} \cdot \{2013\}\} \cap \{\text{month} \cdot \{1\}\} \cap \{\text{day} \cdot \{1\}\})
\]

ChristmasDay $\equiv \exists inDateTime(DateTimeDescription \cap$
\(\{\text{year} \cdot \{2013\}\} \cap \{\text{month} \cdot \{12\}\} \cap \{\text{day} \cdot \{25\}\})
\]

BusinessHours $\equiv \text{Weekdays} \cap (\neg \text{Holidays}) \cap$
\(\exists inDateTime(DateTimeDescription \cap (\text{hour} \geq \{9\} \cap \text{hour} \leq \{16\}))
\]
(5.3)

LunchHours $\equiv inDateTime(DateTimeDescription \cap \text{hour} \cdot \{12\})$

WorkingHours $\equiv BusinessHours \cap (\neg LunchHours)$

NonBusinessHours $\equiv \neg BusinessHours$

Some constraints also exist relating to these concepts. For instance, business hours and non business hours are disjoint with each other; similarly, weekdays and weekends should not overlap. These constraints are represented as follows:

$$BusinessHours \cap NonBusinessHours \sqsubseteq \bot$$
$$Weekdays \cap Weekends \sqsubseteq \bot$$

A complete list of the inference rules and constraints can be found in Appendices $\text{A.1}$
Example. Let us consider a time instant, May 20 2013 Monday 2:30pm. Fig. 5.5 shows how this time instant (called moment) is inserted to the owl:time ontology (as an instance, called time) and is classified to domain-specific concepts (moment is WeekDays, BusinessHours and WorkingHours) using the inference rules 5.2, 5.3 and 5.4.

Figure 5.5: Classifying the time instant (May 20 2013 Monday 2:30pm) to domain-specific concepts

Based on the above time related concepts, we can specify GroupLunch, Lecture, Fac-
ultySeminar, GroupMeeting and IndividualMeeting situations as follows:

\[ \text{GroupLunch} \equiv \exists \text{hasTimeStamp} (\neg \text{Holidays} \sqcap \text{LunchHours} \sqcap \\
\exists \text{inDateTime} (\text{DateTimeDescription} \sqcap \\
\text{dayOfWeek} \cdot \{\text{Thursday}\})) \]

\[ \text{Lecture} \equiv \exists \text{hasTimeStam}(\text{WorkingHours} \sqcap \\
\exists \text{inDateTime} (\text{DateTimeDescription} \sqcap \\
((\text{hour} \cdot \{9\} \sqcap \text{minute} \geq \{30\}) \sqcup \\
(\text{hour} \cdot \{11\} \sqcap \text{minute} \leq \{30\})) \\
\sqcap \text{dayOfWeek} \cdot \{\text{Tuesday}\})) \]

\[ \text{FacultySeminar} \equiv \exists \text{hasTimeStam}(\text{WorkingHours} \sqcap \\
\exists \text{inDateTime} (\text{DateTimeDescription} \sqcap \\
((\text{hour} \cdot \{10\} \sqcap \text{minute} \geq \{30\}) \sqcup \\
(\text{hour} \cdot \{11\} \sqcap \text{minute} \leq \{30\})) \\
\sqcap \text{dayOfWeek} \cdot \{\text{Wednesday}\})) \]

\[ \text{GroupMeeting} \equiv \exists \text{hasTimeStam}(\text{WorkingHours} \sqcap \\
\exists \text{inDateTime} (\text{DateTimeDescription} \sqcap \\
(\text{hour} \cdot \{10\} \sqcup (\text{hour} \cdot \{11\} \sqcap \text{minute} \cdot \{0\})) \\
\sqcap \text{dayOfWeek} \cdot \{\text{Monday}\})) \]

\[ \text{IndividualMeeting} \equiv \exists \text{hasTimeStam}(\text{WorkingHours} \sqcap \\
\exists \text{inDateTime} (\text{DateTimeDescription} \sqcap \\
(\text{hour} \cdot \{14\} \sqcup (\text{hour} \cdot \{15\} \sqcap \text{minute} \cdot \{0\})) \\
\sqcap \text{dayOfWeek} \cdot \{\text{Monday}\})) \] \hspace{1cm} (5.5)

**Example.** Fig. 5.6 shows how Alice’s situation is deduced as *individual meeting* on the *moment* time instant using reasoning rule 5.5.
5.3.2 Abstracting Situations

More abstract general situations (in a business domain) can be defined based on the above time-related situations from users’ location perspective. For instance, a user’s group lunch, lecture, group meeting and faculty seminar situations can be abstracted to ‘not at desk’ situation. Usually there are particular places for these activities and they are not held in a person’s office or at desk. On the other hand, a person’s individual meetings with another person might held in his/her office. Thus, the individual meeting and chatting with friends situations can be abstracted to ‘at desk’ situation. The inference rules for these situations are as follows:

\[
\text{NotAtDesk} \equiv \text{GroupLunch} \sqcup \text{Lecture} \sqcup \text{Seminar} \sqcup \text{GroupMeeting}
\]

\[
\text{AtDesk} \equiv \text{IndividualMeeting} \sqcup \text{Chatting}
\]

The time-related situations also can be abstracted to define user-specific situations. For example, a user might say her status as being ‘busy’ when she is in a lecture or group seminar or group meeting or individual meeting or group lunch. Furthermore, user may want to define situations from an application perspective. For example, a user might prefer not to be interrupted by any phone call when she is busy, but she may
tolerate only urgent phone call when she is in individual meetings or a group lunch. Identifying and making available users’ such situations allow a caller to take decision of her making a phone call. The inference rules for abstracting such situations are given below:

\[
\begin{align*}
    \text{Busy} & \equiv \text{GroupLunch} \sqcup \text{Lecture} \sqcup \text{Seminar} \sqcup \text{Meeting} \\
    \text{PhoneCallNotExpected} & \equiv \text{Busy} \\
    \text{OnlyUrgentCall} & \equiv \neg \text{Busy} \sqcup \text{GroupLunch} \sqcup \text{IndividualMeeting}
\end{align*}
\]

Fig. 5.7 shows the concepts hierarchy in the inferred business situation ontology with granularity level. We have set the granularity level for each concept in the ontology as an increasing number from root to leaf nodes. Thus for a given situation, we can answer the situation of a user at different levels of granularity. For instance, for a situation “IndividualMeeting” with granularity level 2, we can say that the user is “Busy”.

![Figure 5.7: An inferred business situation ontology](image-url)
CHAPTER 5. SOCIAL CONTEXT REASONING

5.4 Inferring Situations from Interaction Events

The advances of internet technologies and the increased prevalence of advanced mobile devices (the so called “smart” phones), have radically changed the way people perform tasks, interact with each other and share information about their lives and works. In order to facilitate such device mediated interactions different social (interaction) platforms or applications have emerged.

Some of the applications are developed for specific tasks, and based on predefined agreements and constraints, e.g., the cooperative convoy application (see Section 2.2). Users interactions in these applications generate interaction events.

On the other hand, users’ interactions using online social networks, instant messaging and email are open and not based on predefined agreements and constraints. On these platforms people interact with each other in many ways: sharing their interests; tagging, bookmarking and commenting on digital contents such as photos and videos; chatting and sending emails; and posting messages about what they did, what they are currently doing or what their plans are for the (near) future [188]. Thus, it is now possible to acquire users’ diverse context information from social platforms which may be rarely possible to acquire from physical sensors [158]. This phenomenon offers a unique opportunity and motivates us to exploit and combine users’ interaction events across different social interaction platforms (e.g., Facebook, Twitter, Email) to infer situations.

We view an interaction event as a user’s interaction/activity in a social platform at a certain time or over a period of time. As such, the interaction events have temporal correlations. Furthermore, they are associated with users’ social context information such as their social relationships and social roles which are captured in the properties of interaction events. Many of these properties are ontological, i.e., their values have underlying semantics. Thus, inferring situations in social interaction platforms needs to consider semantic matching of interaction event properties and temporal relations among historical events.

To address the above issue, we propose an ontology based model, called Interaction Event Ontology (IntEO), to capture users’ (social) interactions with their connections.
(e.g., friends, family members, colleagues) in different social platforms where an interaction refers to like/promoting/commenting on/sharing/posting a status message, current location, music, photo, video, link, and so on. This interaction also includes chatting in instant messenger, sending email, or any type of mediated interactions.

In order to capture the general concepts and incorporate domain-specific concepts, we propose both the upper and domain-specific ontologies. The upper ontology captures the general concepts, abstracted from the analysis of possible interaction events in different platforms (see Section 5.4.1). This upper ontology can be shared, reused, and adapted to specific domains. It is also extensible to allow for the incorporation of new concepts and the specialisation of concepts and constraints for a particular social platform, forming a domain-specific ontology (see Sections 5.4.2 and 5.4.3). We also propose a language to specify situations using atomic and complex interaction event patterns, which integrates semantic aspect of interaction events with their temporal relations (see Sections 5.4.5 and 5.4.6).

5.4.1 Interaction Event Upper Ontology

The core event ontology is the basis of our interaction event upper ontology, and is formulated based on the LODE and SEM models (see Section 3.4) as they are most recently proposed event ontologies and focus on generality and interoperability. The core event ontology (see upper part of the Fig. 5.8), has four key entities/concepts/classes: Event, Time, Place and Thing.

To model the interaction event upper ontology (see lower part of the Fig. 5.8), we extend the core event ontology with concepts concerning interaction events and incorporate our social context ontology (SCOnto) (see Section 4.2). Our upper ontology defines eight first-class entities, namely, InteractionEvent, Actor, SocialRole, SocialRelationship, Topic, InteractionType, SocialInteractionPlatform and TemporalEntity. InteractionEvent is the core class which extends (specialisation of/sub-class of) the Event class in the core event ontology. The upper ontology inherits the Place from the core event ontology. To define the Time in the upper ontology, we specialise the Time class in core event ontology by adopting the TemporalEntity class from the owl:Time ontology [105]. We extend the Thing
concept in the core event ontology to define the SocialInteractionPlatform, InteractionType, Topic, SocialRelationship, SocialRole and Actor concepts in the interaction event ontology, where the SocialRelationship and SocialRole concepts are adopted from the SConTo ontology. We also define the from, to, fromRole, toRole, related, about, hasType and hasPlatform object properties by extending and specialising the associated object property of the core event ontology. These extension and specialisation such as InteractionEvent is a SubClassOf Event, Actor is a SubClassOf Thing, and fromRole is a SubPropertyOf associated, can be represented using OWL-DL as follows:

\[
\text{InteractionEvent} \sqsubseteq \text{Event} \\
\text{Actor} \sqsubseteq \text{Thing} \\
\text{fromRole} \sqsubseteq \text{associated}
\]

The upper ontology can be read as follows. An InteractionEvent is a communication from a Person (which is an Actor) acting in a SocialRole (fromRole) to another Person acting in another SocialRole (toRole). This InteractionEvent hasType InteractionType, is related to a SocialRelationship that exists among the involved Persons, is occurred over a particular Platform atTime of TemporalEntity when the involved Persons are discussing a particular Topic (about).
In our upper ontology, the *SCOnto:Person* class which is a sub-class of *Actor*, identifies all the individual persons associated with an interaction and the *SCOnto:SocialRole* class identifies all the roles played by these persons. The *from* object property links the person who initiated the interaction while the *to* object property links other persons who are directly involved in that interaction. Similarly, the *fromRole* and *toRole* object properties identify *social roles* that are played by these persons. The relationship(s) between these persons are captured in the *SCOnto:SocialRelationship* which is connected to the *InteractionEvent* through the *related* object property. The *Topic* class captures the subject/theme/purpose of an interaction. The *InteractionType* concept captures various types of actions such as posting a message, commenting on a photo, and sharing status. The *SocialInteractionPlatform* class defines the interaction platforms such as Facebook, LinkedIn, Twitter and Email.

### 5.4.2 Domain-specific Interaction Event Ontology for Cooperative Convoy

In this section, we present a domain-specific interaction event ontology for the cooperative convoy by adopting and extending the interaction event upper ontology presented in the previous section. In the cooperative convoy (discussed in Section 2.2.1), collaborative actors (*e.g.*, drivers of the cars) interact with each other to achieve their goals/objectives. Such message-based interactions are mediated through an interaction-centric socially-aware application.

In such convoy, car drivers identify themselves as a driver of a leading car or a following car. Based on the agreements among these drivers, we can define *interaction relationships* among them in terms of interactions, conversations and obligations. Interactions performed by drivers at runtime may follow some patterns among many possible valid interaction patterns, and some of these patterns might refer to different meaningful situations. By inferring such situations, it might possible to understand drivers’ needs and provide services accordingly.

During convoy, drivers may pass through different situations, *e.g.*, ‘hungry’ and thus need to find a suitable restaurant. One possible interaction pattern to infer this ‘hungry’ situation would be: *one of the drivers proposing to stop for the Food and other drivers agreed...*
on that, and it is a breakfast, lunch or dinner time. The cooperative convoy application’s functionality can be enhanced based on this situation. For instance, the application can automatically collect nearby restaurants information by invoking a restaurant searching service and recommend those restaurants to drivers.

To infer situations from drivers’ interaction activities in a cooperative convoy, first we need to define an interaction event ontology for the cooperative convoy by adopting and extending IntEO. Sections 4.3.1 and 4.3.3 present social role and (interaction-oriented) social relationship ontologies, respectively, for the cooperative convoy application. Fig. 5.9 shows how we extend the InteractionType, Topic and Time concepts of the IntEO for the cooperative convoy application.

By motivating the usage of John R. Searle’s “speech act” [170] which is often meant to refer just to the same thing as the term illocutionary act that John L. Austin had originally introduced to Do Things with Words [33], we define an InteractionType ontology for the cooperative convoy application. This ontology provides vocabulary or concepts to classify interactions among collaborative actors. For instance, we can classify the interaction “convoyBreak” to stop for the food while driving as a “proposing” interaction type. A positive response of this interaction from other drivers is classified as “agreeing” interaction type, while an negative response is classified as “disagreeing” interaction type. Fig. 5.9 shows the interaction types that we use to classify all the interactions in our cooperative scenario.

In the cooperative convoy domain, we classify Topic as food, attraction, restaurant,
fuel-station, and restroom, as these are relevant. Time when an interaction takes place is also important to identify situation. For instance, to identify hungry situation, breakfast, lunch and diner times are relevant. Thus, we define BreakfastTime (between 6am and 8am), LunchTime (between 12pm and 2pm) and DinnerTime (between 6pm and 8pm) concepts by classifying Time:DateTimeDescription concept in the IntEO, as follows:

\[
\text{BreakfastTime} \equiv \exists \text{inDateTime}(\text{DateTimeDescription} \cap (\text{hour} \geq \{6\} \cap \text{hour} \leq \{8\}))
\]

\[
\text{LunchTime} \equiv \exists \text{inDateTime}(\text{DateTimeDescription} \cap (\text{hour} \geq \{12\} \cap \text{hour} \leq \{14\}))
\]

\[
\text{DinnerTime} \equiv \exists \text{inDateTime}(\text{DateTimeDescription} \cap (\text{hour} \geq \{18\} \cap \text{hour} \leq \{20\}))
\]

In Section 8.3.6.1, we use this domain-specific interaction event ontology for inferring the ‘hungry’ situation.

5.4.3 Domain-specific Interaction Event Ontology for Facebook

We define a domain-specific interaction event ontology for Facebook by extending the interaction event upper ontology. We choose Facebook because it supports most of the different types of interactions provided by other social media applications such as Google+. Thus, our interaction event ontology for Facebook can be adopted/reused/extended for other social media platforms.

**Interaction Types** Facebook allows users to perform different types of interaction activities. To classify such interactions we extend the InteractionType class as shown in Fig. 5.10. Facebook allows users to post various digital contents such as status, current location, life event, photo, photo album, video and music. We classify a user’s status posting as having the PostingStatus interaction type. In a status, a user can say anything, i.e., whatever on her mind. Thus, it is hardly possible to comprehensively classify all possible status messages. However, in a status update, users generally mention about their current activity, i.e., what they are currently doing or going to do or have already
done, their emotion, current affairs, news, and so on [188].

We classify a user’s life event, current place, photo, photo album, music and video posting as PostingLifeEvent, PostingCurrentLocation, PostingPhoto, PostingPhotoAlbum, PostingMusic and PostingVideo, respectively. Facebook allows a user to post different types of her life events related to the various aspects of her life such as work (e.g., starting new job, retirement), education (e.g., new school, published book or paper), family (e.g., new child, new family member, new pet), relationship (e.g., engagement, marriage), home (e.g., bought home), living (e.g., moved, new vehicle), health & well-being (e.g., overcame an illness, quit a habit), and travel & experiences (e.g., return from holidays, achievement, award). Users can specify different properties related to these events such as name, location, related persons, start time, ending time, description and so on. We recognize posting any of such events using life event posting functionality of Facebook as having the PostingLifeEvent interaction type.

Facebook also allows a user to post her current place/location by referring to an existing event or page object of Facebook which ultimately represents the user’s current location (at a certain scale) and possible activity. If a user posts her current place as “being in a stadium SSS”, it represents that the user is located in a particular stadium, e.g., SSS, and possibly watching a game (until being specified explicitly). Users also can post a photo, a collection of photos (called photo album), a video and music. The related properties of these posting are specified using attributes of the above classes. For example, content of the message, physical address of the place, starting and ending time of the life event, and so on.

Like posting, in Facebook users can comment on, like and share various digital contents which we capture using Commenting, Like and Sharing classes, respectively, with their further classification. SendingMessage and ReplyingMessage refer to a user chatting or sending offline messages (e.g., emails) to other users.

**Topic**  We capture the main theme associated with such different posting using the Topic class which can be extended based on application requirements. For instance, an application that is interested in games, the Topic would be different game names such as cricket, soccer, basket ball, with their further classification hierarchy (if required).
Figure 5.10: An InteractionType ontology
Social Roles and Social Relationships  Facebook allows users to perform different types of interaction activities (as explained above) which are notified to a single individual or a group of individual or none (‘only me’). User can specify the recipient(s) based on the social roles individuals are playing, or the social relationships between user and individuals, or the name/identity of each person. In Section 4.3 we have presented different types of social roles and social relationships which can be adopted and reused to capture social context information related to users’ interaction events in Facebook.

5.4.4 Examples of Interaction Event

Based on the model described above, an interaction event can be specified using a set of attribute-value pairs. For instance, an interaction event “in Facebook Bob has posted his current location, MCG Stadium, at 2:30PM and based on Bob’s posting preference that post has been notified to all of his connections who are living in the Melbourne area” can be specified as follows:

\[
\text{ie1: InteractionEvent(platform= Facebook, from= Bob, related= LivingCity, hasType= PostingCurrentLocation, atPlace= MCG Stadium, atTime= TemporalEntity(start=2:30PM, duration=0ms))}
\]

At 3:21PM bob posts a photo that has been taken in the stadium and shares it with his close friends:

\[
\text{ie2: InteractionEvent(platform= Facebook, from= Bob, related= CloseFriend, hasType= PostingPhoto, about= CricketGame, atTime= TemporalEntity(start=3:21PM, duration=0ms))}
\]

After some time, at 3:39PM Bob posts a status message in Facebook like “Wow! great 6” or “4” or “out” and shares it with his close friends:

\[
\text{ie3: InteractionEvent(platform= Facebook, from= Bob, related= CloseFriend, hasType= PostingStatus, about= CricketGame, atTime= TemporalEntity(start=3:39PM, duration=0ms))}
\]
5.4.5 Interaction Event Based Situation Specification

We specify a situation (SITU) as a set of interaction event patterns with temporal and/or logical relationships among them, while any information in the event instance(s) can be integrated to the situation description. Our proposed situation specification language comprise five basic elements: (i) Interaction Event Pattern (IEP), (ii) Complex Interaction Event Pattern (CIEP), (iii) Temporal Operator (TOP), (iv) Binary Logical Operator (BOP), and (v) Relational Operator (ROP). The syntax of our situation specification language is shown below.

\[
\begin{align*}
\text{SITU} &:= \text{CIEP} | \text{IEP} \\
\text{CIEP} &:= (\text{CIEP} \ TOP \ CIEP) | (\text{IEP} \ TOP \ IEP) | (\text{CIEP} \ TOP \ IEP) | \\
&\quad (\text{IEP} \ TOP \ CIEP) | (\text{CIEP} \ BOP \ CIEP) | (\text{IEP} \ BOP \ IEP) | \\
&\quad (\text{CIEP} \ BOP \ IEP) | (\text{IEP} \ BOP \ CIEP) | (\text{NOT} \ CIEP) \\
\text{IEP} &:= \text{InteractionEvent(CONST)} | (\text{NOT} \ IEP) \\
\text{CONST} &:= (\text{Attr ROP Value}) | (\text{Attr ONTOEQ Value}) | \\
&\quad (\text{CONST} \ BOP \ CONST) | (\text{CONST}) | () \\
\text{TOP} &:= \text{BEFORE} | \text{AFTER} | \text{COINCIDES} | \text{DURING} | \text{FINishes} | \\
&\quad \text{FINishedBY} | \text{STARTs} | \text{STARTedBY} | \text{MEETS} | \text{METBY} | \\
&\quad \text{OVERLAPS} | \text{OVERLAPPEDBY} | \text{INCLUDES} \\
\text{BOP} &:= \text{AND} | \text{OR} \\
\text{ROP} &:= \text{LT} | \text{GT} | \text{LE} | \text{GE} | \text{EQ} | \text{NE}
\end{align*}
\]

An interaction event pattern (IEP) is defined by imposing constraints in the attributes of the interaction event model. Each constraint restricts an attribute’s (Attr) value. Our interaction event ontology serves as a vocabulary for domain experts to specify the attributes’ values. If an attribute’s value refers to an ontological concept, ONTOEQ keyword is used which means that the attribute should be matched semantically. For non-ontological concept, the relational operators (ROP) such as less than (LT), greater than (GT), less than or equal (LE), greater than or equal (GE), equal to (EQ) and not equal (NE) are used to specify constraint. The binary logical operators (BOP) AND and OR can be used to compose multiple constraints in an interaction pattern.
An interaction event can have one of two temporal features: instantaneous/point-in-time (zero duration) or interval-based (duration greater than zero). For point-in-time, the correlation between events could only be one of three types: “before”, “after” or “coincides”. For interval-based, as introduced by Allen [28], the correlation types between events become thirteen: “before”, “after”, “during”, “coincides”, “meets”, “met by”, “overlaps”, “overlapped by”, “starts”, “started by”, “finishes”, “finished by” and “includes”. We refer these correlation types as temporal operators (TOP).

A complex interaction event pattern (CIEP) correlates two IEPs or complex interaction event patterns using a temporal operator or binary logical operator. CIEP also can be specified by complementing (using NOT) a complex interaction event pattern.

5.4.6 Human-readable Situation Specification Language

An important aspect of situation specification is there readability and user friendliness. Situation specification language presented in the previous section aimed mostly at developers. However, it sometimes needed that these rules are readable and understandable by the domain experts or non-technical persons. Ideally, they should be able to change the rules or even write new ones. Looking at a rule, one should immediately have an idea of what it is about. Thus, we propose a human-readable situation specification language (SituLang) which is close to the natural English language.

```
when
   [not]{An atomic interaction event $aie_id$ occurred [in "platform_name"] [which is a message communication, {type "value"} and/or {from actor "value"} and/or {from role "value"} and/or {in place "value"} and/or {to actor "value"} and/or {to role "value"} and/or {related with "value" relationship} and/or {discuss about "value"} and/or {at time "value"} and/or {duration is less than or equal to | is less than | is greater than or equal to | is greater than | is equal to | equals "value")}]
```
[and|or]

[not]{A complex interaction event $cie_id$ observed between $ie_id$ and $aie_id$, where interaction event $aie_id$ is a message communication, {type "value"} and|or {from actor "value"} and|or {from role "value"} and|or {in place "value"} and|or {to actor "value"} and|or {to role "value"} and|or {related with "value" relationship} and|or {discuss about "value"} and|or {at time "value"} and|or {duration is less than or equal to | is less than | is greater than or equal to | is greater than | is equal to | equals "value"}, has occurred "before"|"after"|"coincides"|"during"|"finishes"|"finishedby"|"includes"|"meets"|"metby"|"starts"|"startedby"|"overlaps"|"overlappedby" $ie_id$ [by "value"|by "value1" and "value2"|between "value1" and "value2"]}

then

Notify "situation" //or any additional actions

The syntax of our human-readable situation specification is shown above, where the word in-between '$' represents variable name, '"' represents fix attribute value, '{' represents optional terms, '{' represents zero or more terms. The symbol '|' represents choice. In this language, a situation is composed of a set of sentences. Each sentence describes either (1) a type of simple interaction event with constraints in its properties, or (2) a complex interaction event which describes a correlation of an interaction event with another already stated simple or complex interaction event.
5.4.7 Situation Inference Example

A person’s situation, for example, “watching a game in a stadium” can be inferred from her interaction activities in Facebook. We can say a person is most likely watching a game in a stadium - if the person posts her current location “at stadium” and after a while posts something about sports or stadium. We can specify this situation using our human-readable situation specification language as follows:

when
An atomic interaction event IE1 occurred in "Facebook" which is a message communication, type "PostingCurrentLocation" and in place "Stadium"
A complex interaction event CIE1 observed between IE1 and IE2, where interaction event IE2 is a message communication, type "PostingSomeThing" and discuss about "Sports", has occurred "after" IE1
then
Notify "Person is watching a game in a stadium"

Following the situation specification syntax defined in Section 5.4.5, the above situation can be rewrite as follows:

IE1 := InteractionEvent(hasType EQ "PostingCurrentLocation" AND atPlace ONTOEQ "Stadium")
IE2 := InteractionEvent(hasType ONTOEQ "PostingSomeThing" AND about ONTOEQ "Sports")
CIE1 := IE2 AFTER IE1
SITU(Person is watching a game in a stadium) := CIE1

Note that in this situation specification rule, we have used the EQ relational operator with the hasType property in specifying the IE1 interaction event pattern (IEP), because PostingCurrentLocation is one of the possible primitive values for the hasType property (i.e., no further classification of this concept, see Fig. 5.10) and an interaction event of
the `PostingCurrentLocation` type will be reported with the `hasType` attribute value equal to `PostingCurrentLocation` (see example `ie1` in Section 5.4.4). Thus, in this case, we do not need to use ONTOEQ for semantic matching. On the other hand, in the `IE2` pattern we should use ONTOEQ with the `hasType` property as `PostingSomeThing` is not a primitive value and thus semantic matching is required. For instance, `IE2` pattern has restricted the `hasType` attribute value `PostingSomeThing`, where the `ie2` and `ie3` interaction events are reported with the `hasType` attribute value `PostingPhoto` and `PostingStatus`, respectively. Therefore, without semantic matching the inference system cannot identify that the `ie2` and `ie3` interaction events match the `IE2` pattern.

The key benefit of incorporating the `IntEO` in the situation specification language is that it simplifies the situation rules by reducing complexity. For instance, the `IntEO` simplifies the constraint in the `IE2` by allowing us to state `hasType ONTOEQ "Posting-SomeThing"` in place of

\[
\text{hasType EQ } "\text{PostingPhotoAlbum}" \text{ OR hasType EQ } "\text{PostingLifeEvent}" \text{ OR hasType EQ } "\text{PostingPhoto}" \text{ OR hasType EQ } "\text{PostingMusic}" \text{ OR hasType EQ } "\text{PostingStatus}" \text{ OR hasType EQ } "\text{PostingVideo}" \text{ OR hasType EQ } "\text{PostingSCurrentLocation}\"
\]

This detected situation, i.e., “Person is watching a game in a stadium”, can be used by different applications and services. For example, an online ticket selling service can identify a person of interest to notify the schedule of the next tournament; automatically increase phone ring tone volume because of a noisy environment.

### 5.5 Summary

In this chapter, we have presented our approach to reasoning about different types of social context information. We have considered four aspects of reasoning social context information such as property, time-stamp, abstraction and temporal correlation.

We have adopted an ontology-based approach and used OWL2-DL for deriving, deducing and abstracting social context information. We have presented reasoning rules to deriving different types of family relationships from social role and gender information. We
have shown how to derive new types of friend relationship from already defined friend relationships and their properties. We have also presented reasoning rules to deducing situations considering their time-stamp aspect, and further abstracting the deduced situations to define more useful situations.

In related to the temporal correlation aspect, we have further presented a novel approach to inferring users’ situations from their interaction events in different social interaction platforms. The novelties of our approach are that we have considered users’ interaction events in inferring situations, and we have combined the semantic and temporal aspects of these events in defining and specifying situations. In doing so, we have developed an interaction event ontology (IntEO) to capture users’ interaction activities in different platforms. To capture both the general and domain-specific concepts, the IntEO includes an upper interaction event ontology which can be customized and specialised to form domain-specific ontologies. As examples, we have presented domain-specific ontologies for Facebook and cooperative convoy application. To allow developers to specify situations based on interaction event patterns and their correlations, we have proposed a situation specification language that integrates the IntEO with temporal relations, where the IntEO serves as a vocabulary for developers to specify the attribute values in an interaction event pattern and consequently simplifies the situation rules. We further proposed a human-readable situation specification language for domain experts, which is close to the natural English language and thus assists non-technical person to understand, define and modify situation specifications.
Chapter 6

Social Context Management

In Chapter 4 and Chapter 5, we have presented ontologies, description logic based inference rules and a situation specification language for modelling and reasoning about different types of social context information. In this chapter, we introduce our approach to managing access to such social context information by considering users’ privacy preferences (section 6.1). We also present our approach to modelling runtime social interactions and managing their runtime adaptation (section 6.2).

6.1 Managing Access to Social Context Information

Users’ social context information is inherently sensitive. In pervasive social computing environments, users might need to allow others to access their social context information for greater benefits but this may also compromise their privacy. Thus, users should be able to retain control over who has access to their personal information under which conditions. We adopt policy-based access control to allow users to control their social context information. In this regard, we propose an ontology-based socially-aware access control policy model (SACOnto) (section 6.1.1) and a description logic based language (section 6.1.2) to represent and specify users’ privacy preferences. We also present a mechanism to check the inconsistency in policy specification (section 6.1.2) and to enforce users’ policies while accessing their social context information (section 6.1.3).
The main advantages of our approach are that a DL reasoner can be used for (1) detecting automatically inconsistencies in the specification of policies, and (2) executing policies.

6.1.1 A Socially-Aware Access Control Policy Model

We propose an access control policy model to protect the users’ social context information based on the context information itself. Thus, our policy model, called Socially-aware Access Control Ontology (SACOnto) (see Fig. 6.1), reuses the concepts from the core SCOnto model to define under which conditions (i.e., the social context) a given resource is accessible. The accessible resource could be any social context information (e.g., relationship, situation), including the meta-information related to any relationship of a user such as the numbers of friends and colleagues.

Studies have revealed that users want to define policies that apply to all people with certain relationships or roles rather than explicitly name each person [181] [123]. Thus in access conditions, we consider both the requester’s role and her relationships with the information owner to reflect the way users tend to group similar sets of people when deciding to share resources with other users. Therefore, our model is able to represent existing policy models, such as the popular role-based model [80] and the recently emerged relationship-based model [81], with enhanced capabilities.

In addition, we consider the owner’s current situation information which is another important aspect of defining policies [181]. Users typically specify their policies using combinations of the main dimensions driving access control decisions, including

- Who is requesting access and what is her relationship with the owner?
- What role is the requester playing?
- The (current) situation of the owner when the request is made?
- What type of resource is being requested and what are its characteristics?

Our ontological model enables a user (owner) to specify logical relations between the above fundamental elements. In the owner’s conceptual model, such aspects as
Figure 6.1: Access control policy model
who/what/when dimensions are typically interrelated. Our model (Fig. 6.1) captures these dimensions which can be read as follows: A Policy, is defined using a policy id (an instance of the Policy class) and a policyStatement, isDefinedBy an Owner which specifies Access decision (Denied or Granted or GrantedInGL1 (granted in granularity level 1), and so on) for Requester(s) who plays a SocialRole, relatedTo Owner and hasRelationship with owner named SocialRelationship, asksFor a Resource, when the Owner hasSituation Situation.

There are a number of constraints related to the SCACOnto. For instance, a policy should have exactly one policy statement; an access decision can not be a member of multiple access types, e.g., both granted and denied. The OWL2-DL representation of these constraints are as follows:

\[
\text{Policy} \sqsubseteq = 1 \text{policyStatement} \\
\text{Granted} \sqcap \text{Denied} \sqsubseteq \bot
\]

A complete list can be found in Appendix A.2.1.

Let us consider a policy, “Any of my colleagues can access my current situation when I am in a meeting” (Policy#1). In such a case, the owner’s access decision is based on the relationship with the requester (who), the resource being accessed (what), and the situation of the owner at request time (when). While traditional approaches generally consider these dimensions to be orthogonal, our model reflects the owner’s way of thinking by supporting the cross-dimensional definition of policies.

Another key feature of our model is the ability to specify access permissions at different levels of granularity, i.e., disclosing information at a certain level of abstraction by hiding the specific fact. For instance, “Upon being asked the current status (situation) by supervisor, a student may want to state her situation as being ‘AtDesk’ (granularity 1) while she is actually ‘Chatting’ with friends (granularity level 3)” (Policy#2).

### 6.1.2 Policy Specification and Inconsistency Detection

Fig. 6.2 shows the basic steps in policy specification. Using a graphical user interface,
an owner specifies her privacy policies. These policies can then be easily transformed to the Description Logic (DL) format. The template of our policy rule in the DL is defined as follows:

\[
\text{Denied} / \text{Granted}/\text{GrantedInGL}\text{n} \equiv \text{Access} \sqcap \exists \text{decision}(\text{Requester} \sqcap \\
\exists \text{hasRelationship}.?\text{SocialRelationship} \sqcap \exists \text{asksFor}.?\text{Resource} \sqcap \\
\exists \text{plays}.?\text{SocialRole} \sqcap \\
\exists \text{relatedTo}.(\text{Owner} \sqcap \exists \text{hasSituation}.?\text{Situation}))
\]

where bold words with a preceding ‘?’ mark are variables that will be filled based on the owner’s privacy preference statements. For example, we can represent the above Policy#1 and Policy#2 as follows:
Chapter 6. Social Context Management

Policy#1:

\[ \text{Granted} \equiv \text{Access} \sqcap \\
\exists \text{decision}(\text{Requester} \sqcap \exists \text{asksFor.Situation}) \\
\exists \text{hasRelationship.Coleague} \\
\exists \text{relatedTo.}(\text{Owner} \sqcap \exists \text{hasSituation.Meeting}) \]

Policy#2:

\[ \text{GrantedInGL1} \equiv \text{Access} \sqcap \\
\exists \text{decision}(\text{Requester} \sqcap \exists \text{asksFor.Situation}) \\
\exists \text{hasRelationship.Student-Supervisor} \\
\exists \text{relatedTo.}(\text{Owner} \sqcap \exists \text{hasSituation.Chatting}) \]

Once the policy is transformed to the DL format, it is inserted into the owner’s access control ontology. For instance, Policy#1 and Policy#2 will be inserted into the SACOnto as an equivalent class of Granted class and GrantedInGL1 class, respectively.

Let us assume that the owner specified another policy, “An admin-staff can NOT access my current situation when I am in individual meeting” (Policy#3), which can be represented in DL format as follows:

Policy#3:

\[ \text{Denied} \equiv \text{Access} \sqcap \\
\exists \text{decision}(\text{Requester} \sqcap \exists \text{asksFor.Situation}) \\
\exists \text{hasRelationship.AcademicStaff-AdminStaff} \\
\exists \text{relatedTo.}(\text{Owner} \sqcap \exists \text{hasSituation.IndividualMeeting}) \]

The AcademicStaff-AdminStaff relationship (used in Policy#3) is a subclass of Colleague (used in Policy#1) and the IndividualMeeting (used in Policy#3) is a subclass of Meeting (used in Policy#1). However, Policy#1 grants the access and Policy#3 denies the
access, where the Granted and Denied classes are disjoint with each other (according to the constraint [6.1]). Thus, Policy#3 makes the owner’s SAConto KB inconsistent as it violates the disjoint constraint by overlapping with Policy#1. During the policy specification process, if such inconsistency occurs the inconsistent policy will be rejected and deleted, and the owner will be notified.

6.1.3 Policy Enforcement in Accessing Social Context Information

When an access request or query comes from an user’s application which basically corresponds to invoking a function from the query APIs (see section [7.1.5]), some facts about the context of the query are temporarily inserted into the SAConto knowledge base to get the access decision based on the information owner’s privacy preferences. The following steps describe this process.

1. Identify the individual (who requests information) of the SConto:Person class as an individual of the Requester class, say John

2. Insert an individual, say resourceIns, into the requested resource class such as MetaInfo or Situation or SocialRelationship or any of their subclasses

3. Insert an individual, say accessRequestResult, into the Access class

4. Link the inserted Requester individual, i.e., John, to the Resource individual, i.e., resourceIns, using asksFor object property

5. Link the inserted Access individual, i.e., accessRequestResult, to the Requester individual, i.e., John, using decision object property

After that a DL reasoner is fired which classifies the individual of Access class to one of its subclasses, i.e., Denied, GrantedInGL1, and so on, based on the defined policies. The query processing manager considers this result to answer the query.

Let us consider a scenario, “Alice, who is a graduate student, specifies her privacy policy as Policy#2. John who is the supervisor of Alice requests the current situation of Alice while Alice is chatting with her friend via instant messaging”. The process of checking Alice’s privacy preferences regarding that request by following the above mentioned steps are:
(1) Specify John as an individual of Requester class, (2) insert an individual, namely resourceIns, to the Situation class, (3) insert an individual, namely accessRequestResult, to the Access class, (4) link John to the resourceIns using asksFor object property, and (5) link accessRequestResult to John using decision object property. Finally, the DL reasoner is fired which classifies the accessRequestResult as an individual of GrantedInGL1 (one of the subclasses of Access class), which means that in the above scenario Alice allows John to know her current situation at granularity level 1. Thus, in this case, John will get a reply from the query processing module about Alice’s current situation as ‘AtDesk’ (instead of ‘Chatting’).

6.2 Modelling and Managing Social Interactions

There are three major requirements of developing an interaction-centric socially-aware application for facilitating social interactions (as identified in Section 2.2.2):

- **First**, the application should support social interactions complying with the agreed interaction relationships, *i.e.*, constraints and obligations.
- **Second**, to facilitate collaboration, the application needs to provide a coordinated view of the interactions.
- **Third**, the application needs to support runtime adaptation as the interaction relationships evolve over time, and need to adapt with the changes in requirements and environments.

In section 4.3.3, we have presented an *interaction-oriented relationship ontology* and a *social interaction ontology*, which allow us to capture different types of *interaction-oriented social relationships* between collaborative actors and their associated *agreements* and *constraints*. In this section, we present an approach to modelling *runtime social interactions* where the runtime models are built based on the information captured in the ontologies. The runtime models *enforce* agreements and constraints while *mediating* social interactions between actors. Such *agreements* and *constraints*, and thus models, are also subject to change. To address this issue, we present an approach to *managing adaptation* in such runtime models.
A promising approach to support runtime changes is to develop adaptation mechanisms that leverage software models, referred to as models@run.time \[43\]. Runtime models provide “abstractions of runtime phenomena” \[83\]. That is, the model\[1\] should represent the system and should be linked in such a way that they constantly mirror the system and its current state and behaviour; if the system changes, the representations of the system—the models—should also change, and vice versa. A runtime model can be seen as a live development model that enables dynamic evolution and the realisation of software design \[43\].

To fulfill the aforementioned requirements of developing interaction-centric socially-aware applications, we propose an approach to modelling social interactions @run.time (Section 6.2.1), and a technique to perform runtime adaptation in a consistent manner (Section 6.2.2).

### 6.2.1 Modelling Social Interactions

Role based approaches have been increasingly used in software system design and implementation. Roles, in different variations, have been applied in many fields such as computer supported collaborative works, role based access control, multi-agent systems, and object modelling \[197\]. In the access control domain, the notion of roles has been used to represent a set of responsibilities and privileges associated with a particular position within an organisation \[162\]. In CSCW and group-aware applications, the notion of role has been used to provide a natural abstraction for representing users without requiring prior knowledge of their individual identities \[61,126\]. In multi-agent systems, the concept of role has been used as an abstraction for representing an entity’s behaviour \[48\], where a role specifies a functional behaviour of an agent.

We have taken a role based approach to model social interactions as it has been evaluated as being very useful in modelling entities, their relationships, functions and interactions \[49\]. Our approach utilises the notion of roles for functional abstractions. Among the many different approaches to role modelling \[175\], one common way to develop a role based model is to use organisational concepts \[60\]. According to Daft \[62\]

---

\[1\]through out this section model(s) refers to runtime model(s)
“organisations are social entities that are goal directed, deliberately structured, and coordinated activity systems, linked to the environment”. A similar view has taken by Rollison [156], who says “organisations are artifacts, that are goal directed social entities with structured activities”. Organisations are typically viewed as a managed network of roles that decompose the abstract functions of the organisation into descriptions that can be performed by role players (e.g., people, subsystems, other organisations). Organisation theory addresses the structuring of organisations in order to more effectively achieve those goals (e.g., [102]). The structure defines how the organisation is composed, i.e., what units it consists of and what are the relationships or the connections between these units.

We adopt the view of organisation as a composition structure of dynamic relationships between social roles in order to model runtime social interactions. Role Oriented Adaptive Design (ROAD) [59] introduces such an organisational structure to define self-managed software composites. In this thesis, we use ROAD and further extend it to define adaptable runtime social interaction models. ROAD defines roles or positions of a composite. Players can come and play these roles. We enrich the interactions in ROAD with the new constructs of obligation and conversation. We specialise the ROAD composite structure (meta model) from the domain and player perspectives. To provide a collaborative view of runtime social interactions among actors, we propose a domain-centric social interaction model (DSIM) (Section 6.2.1.1). To facilitate collaboration, the applications need to provide a coordinated view of the interactions, allow a user to specify his/her coordination preferences and perform the coordination in an automated manner. To address this requirement, we further propose a player-centric social interaction model (PSIM) (Section 6.2.1.2).

6.2.1.1 Domain-centric Social Interaction Model

The Domain-centric Social Interaction Model (DSIM) provides a structure of the relationships among social roles associated with a particular domain or environment such as a company, a cooperative convoy, and so on. The DSIM grouped the social interactions (stored in the social interaction ontology) from a domain perspective. Fig. 6.3 shows the meta-model of the DSIM, which adopts and extends the ROAD meta-model for creating adaptive software. The DSIM meta-model comprises four key elements: social role, social...
relationship, player, and organiser role. To visualise the structure of the social interaction model, we define a set of graphical notations (see Fig. 6.4).

![Figure 6.3: Meta-model of DSIM](image)

**Figure 6.3: Meta-model of DSIM**

![Figure 6.4: Graphical notations for representing social interaction model](image)

**Figure 6.4: Graphical notations for representing social interaction model**

A social role (called functional role in ROAD) defines and represents expected functional interactions of a participating actor. Social roles are loosely-coupled elements and are modelled as first class entities, and as such they are separated from their players who play those roles. An interaction-oriented social relationship (called contract in ROAD)

\[2\text{ we use the term “actor” and “player” interchangeable throughout this thesis} \]
is an association between two social roles, which defines and represents the social interactions (called interactions in ROAD) between them.

We incorporate conversation and obligation into a social relationship by extending the ROAD contract, as these constructs are important to represent interaction-oriented social relationships. As such, the social relationship can suitably captures the interdependencies between the social roles (or their corresponding players). A social relationship specifies the interactions between these roles by defining what social interactions can occur between the roles and the sequences of these interactions (namely conversations). In addition, a relationship also defines the non-functional requirements of the interactions in terms of operational parameters and obligations (e.g., time constraints on interactions) imposed on the players associated with that relationship.

The organiser role and its player provide the capability for managing and adapting a social interaction model to cope with changes in the requirements and dynamic environments. As such, the social roles, relationships, interactions, conversations, obligations and operational parameters can be added, modified and removed depending on the changing requirements and environments. While social roles, players and relationships are entities at the DSIM’s functional layer, the organiser role and its player are entities at the DSIM’s management layer, as depicted in Fig. 6.5.

![Figure 6.5: Separation of functional and management aspects](Image)

We have developed an XML schema based on the DSIM meta-model (see Appendix E.1). For the sake of readability, we define a template for the DSIM (see Fig. 6.6).
by using a pseudo notation based on the DSIM XML schema. In this notation, the terms in boldface represent the XML tags in our schema. Social interactions among users (in an interaction-centric socially-aware application) are designed using the DSIM template.

To illustrate the domain-centric social interaction model, we use the cooperative convoy application scenario, introduced in Section 2.2.1, as an example. In that scenario, there are three domains that need to be modelled, namely, the two car rental companies (Budget and AVIS) and the cooperative convoy.

The Budget rental company allows travel guide and roadside assistance services, and the company has its own policies and constraints to use these services through the telematics system in the cars rented from them. These constraints and policies are captured in the domain-centric social interaction model of the Budget, called BudgetDSIM, as shown in Fig 6.7. This model comprises four social roles (Rented Car, Rental Company, Travel Guide and Roadside Assistance), four interaction-centric social relationships (R2, R3, R4 and R5), and one organiser role. These relationships represent the interaction constraints and agreements (as specified in the scenario) among the social roles. For instance, R2 (RentedCar-TravelGuide) relationship represents that “upon receiving a route request from a rented car, travel guide should send a route notification by 15 seconds”. Fig. 6.8 shows the representation of this agreement which consists of two social interactions (i7 and i8), one conversation (c1) and one related obligation (o3). The i7 social interaction represents the route request from rented car to travel guide, while the i8 represents the route notification from travel guide to rented car. The c1 conversation imposes the sequence related constraints between these two social interactions, while o1 imposes time related obligation.

The Budget company is the owner of this runtime model and controls it through the organiser role. The customer, e.g., the tourist group, who rents Car#1 plays the rented car role through the telematics system of that car. The roadside assistance service (e.g., RACV.com.au) and the travel guide service (e.g., travelguide.org) play the roadside assistance and travel guide roles respectively. A complete description of the BudgetDSIM is presented in Appendix C.1.2.
DSIM $\text{modelName}$

OrganiserRole $\text{orgRoleName}$;
SocialRole $\text{roleName}_1, \text{roleName}_2, ..., \text{roleName}_n$;
RolePlayerBinding $b_1$

RoleName $\text{roleName}_n$;
PlayerIdentity;
;
RolePlayerBinding $b_2[...], ..., b_n[...]$;

SocialRelationship $\text{relID}_1$

RelationshipName $\text{rel}\_name$;
RoleA $\text{roleName}_m$;
RoleB $\text{roleName}_n$;

SocialInteraction $\text{interactionID}_1$

// message format $\text{msg\_name(type:paramName,...)}$
MsgSignature $\text{message}$;
Direction $\text{direction}$; // RoleAToRoleB or RoleBToRoleA
InteractionType $\text{type\_name}$; // any string (optional)
RelatedObligation $\text{obgID}_n$;
;
SocialInteraction $\text{interactionID}_2[...], ..., \text{interactionID}_n[...]$;

OperationalParam $\text{opParamID}_1$

OPParamName $\text{param\_name}$; // any string
OPParamValue $\text{value}$; // any non-negative number
TimeUnit $\text{tUnit}$; // MS, SEC, MIN, HOUR or DAY
DistanceUnit $\text{dUnit}$; // MM, CM, M or KM
;
OperationalParam $\text{opParamID}_2[...], ..., \text{opParamID}_n[...]$;

Obligation $\text{oblID}_1$

Timer $\text{timer}$; // either DURATION or PERIOD
RelatedOperationalParam $\text{opParamID}_n$;
;
Obligation $\text{oblID}_2[...], ..., \text{oblID}_n[...]$;

Conversation $\text{conID}_1$

SocialInteractionX $\text{interactionID}_m$;
SocialInteractionY $\text{interactionID}_n$;
Sequence $\text{sequence\_type}$; // leadsTo or precedes
RelatedObligation $\text{obgID}_n$;
;
Conversation $\text{conID}_2[...], ..., \text{conID}_n[...]$;
;
SocialRelationship $\text{relID}_2[...], ..., \text{relID}_n[...]$;

Figure 6.6: A template for the domain-centric social interaction model (DSIM)
On the other hand, the AVIS company allows access to traffic management and roadside assistance services. Like Budget, we also model the domain-centric runtime social interactions for AVIS, called AVISDSIM, as shown in Fig. 6.9. It also consists of four social roles. Unlike Budget, it does not have a travel guide role. However, it has a traffic management role. There are four interaction-oriented social relationships, i.e., R6, R7, R8 and R9, among these roles, which represent agreements and constraints among social roles, according to the specification in the scenario. The AVIS company plays the organiser role of this runtime model as the company is the owner of this model. The customer, e.g., the tourist group, who rents Car#2 plays the rented car role through the telematics system of that car. The roadside assistance service (e.g., RACV.com.au) and the traffic management service (e.g., VicRoad.org) play the roadside assistance and traffic management roles respectively. A complete description of the AVISDSIM is presented in Appendix C.1.3.

Like Budget and AVIS, the interaction-relationship between the drivers of Car#1 and Car#2 is modelled from a domain perspective, called ConvoyDSIM, as shown in Fig. 6.10. According to their agreements, it consists of two social roles: LeadingCar (LC) and FollowingCar (FC), and their interactions are captured in the R1 (LeadingCar-
SocialRelationship R2{
    RelationshipName RentedCar-TravelGuide;
    RoleA RentedCar;
    RoleB TravelGuide;
    SocialInteraction i7{
        MsgSignature routeRequest(String sourceAddr, String destAddr,
                                   String preference);
        Direction RoleAToRoleB;
    };  
    SocialInteraction i8{
        MsgSignature routeNotification(String route);
        Direction RoleBToRoleA;
    };
    Conversation c1{
        SocialInteractionX i7;
        SocialInteractionX i8;
        Sequence leadsTo; // i7 leadsTo i8
        RelatedObligation o3;
    };
    Obligation o3{
        Timer DURATION;
        RelatedOperationalParam p3;
    };  
    OperationalParam p3{
        OPParamName maxTimeDelay;
        OPParamValue 15;
        TimeUnit SEC;
    };
    ......  
};

Figure 6.8: A partial description of R2 relationship in BudgetDSIM
FollowingCar) relationship. The Car#1 plays the LeadingCar role, while the Car#2 plays the FollowingCar role. Car#1 is also designated to play the organiser role of the ConvoyDSIM.

Fig. 6.11 shows a partial notational description of ConvoyDSIM. It includes description of the $i_1$ social interaction, and its associated obligation ($o_1$) and operational parameter ($p_1$), which represents the agreement “following car should send a position update to the leading car every 10 seconds”. The complete description of ConvoyDSIM is presented in Appendix C.1.1.
**Figure 6.11:** A partial description of the convoy domain-centric social interaction model (ConvoyDSIM)
6.2.1.2 Player-centric Social Interaction Model

In addition to the common view of a domain-centric social interaction model, a player may have its own perception or view of the domain with respect to the role(s) it plays and the interactions it participates in that domain. Moreover, a player may operate in different domains. Thus, we also model the social interaction from a player’s perspective, namely Player-centric Social Interaction Model (PSIM). The PSIM provides an overall view of all the interactions of an individual across different domains and allows coordination of its interactions across these domains.

As presented in the previous section, the R1 interaction-relationship of ConvoyD-SIM captures the agreements between LC and FC in terms of their social interactions. But some of these interactions are driven by or depend on their interactions with other associated domains (i.e., Budget and AVIS). For instance, when the road block information is received from the traffic management service (TMS), FC should immediately forward it to LC. Likewise, when the route information is received from the travel guide service, LC should forward it to the LC. However, in some cases the driver may want to carry out these activities automatically with less or no explicit human intervention. For example, the driver of FC may want to forward the road block information to LC as soon as it is received from the TMS. The player-centric model addresses these issues by introducing the coordinator role which allows player to coordinate (automatically or manually) his/her social interactions.

Like DSIM, the PSIM is also a structured composition of social roles. Like a DSIM, a PSIM contains social roles, players and an organiser role. In PSIM, however, all social roles are played by the actor in question and are connected with the coordinator role through the role-centric relationships. Thus, in addition, it contains a coordinator role and role-centric relationships. Therefore, the PSIM meta-model comprises of five key elements: social role, coordinator role, role-centric social relationship, player, and organiser role. Fig. 6.12 shows the meta-model of PSIM. Like DSIM, we also define a template for the PSIM (see Fig. 6.13) based on the PSIM meta-model.

The coordinator role is a means of achieving inter-domain coordination, i.e., interactions

3 person with an application
Figure 6.12: Meta-model of PSIM
in one domain can be used for interactions in another domain. In the cooperative convoy scenario, drivers relate their interactions in their own rental company with each other in their cooperative convoy. For instance, Car#2 should send road blocks information to Car#1 when Car#2 receives that information from the traffic management service. The coordinator role provides a position to the player/user in question for coordinating his/her social interactions across multiple domains. A user (driver) can coordinate interactions explicitly through the application’s user interface or he/she may define some rules or use an intelligent application to coordinate his/her interactions on behalf.

A role-centric relationship is the aggregation of all the relationships associated with a particular social role in a DSIM model, but localised in the player-centric model. Algorithm 6.1 presents the aggregation procedure of computing a role-centric social relationship for a particular social roles in a domain-centric social interaction model. The algorithm takes an instance of a runtime DSIM model and a social role name, say $r$, of that model from which perspective the role-centric relationship needs to be computed. First, it creates an empty instance of social relationship. After that, it invokes the management operation `getAllRel4Role` to get all the relationships connected with the target social role $r$. Finally, the algorithm adds these relationships to the empty social relationship instance which represents the role-centric social relationship for the social role $r$. 

```
PSIM modelName{
    OrganiserRole orgRoleName;
    SocialRole roleName_1, roleName_2,...,roleName_n;
    CoordinatorRole coordRole; // single instance
    RolePlayerBinding b_1[...],...,b_n[...];
    RoleCentricSocialRelationship relID_1{
        RelationshipName rel_name;
        RoleA coordRole;
        RoleB roleName_i;
        SocialInteraction interactionID_1[...],...,interactionID_n[...];
        OperationalParam opParamID_1[...],...,opParamID_n[...];
        Obligation oblID_1[...],...,oblID_n[...];
        Conversation conID_1[...],...,conID_n[...];
    },relID_2[...],...,relID_n[...];
};
```

Figure 6.13: Template for the player-centric social interaction model (PSIM)
CHAPTER 6. SOCIAL CONTEXT MANAGEMENT

Algorithm 1: ComputeRoleCentricRel (dsim, r)

Data: r is a social role and dsim is a domain-centric social interaction model
Result: a role centric relationship, roleCentricRel, for the social role r

begin
1  roleCentricRel ← empty /* create an empty relationship */
2  relList ← getAllRel4Role(dsim, r) /* all social relationships connected to the social role r */
3  foreach rel ∈ relList do /* rel is a relationship in the relList */
4      foreach i ∈ rel do /* i is an interaction in rel */
5          roleCentricRel.addInteraction(i) /* adding interaction i into roleCentricRel */
6      foreach c ∈ rel do /* c is a conversation in rel */
7          roleCentricRel.addConversation(c) /* adding conversation c into roleCentricRel */
8      foreach o ∈ rel do /* o is an obligation in rel */
9          roleCentricRel.addObligation(o) /* adding obligation o into roleCentricRel */
10     foreach p ∈ rel do /* p is an operational parameter in rel */
11        roleCentricRel.addOperationalParam(p) /* adding operational parameter p into roleCentricRel */
12  return roleCentricRel
Fig. 6.14 shows the player-centric social interaction models of Car#1 and Car#2, called Car#1PSIM and Car#2PSIM respectively, and how they relate to the domain-centric models (i.e., BudgetDSIM, ConvoyDSIM and AVISDSIM). According to the cooperative convoy scenario, Car#2 plays the FollowingCar role in ConvoyDSIM and RentedCar role in AVISDSIM. Thus, the Car#2PSIM contains FollowingCar and RentedCar social roles and their associated role-centric social relationships, i.e., Rr3 and Rr4. The Rr4 is the role-centric social relationship of RentedCar social role in AVISDSIM. Therefore, Rr4 is the aggregation of R6, R7 and R8 social relationships in the AVISDSIM. The Rr3 and Rr4 connect the Coordinator Role to the FollowingCar and RentedCar roles respectively. Similarly, Car#1PSIM consists of LeadingCar and RentedCar social roles and their associated role-centric social relationships, i.e., Rr1 and Rr2, which connect these social roles to the Coordinator Role.

A player/actor plays social role(s) in different DSIM through its PSIM. For instance, the Car#1 plays RentedCar role in BudgetDSIM and LeadingCar role in ConvoyDSIM through the Car#1PSIM. Similarly, the Car#2 plays RentedCar role in AVISDSIM and FollowingCar role in ConvoyDSIM through the Car#2PSIM.

In the PSIM, the coordinator (role and player) does the most important work as it...
performs actions on behalf of the player based on his/her preferences. These preferences or coordination rules are player specific and can be defined using Event-Condition-Action (ECA) rules. For instance, the Car#2 driver’s preference of automatically forwarding road block information, received from traffic management service, can be represented using the following ECA rule:

```
Coordination-Rule "Preference of the Car#2 driver"
when OnNotificationReceived //Event
then
  if (?receivedNotification == notifyRoadBlock) //Condition
  do FC.notifyRoadBlock //Action
```

### 6.2.1.3 Mediating Runtime Social Interactions

Social interaction models (SIMs) are not just a modeling or design-time construct, and they are also runtime entities that mediate runtime interactions between actors. Interactions among an actor, its player-centric social interaction model (PSIM), and the relevant domain-centric social interaction models (DSIMs) are loosely coupled and use a messaging style. A runtime social interaction model/composite acts as a message router (see Fig. 6.15) that (1) receives messages from an actor, (2) evaluates conditions specified in associated relationships, and (3) passes the messages to another actor or a social interaction model (as a player) or notifies the actor(s) in case of any condition violation. For the PSIM, all the incoming and outgoing messages are intercepted by the coordinator role which is played by the person’s coordination application. The coordination application coordinates messages on behalf of the person based on his/her preferences.

### 6.2.2 Managing Runtime Adaptation

Social interaction models (i.e., PSIMs and DSIMs) externalise users’ interaction relationships from the applications, as depicted in Fig. 6.16. However, the requirements of such applications are subject to continuous change. Thus, both the PSIMs and DSIMs need to be managed and adapted to ensure the proper functioning and evolution of the applications in which the social interaction models play a part. Moreover, the PSIMs are

---

4person with an application
dependent on the DSIMs. Thus, the adaptation/modification in a DSIM should be propagated to the related PSIMs. In this section, we discuss the management and runtime adaptation of social interaction models to cope with the different types of changes.

![Figure 6.15: Message based interaction via a runtime social interaction model](image)

**Figure 6.15:** Message based interaction via a runtime social interaction model

6.2.2.1 Types of Changes

In general, there are two types of changes that require adaptation in a social interaction model as well as across social interaction models.

**Changes in Environments.** During convoy, it may start to rain heavily or the cars may move from one jurisdiction to another which operates different traffic management systems. Such changes cause social interaction model adaptation and are referred to as changes in environments.

**Changes in Requirements.** A third vehicle could join when the convoy is already on the way; a broken-down following car might leave the convoy before reaching its destination; or the leading car might have a mechanical problem which requires it to handover the leading car role to one of the following cars (assuming multiple following

![Figure 6.16: Externalisation of users’ interaction-relationships from applications](image)

**Figure 6.16:** Externalisation of users’ interaction-relationships from applications
cars). Such situations also cause adaptation, and are referred to as changes in requirements.

6.2.2.2 Adaptation in a Social Interaction Model

Runtime adaptation, often called dynamic adaptation, is a widely used term and is extensively studied in multiple disciplines. In pervasive computing, this term is used to denote any kind of modification at the running phase of the system \[119\]. In general, such runtime adaptation can be classified into two categories: structure (compositional) adaptation and parameter adaptation \[138\]. The adaptation in a social interaction model to cope with the changes in user requirements and environments also can be of these two types: structural (compositional) and parametric.

We achieve structural adaptation in two ways:

Modifying topology — Adding and removing social roles, players and relationships are carried out. For instance, a third vehicle could join the convoy when it is already on the way, or a broken-down leading car leaves the convoy before reaching the destination. These situations lead to the addition or removal of roles, players and relationships in the ConvoyDSIM.

Modifying the binding between a social role and its player — The same social role can be played by different players at different times. The binding between the role and the players is dynamic. For instance, in the AVISDSIM, the traffic management role can be played by different traffic management systems in the convoy at different times as the vehicle moves from one jurisdiction to another. Also due to a mechanical problem of the leading car (Car#1), the Car#1PSIM needs to unbind from the leading car role in the ConvoyDSIM and one of the following cars can be assigned to play the leading car role by binding that car’s PSIM to the leading car role in the ConvoyDSIM.

We achieve parametric adaptation by modifying relationships where adding, removing or updating interactions, obligations, conversations and operational parameters are carried out. For instance, in the ConvoyDSIM, the \textit{maxDesiredDistance} parameter value
may be reduced, say from 1 kilometre to 600 metre because of heavy rain.

6.2.2.3 Management Operations and Adaptation Rules

The organiser role provides the management capability of a runtime social interaction model, in short runtime model. The principle of separation between a role and its player is also applied to the organiser role. The organiser role is internal to a runtime model and allows its player to manage both the structure and parameters of the runtime model. The organiser role presents management rights over its runtime model, for example, to a person who owns the composite.

The organiser role exposes a management interface that contains methods for manipulating the structure (see Table 6.1), parameter (see Table 6.2) and state (see Table 6.3) of the runtime model. Operations manipulating the structure of a runtime model can be divided into two categories: Modifying topology – management operations those modify the topology of the runtime social interaction model by adding or removing social roles or social relationships, Modifying bindings - management operations those binds or unbinds a player to or from a social role belong to this category. The organiser interface also provides methods for acquiring information related to the structure, parameter and state of the runtime social interaction model (see Table 6.4), and subsequently allows the organiser player to monitor the runtime model and perform adaptation safely.

By playing the organiser role, a human can perform adaptation manually using a graphical interface. On the other hand, automatic adaptation can be defined and performed through a computer program or an agent or a set of predefined adaptation rules, as a player of organiser (depicted in Fig. 6.17). For example, the maxDistance parameter value can be reset to 600m using the following Event-Condition-Action (ECA) rule:

```adaptation-rule "Update maximum distance"
when
  EnvironmentChangeEvent(name=="RainingStatusValueChanged") //Event
then
  if(rainingStatus == HEAVY_RAIN) //Condition
    callMethodInOrgInterface("updateOperationalParam(\"R1\", maxDesiredDistance,600m)") //Action```
CHAPTER 6. SOCIAL CONTEXT MANAGEMENT

Runtime social interaction model
Organiser
Monitoring
Adaptation

A human with an application (GUI)
A computer program or rules

Figure 6.17: Organiser player and adaptation

Table 6.1: Operations provided by the organiser role to manipulate structure of the runtime model

<table>
<thead>
<tr>
<th>Category</th>
<th>Operation Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modifying topology</td>
<td>addSocialRole</td>
<td>Adds a social role to a runtime model</td>
</tr>
<tr>
<td></td>
<td>deleteSocialRole</td>
<td>Deletes a social role from a model</td>
</tr>
<tr>
<td></td>
<td>addRelationship</td>
<td>Adds a relationship between two roles</td>
</tr>
<tr>
<td></td>
<td>deleteRelationship</td>
<td>Deletes a relationship between two social roles</td>
</tr>
<tr>
<td></td>
<td>bindRolePlayer</td>
<td>Binds a player to a role</td>
</tr>
<tr>
<td></td>
<td>unbindRolePlayer</td>
<td>Unbinds a player from a role</td>
</tr>
</tbody>
</table>
Table 6.2: Operations provided by the *organiser role* to manipulate *parameter* of the runtime model

<table>
<thead>
<tr>
<th>Category</th>
<th>Operation Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modifying relationships</td>
<td>addInteraction</td>
<td>Adds an interaction to a relationship</td>
</tr>
<tr>
<td></td>
<td>addObligation</td>
<td>Adds an obligation to a relationship</td>
</tr>
<tr>
<td></td>
<td>addConversation</td>
<td>Adds a conversation to a relationship</td>
</tr>
<tr>
<td></td>
<td>addOPParam</td>
<td>Adds an operational parameter in a relationship</td>
</tr>
<tr>
<td></td>
<td>deleteInteraction</td>
<td>Deletes an interaction from a relationship</td>
</tr>
<tr>
<td></td>
<td>deleteObligation</td>
<td>Deletes an obligation from a relationship</td>
</tr>
<tr>
<td></td>
<td>deleteConversation</td>
<td>Deletes a conversation from a relationship</td>
</tr>
<tr>
<td></td>
<td>deleteOPParam</td>
<td>Deletes an operational parameter from a relationship</td>
</tr>
<tr>
<td></td>
<td>updateInteraction</td>
<td>Updates an interaction in a relationship</td>
</tr>
<tr>
<td></td>
<td>updateObligation</td>
<td>Updates an obligation in a relationship</td>
</tr>
<tr>
<td></td>
<td>updateConversation</td>
<td>Updates a conversation in a relationship</td>
</tr>
<tr>
<td></td>
<td>updateOPParam</td>
<td>Updates an operational parameter value in a relationship</td>
</tr>
</tbody>
</table>

Table 6.3: Operations provided by the *organiser role* to manipulate *state*

<table>
<thead>
<tr>
<th>Category</th>
<th>Operation Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setting state</td>
<td>setRoleState</td>
<td>Sets a state to a social role</td>
</tr>
<tr>
<td></td>
<td>setRelationshipState</td>
<td>Sets a state to a social relationship</td>
</tr>
<tr>
<td></td>
<td>setSIMState</td>
<td>Sets a state to a runtime model</td>
</tr>
</tbody>
</table>

Table 6.4: Operations provided by the *organiser role* to *monitor* a runtime model by acquiring information

<table>
<thead>
<tr>
<th>Category</th>
<th>Operation Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structure related</td>
<td>getRoleList</td>
<td>Returns a list of role names in a SIM</td>
</tr>
<tr>
<td></td>
<td>getRelationshipList</td>
<td>Returns a list of relationship names in a SIM</td>
</tr>
<tr>
<td></td>
<td>getAllRel4Role</td>
<td>Returns a list of relationship associated with a role</td>
</tr>
<tr>
<td></td>
<td>getRelationship</td>
<td>Returns a relationship name between two roles</td>
</tr>
<tr>
<td>Parameter related</td>
<td>getInteractionList</td>
<td>Returns a list of interactions in a relationship</td>
</tr>
<tr>
<td></td>
<td>getConversationList</td>
<td>Returns a list of conversations related to an interaction</td>
</tr>
<tr>
<td></td>
<td>getObligationList</td>
<td>Returns a list of obligations related to a conversation or an interaction</td>
</tr>
<tr>
<td></td>
<td>getOPParam</td>
<td>Returns an operational parameter related to an obligation</td>
</tr>
<tr>
<td>State related</td>
<td>getRoleState</td>
<td>Returns the state of a social role</td>
</tr>
<tr>
<td></td>
<td>getRelationshipState</td>
<td>Returns the state of a social relationship</td>
</tr>
<tr>
<td></td>
<td>getSIMState</td>
<td>Returns the state of a SIM</td>
</tr>
</tbody>
</table>
6.2.2.4 Social Interaction State and Safe Change

To perform adaptation in a safe manner without affecting the message flow and loss of messages, we maintain the state of each entity, i.e., the social role, social relationship and social interaction model as a whole. Fig. 6.18 shows the states and their transitions. When a social interaction model is deployed all of its entities enter into the Idle state. When a conversation (i.e., sequence of social interactions) is started, the associated social roles and relationship move to the Active state and remain there until the conversation completes. A (runtime) social interaction model (i.e., DSIM or PSIM) enters into the Active state when any of its social role or relationship becomes Active and remains there until all of its roles and relationships become Idle. When an adaptation operation (structural or parametric) starts, the corresponding entity enters to the Reconfiguration state and remains there until the adaptation completes.

![State machine for Social Role, Social Relationship and SIM Composite](image)

For Social Role and Relationship, $x =$ started conversation, $y =$ completed conversation

For Social Interaction Model, $x =$ exist at least one active role or relationship,
$y =$ no more active role and relationship

Figure 6.18: State machine for Social Role, Social Relationship and SIM Composite

The time when a change cannot be made is the moment when the entity is in Active state. For instance, an adaptation operation cannot be performed in a social role or relationship when a conversation (request/response) associated with these entities is in progress. In that case, the adaptation request will be buffered and executed in the future after the entities enter into the Idle state.
6.2.3 Adaptation across Social Interaction Models

As stated in Section 6.2.1.2, a PSIM provides a coordinated view of all the interactions of an individual across different domains, and an individual plays roles in multiple domains/DSIMs through her PSIM. Thus, an adaptation in a DSIM should be propagated to its corresponding PSIMs. Fig. 6.19 shows a basic protocol for such adaptation propagation. In this protocol, the DSIM organiser triggers the adaptation in a PSIM by invoking the following methods: triggerRoleAcquisition, triggerRoleRelinquishment and triggerUpdateRelationship.

![Figure 6.19: Cross-DSIM/PSIM adaptation propagation](image)

The adaptation across social interaction models has three aspects:

**Binding a player to a social role in a DSIM** — When a player binds to a role in a DSIM, her PSIM should add that role and its role-centric relationship. Thus, for the bindRolePlayer request, the DSIM invokes computeRoleCentricRel method to compute the role-centric relationship of a particular social role which is the aggregation of all the relationships associated with that social role in the DSIM (see Algorithm. [1]). Then the DSIM invokes the triggerRoleAcquisition method in the PSIM organiser with the social role and its role-centric relationship, as parameters. The PSIM organiser executes the roleAcquisition method to adapt its structure by adding a social role and relationship based on the received
The `roleAcquisition` method (see Algorithm 2) first adds a social role based on the received information. After that it checks the state of the coordinator role using the `getRoleState` management method. When the coordinator role enters into the `Idle` state which is the safe state to perform adaptation, the state of the coordinator role is set to the `Reconfiguration`. In this state, the coordinator role does not accept any incoming message and thus, it is safe to perform adaptation. A relationship between the added social role and the coordinator role is added based on the received information. Once the adaptation is completed, the state of the coordinator role is set back to the `Idle` state.

**Algorithm 2: roleAcquisition (r, rel, url)**

```
begin
1  addSocialRole(r)                /* adding a social role */
2  bindRolePlayer(r, url)         /* checking until CoordinatorRole enters into Idle state */
3  crState ← getRoleState(“CoordinatorRole”)  /* wait for some time */
4  while crState ≠ Idle do
5   wait()                         /* wait for some time */
6   crState ← getRoleState(“CoordinatorRole”)  /* wait for some time */
7  setRoleState(“CoordinatorRole”, Reconfiguration) /* Setting CoordinatorRole state as Reconfiguration so that adaptation can be performed safely */
8  addRelationship(r,”CoordinatorRole”,rel) /* adding a relationship (rel) between roles r and CoordinatorRole */
9  setRoleState(“CoordinatorRole”, Idle) /* adaptation complete so set back the state to Idle */
```

For instance, to add a new car, say Car#3, as a following car in the `ConvoyDSIM`, the `ConvoyDSIM` organiser player invokes (using organiser interface) the following management methods:

- `addSocialRole (“FC2”)` – to add another following car role to the `ConvoyDSIM`,

- `addSRelationship(“LC”, “FC2”, relName) ` – to add a relationship between the
LC and FC2 social roles,

- `addInteraction`, `addConversation`, `addObligation` and `addOperationalParam` – to add functional and non-functional information to the relationship as required, and

- finally, `bindRolePlayer` – to bind the third vehicle to the FC2 role.

Because the `bindRolePlayer` is invoked, the ConvoyDSIM (as part of the cross adaptation process) computes the role-centric relationship of the FC2 (say Rr5) using the Algorithm 1 and invokes the `triggerRoleAcquisition` method in Car#3PSIM with FC2 and Rr5 as parameters. As a result, the Car#3PSIM updates its structure by adding the FC2 role and the Rr5 role-centric relationship between the FC2 and Coordinator roles.

**Unbinding a player from a social role in a DSIM** — When a player is unbound from the DSIM, her PSIM should be adapted by removing that role and its role-centric relationship. Thus, for the `unbindRolePlayer` request the DSIM invokes the `triggerRoleRelinquishment` method with the social role name as the parameter. The PSIM organiser executes `roleRelinquishment` method to adapt its structure by removing the social role, and the relationship between that social role and the coordinator role, when those entities are in `Idle` state (i.e., safe to delete).

The `roleRelinquishment` method (see Algorithm 3) invokes the `getRelationship` management operation to get the relationship name between the target social role and the coordinator role. Then, it checks the state of the relationship (using the `getRelationshipState` management method) and waits until it enters to the `Idle` state which is the safe state to perform adaptation. Once the relationship enters into the `Idle` state, the state of the relationship is set to `Reconfiguration`. After that the adaptation is performed by removing the relationship, unbinding player from the target social role, and finally removing the social role.

For instance, if the leading car (Car#1) breaks-down, the ConvoyDSIM organiser player invokes (using the organiser interface) the `unbindRolePlayer` method to unbind Car#1PSIM from the LC. As a consequence, the ConvoyDSIM (as part of the
cross adaptation process) invokes the `triggerRoleRelinquishment("LC")` method in Car#1PSIM. As a result, the Car#1PSIM updates its structure by removing the LC role and the relationship between the LC and Coordinator roles.

**Updating a social relationship in a DSIM** — All the updates in a relationship and/or a social role in a DSIM should be propagated to the corresponding PSIM(s). Thus, for any modification request in a relationship (i.e., to add, delete or update an interaction, conversation or obligation), the DSIM organiser invokes the `triggerUpdateRelationship` method in the PSIMs which are bound to the associated social roles in that relationship. Then the PSIM organiser executes the `updateRelationship` method to reflect the changes.

The `updateRelationship` method (see Algorithm 4) invokes the `getRelationship` management operation to get the target relationship name which needs to be updated. After that, it checks the relationship state and waits until the relationship enters into the *Idle* state. When the relationship enters into the *Idle* state which is the safe state to perform adaptation, the state of the relationship is set to the *Reconfiguration*. Based on the update request such as `addInteraction`, `deleteConversation`, `updateOperationalParam`, and so on, the relationship is modified. Once the adaptation is completed, the state of the relationship is set back to the *Idle* state.

For instance, assume that during convoy it is started raining heavily, and as a
consequence the adaptation rule defined in section 6.2.2.3 is fired or the ConvoyDSIM organiser player (using organiser interface) invokes the updateOperationalParam(R1, maxDesiredDistance, 600m) method in ConvoyDSIM to change the maxDesiredDistance parameter value from 1km to 600m. Both the Car#2PSIM and Car#3PSIM are related to the R1 relationship. Thus, the ConvoyDSIM (as part of the cross adaptation process) invokes the triggerUpdateRelationship method in Car#2PSIM and Car#3PSIM to update the maxDesiredDistance value in both of these PSIMs.

6.3 Summary

In this chapter, we have presented our approach to preserving privacy in accessing social context information and to managing runtime adaptation in social interaction models.

We have proposed a way to preserving owners’ privacy by allowing the owners to fine-tune the granularity of information access and to specify access control policies. In this regard, we have presented an ontology-based socially-aware access control policy model and language for owners to control access to their information. Our policy model provides intuitive support by considering social role, social relationship and situation information when defining privacy preferences, and allowing owners to fine-tune the granularity of accessing information.

We have also presented a way of managing runtime adaptation in social interaction models. Our explicit modelling of social interactions from domain- and player-perspectives, and separation of functional and management operations facilitate the management of dynamic interaction relationships between collaborative persons, and support adaptation in such runtime models to cope with the continuous changes in requirements and environments. The adaptation in a runtime social interaction model (i.e., DSIM or PSIM) can be structural and parametric, and realised through the management (organiser) interface of the runtime model. The inherent dependencies between runtime models, i.e., DSIMs and PSIMs, are maintained and kept consistent through coordinated cross-model adaptation (i.e., adaptation propagation).
CHAPTER 6. SOCIAL CONTEXT MANAGEMENT

Algorithm 4: updateRelationship(r, updateType, data)

Data: r is a social role name related to the relationship that needs to be updated, updateType is the type of updates in the relationship such as addInteraction, deleteConversation, and so on (any of the operations listed in Table 6.2 under category of modifying social relationship), data is the information that needs to be updated.

begin
1   relName ← getRelationship(r, "CoordinatorRole")  /* getting the relationship name between role r and CoordinatorRole */
2   relState ← getRelState(relName)
3   while relState ≠ Idle do  /* checking until the relationship rel enters into Idle */
4       wait()  /* wait for some time */
5       relState ← getRelState(relName)
6   setRelState(relName, Reconfiguration)  /* setting relationship relName state to the Reconfiguration so that the adaptation can be performed safely */
7   if updateType = "addInteraction" then
8       addInteraction(relName, data)
9   else if updateType = "addObligation" then
10      addObligation(relName, data)
11   else if updateType = "addConversation" then
12      addConversation(relName, data)
13   else if updateType = "deleteInteraction" then
14      deleteInteraction(relName, data)
15   else if updateType = "deleteObligation" then
16      deleteObligation(relName, data)
17   else if updateType = "deleteConversation" then
18      deleteConversation(relName, data)
19   else if updateType = "updateInteraction" then
20      updateInteraction(relName, data)
21   else if updateType = "updateConversation" then
22      updateConversation(relName, data)
23   else if updateType = "updateObligation" then
24      updateObligation(relName, data)
25   else if updateType = "addOperationalParam" then
26      addOperationalParam(relName, data)
27   else if updateType = "updateOperationalParam" then
28      updateOperationalParam(relName, data)
29   else if updateType = "deleteOperationalParam" then
30      deleteOperationalParam(relName, data)
31   setRelState(relName, Idle)  /* adaptation complete so set back the state to Idle */
Part III
Chapter 7

**SocioPlatform Architecture and Prototype Implementation**

In Chapter 4 and Chapter 5, we have presented the modelling of and reasoning about different types of social context information, whilst Chapter 6 has shown how we manage the privacy in accessing social context information and support the adaptation in runtime social interactions. These three chapters have collectively presented the models, languages and techniques underlying the framework for developing socially-aware applications. In this chapter, we present a platform, called SocioPlatform, to provide high-level support for developing socially-aware applications by acquiring, reasoning, storing and provisioning different types of social context information, and managing their runtime adaptation.

![Figure 7.1: SocioPlatform overview](image-url)
The SocioPlatform can be divided into two main parts: Social Context Information Management (SCIM) and Social Interaction Management (SIM), as shown in Fig. 7.1. Collectively these two parts provide support to building socially-aware applications of two different focuses, data-centric and interaction-centric.

Social Context Information Management (SCIM) supports data-centric socially-aware applications (e.g., the socially-aware phone call application) by collecting social context information (SCI) from different sources, reasoning about more meaningful SCI (as required by the applications) based on collected SCI, storing SCI into a knowledge base, and supporting access to such SCI while considering information owners’ privacy preferences. SCIM also provides functionality to infer situations from users’ interaction events in social media and interaction-centric socially-aware applications (e.g., cooperative convoy application). SCIM acquires interaction events from social media, while SCIM receives interaction events in interaction-centric socially-aware applications from SIM (see Section 7.4.5).

Social Interaction Management (SIM) provides the runtime environment and adaptation management for the interaction-centric socially-aware applications (e.g., the cooperative convoy application). In our approach, the agreements between collaborative users in a socially-aware application are captured in the social interaction ontology and stored in the SCIM knowledge base. The information in the ontology is used to create both the domain-centric social interaction models (DSIMs) and player-centric social interaction models (PSIMs) (see Section 7.4.4). SIM provides runtime environment for the application to instantiate these models. These instantiated runtime models mediate social interactions among collaborative users according to their agreements. SIM also supports the runtime adaptation of such models to cope with the changes in users’ requirements and environments.

The rest of this chapter is organised as follows. Section 7.1 presents the architecture and design of the SCIM followed by its prototype implementation in Section 7.2. The SIM architecture and design is presented in Section 7.3 while its prototype implementation is presented in Section 7.4. Section 7.5 summarises this chapter.
CHAPTER 7. SocioPlatform ARCHITECTURE AND PROTOTYPE IMPLEMENTATION

7.1 Architecture and Design of the SCIM

The SCIM architecture comprises two layers: (i) information acquisition layer and (ii) information management layer, as shown in Fig. 7.2.

![Figure 7.2: Architecture of the social context information management (SCIM)](image)

The information acquisition layer is responsible for acquiring social context information from various sources such as Google Calendar, Facebook, LinkedIn, Twitter and other social media. A common interface, called information acquisition interface, is provided so that different adapters can be built based on that interface to collect data from various sources. Section 7.1.1 describes this interface. The acquisition module is responsible for managing and operating adapters to fetch raw data from the different sources, make the data consistent (i.e., remove irrelevant data, integrate data of interest, and so on) and store it into the knowledge base (see Sections 7.1.2 and 7.1.3 for the details discussion).

The information management layer is responsible to store users’ social context information as acquired and to preserve their privacy when accessing this information. This layer consists of three main modules: (1) Reasoning module — classifies users’ so-
cial data that is collected by the information acquisition layer, stores it into the SCOnto knowledge base, and provides a reasoning functionality to infer abstract social context information that is of applications interest; (2) Privacy policy module — provides a policy interface (Section 7.1.4 provides details) to allow users to specify and manage their privacy preferences, and supports policy enforcement in accessing users’ social context information; and (3) Query processing module — allows different applications to access users’ social context information and provides a query interface (Section 7.1.5 provides details) so that application developers can build applications without the need to deal with the details of information representation schema and management.

### 7.1.1 Information Acquisition Interface

We propose a set of information acquisition application programming interfaces (APIs) to collect and store information relevant to the concepts defined in the ontologies where a specific type of information can be collected from different sources. These APIs decoupled the information representation and management from the underlying information acquisition process. They provide a common interface to implement various adapters for different data sources. Table 7.1 lists the APIs for collecting different types of information and their possible sources. An API is used to collect a specific type of social context information and can be implemented by one or more adapters such as those for Facebook, LinkedIn, and so on, because one sensor may not provide all the attribute values of a particular type of information. For instance, we can acquire users’ profile information from Facebook, or LinkedIn, or Twitter. Without special agreement and service charge, Facebook allows one to fetch a person’s email address but not the mobile phone number even though the person gives permission. On the other hand, LinkedIn allows one to fetch mobile phone number but not email address. Moreover, a person may not be a user of or have an account in, all the online social networks (OSNs). Thus, we analyse some of the prominent OSNs and categorize them based on their richness and accessibility to acquire different types of SCI.

As reported in Table 7.1, we can acquire a person’s profile, status and event based relationship information from Facebook, LinkedIn or Twitter. Google Calendar can also be a possible source of situation information as people use it to maintain their sched-
### Table 7.1: Information acquisition APIs for accumulating social context information

<table>
<thead>
<tr>
<th>No</th>
<th>Acquisition APIs</th>
<th>Adapters</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>getProfile(...)</td>
<td>Facebook, LinkedIn, Twitter</td>
</tr>
<tr>
<td>2</td>
<td>getStatusUpdate(...)</td>
<td>Facebook, LinkedIn, Twitter, Google Calendar</td>
</tr>
<tr>
<td>3</td>
<td>getEventBasedRel(...)</td>
<td>Facebook, LinkedIn, Twitter</td>
</tr>
<tr>
<td>4</td>
<td>getWorkRelationships(...)</td>
<td>Facebook, LinkedIn</td>
</tr>
<tr>
<td>5</td>
<td>getEducationRelationships(...)</td>
<td>Facebook, LinkedIn</td>
</tr>
<tr>
<td>6</td>
<td>getLivingAddrBasedRel(...)</td>
<td>Facebook, LinkedIn</td>
</tr>
<tr>
<td>7</td>
<td>getGroupInterestBasedRel(...)</td>
<td>Facebook, LinkedIn</td>
</tr>
<tr>
<td>8</td>
<td>getFamilyRelationships(...)</td>
<td>Facebook</td>
</tr>
<tr>
<td>9</td>
<td>getResearchTopicBasedRel(...)</td>
<td>LinkedIn</td>
</tr>
<tr>
<td>10</td>
<td>getFollowingFollowerRel(...)</td>
<td>Twitter</td>
</tr>
<tr>
<td>11</td>
<td>getDigitalContentBasedRel(...)</td>
<td>Facebook</td>
</tr>
<tr>
<td>12</td>
<td>getInteractionEvent(...)</td>
<td>Facebook, LinkedIn, Twitter</td>
</tr>
</tbody>
</table>

The relationships based on work, education, living address and group interest can be acquired from Facebook or LinkedIn. A person’s family, research topic, and following-follower relationships can be collected from her Facebook, LinkedIn, and Twitter accounts, respectively. In Facebook, people usually share various digital contents such as photos, audios and videos, and other Facebook users express their impression (e.g., like, dislike) about these contents. Thus, we can acquire the digital content based relationships from Facebook.

In the following two sections, we describe how the raw social data can be collected from different sources and integrated to create a consistent representation of social context information.

### 7.1.2 Designing Adapters to Fetch Raw Data

Online social networks have the potential to act as social data sources, providing us the opportunity to access user’s social context information. In this research, we consider a number of social data sources such as Facebook, LinkedIn, Twitter and Google Calendar, and have built adapters for them by implementing the information acquisition APIs (as listed in Table 7.1).
Facebook provides a Facebook Query Language (FQL)\(^1\) for fetching data from individual tables in their system. Thus, we cannot fetch all the data of our interest using a single request. Furthermore, fetching information is a time consuming task and has a high network overhead (see section [9.2.1](#)). We have taken into account this issue in designing the adapter for Facebook and tried to minimize the number of data fetching requests to the lowest level without compromising the data of interest. For Facebook, we require three fetch requests to collect people-centric relationships: (1) fetch user profile data, (2) fetch classified relationships (i.e., relationships under the built-in and user defined classifications), and (3) fetch all connections in the friend list.

In Facebook, the general friend list also includes people who are already classified as family members or colleagues. Thus, to remove duplication, we compare the connections in the friend list with the connections in the classified relationship lists, and remove the connections which are already classified. The result of that process gives us a list of friends/connections who belong to a general friend category. Furthermore, we match the user’s profile with its friends’ profiles to deduce more specialized relationships such as university friend (if they studied in a same institution), colleague (if they work in a same organization), living address based friend (if they live in a same city), and so on.

Similarly, we have designed adapters for LinkedIn, Twitter and Google Calendar by implementing the APIs as shown in the Table 7.1.

### 7.1.3 Fetching and Processing Raw Data for Consistent Social Context Information

The acquisition module (see Fig. [7.2](#)) is responsible for fetching raw social data from diverse sources, processing and storing it into knowledge base. The acquisition module keeps the list of the available adapters and their implemented APIs. After a certain time interval, which is configurable, the acquisition manager executes the fetching and processing steps to update the knowledge base with users’ recent social context information. The frequency for such information update, however, is not the same for all

---

\(^1\)https://developers.facebook.com/docs/reference/fql/
types of social context information. For instance, a user’s situation information may need to be updated more frequently compare to his/her family relationship. To fulfil that requirement, we allow users to schedule the execution of different acquisition functions listed in Table 7.1. Moreover, it is also possible to trigger the execution of these functions manually (i.e., on demand).

After fetching raw social data, the acquisition manager processes the data before storing it into the knowledge base. To keep social context information consistent, raw data fetched from the various sources needs to be further processed, e.g., performing cross integration among the data from different sources. The data processing depends on the types of raw social data and their possible sources. For instance, we fetch a user’s situation information from her schedule on Google Calendar and her status updates in Facebook, LinkedIn and Twitter. To identify users’ current “situation” more accurately, we integrate the data fetched from these sources based on their update time-stamp, and select the most related and recent update as users’ current situation.

7.1.4 Policy Interface

Table 7.2 lists the set of Policy APIs that allow users’ applications to add, delete, retrieve and update their privacy policies.

<table>
<thead>
<tr>
<th>No</th>
<th>Policy APIs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>getAllPolicyIDs()</td>
<td>Returns all the specified policy IDs as a list</td>
</tr>
<tr>
<td>2</td>
<td>getPolicyStatement(pID)</td>
<td>Returns pID policy statement as a string from the SACOnto knowledge base</td>
</tr>
<tr>
<td>3</td>
<td>addPolicy(P)</td>
<td>Adds a policy statement P to the SACOnto knowledge base and returns its policy ID</td>
</tr>
<tr>
<td>4</td>
<td>deletePolicy(pID)</td>
<td>Deletes the pID policy from the SACOnto knowledge base</td>
</tr>
<tr>
<td>5</td>
<td>updatePolicy(pID,P)</td>
<td>Replaces the pID policy with the new policy statement P</td>
</tr>
</tbody>
</table>

To visualize all the specified policies to a user, the application first invokes the getAllPolicyIDs operation which returns a list of policy IDs. For each of these IDs, the
application invokes `getPolicyStatement` to retrieve the associated policy statement. To add a new policy to the user’s SACOnto knowledge base, the application invokes the `addPolicy` operation and passes a policy statement as a string. The `deletePolicy` operation receives a policy ID and delete the associated policy. The `updatePolicy` operation replaces an existing policy with a new policy statement.

7.1.5 Query Interface

The query interface provides a set of basic query APIs to allow applications to access both context and meta-context information. These functions or APIs can be further used to build more complex social inferences or queries. These APIs can be divided into two main categories: (1) APIs for querying social context information (listed in the Table 7.3) and (2) APIs for querying meta-context information (listed in the Table 7.4).

**APIs for querying social context information (No. 1 to No. 19)** – In this category, the first seven (i.e., No. 1 to No. 7) APIs are boolean functions, and among those No.7 is the most generic function. It takes a relationship type R and two person ids as input and checks whether the relationship R exists between these two persons. Other six APIs, i.e., No.1 to 6, are a specialized type of the No.7 API, which check the existence of a particular type of relationship between two persons A and B. For example, the No.1, 2 and 3 APIs check the existence of a colleague, friend and family relationships respectively. The No.8 and 9 APIs find an individual person based on his/her phone number and email id, respectively. The No. 10 to 13 APIs return a list of persons with whom an individual A has a particular type of relationship. Among those the No. 13 API is the most generic function as it can answer the query on any relationship type R for an individual A. The No. 10, 11, and 12 APIs are a specialized type of No.13 API, defined for a particular type of relationship, i.e., close friend, bosom friend and best friend, respectively. The No. 14 and 15 APIs return an attribute value of the relationship R such as strength and trust. The No. 16 API returns all the types of relationship between two persons while the No. 17 API returns the relationship type at a certain granularity level. The No. 18 and 19 APIs are defined to access a person’s current situation information. The No. 18 API returns the exact situation while the No. 19 API returns the situation
### Table 7.3: Query APIs for accessing social context information

<table>
<thead>
<tr>
<th>No</th>
<th>Query APIs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>isAColleagueOfB(A,B)</code></td>
<td>Boolean function that checks whether a person A is a colleague of B</td>
</tr>
<tr>
<td>2</td>
<td><code>isAFriendOfB(A,B)</code></td>
<td>Checks whether person A is a friend of B</td>
</tr>
<tr>
<td>3</td>
<td><code>isAFamilyOfB(A,B)</code></td>
<td>Checks whether person A is a family member of B</td>
</tr>
<tr>
<td>4</td>
<td><code>isAandBLivingSameCity(A,B)</code></td>
<td>Checks whether person A and B are living in a same city</td>
</tr>
<tr>
<td>5</td>
<td><code>isAandBLivingSameCountry(A,B)</code></td>
<td>Checks whether person A and B are living in a same country</td>
</tr>
<tr>
<td>6</td>
<td><code>isAHasRelationshipWithB(A,B)</code></td>
<td>Checks whether person A has any relationship with B</td>
</tr>
<tr>
<td>7</td>
<td><code>isRelRExistBetAandB(R,A,B)</code></td>
<td>Checks whether a relationship R exists between A and B</td>
</tr>
<tr>
<td>8</td>
<td><code>getPersonByPhoneNum(phNum)</code></td>
<td>Returns a person’s id whose phone number is <code>phoneNum</code></td>
</tr>
<tr>
<td>9</td>
<td><code>getPersonByEmailID(emailID)</code></td>
<td>Returns a person’s id whose email address is <code>emailID</code></td>
</tr>
<tr>
<td>10</td>
<td><code>getCloseFriends(A)</code></td>
<td>Returns close friends list of person A</td>
</tr>
<tr>
<td>11</td>
<td><code>getBosomFriends(A)</code></td>
<td>Returns bosom friends list of person A</td>
</tr>
<tr>
<td>12</td>
<td><code>getBestFriends(A)</code></td>
<td>Returns best friends list of person A</td>
</tr>
<tr>
<td>13</td>
<td><code>getAllRelatedPersonsByR(A,R)</code></td>
<td>Returns all related by the relationship R</td>
</tr>
<tr>
<td>14</td>
<td><code>getStrengthOfRel(A,R)</code></td>
<td>Returns the strength value of the relationship R</td>
</tr>
<tr>
<td>15</td>
<td><code>getTrustOfRel(A,R)</code></td>
<td>Returns the trust value of the relationship R</td>
</tr>
<tr>
<td>16</td>
<td><code>getAllRelationshipsName(A,B)</code></td>
<td>Returns a list of relationships name between person A and B</td>
</tr>
<tr>
<td>17</td>
<td><code>getRelByGranularity(A,B,gLevel)</code></td>
<td>Returns the name of a relationship between person A and B at <code>gLevel</code> granularity</td>
</tr>
<tr>
<td>18</td>
<td><code>getSituation(A)</code></td>
<td>Returns current situation of person A</td>
</tr>
<tr>
<td>19</td>
<td><code>getSituAtGranularity(A,gLevel)</code></td>
<td>Returns current situation of person A at <code>gLevel</code> granularity</td>
</tr>
</tbody>
</table>
Table 7.4: Query APIs for accessing meta-context information

<table>
<thead>
<tr>
<th>No</th>
<th>Query APIs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>getNumberOfFriends(A)</td>
<td>Returns person A’s number of friends</td>
</tr>
<tr>
<td>21</td>
<td>getNumberOfConnections4R(A,R)</td>
<td>Returns person A’s number of connections for a relationship name R</td>
</tr>
<tr>
<td>22</td>
<td>getNumberOfRelationships(A,B)</td>
<td>Returns a number of relationship types between A and B</td>
</tr>
</tbody>
</table>

Information at a certain granularity level.

APIs for querying meta-context information (No. 20 to No. 22) – In this category, the No. 20 and 21 APIs return the number of an individual’s connections in a specific type of relationship. The No. 20 API returns the number of connections in a friend relationship, while the No. 21 API is a generic API as it allows one to input the name of the relationship type. The No. 22 API returns the number of relationships between two persons.

7.1.6 Rule Interface

Table 7.5 lists the set of Rule APIs that can be used to develop a graphical user interface application which can be used by the domain experts to add, delete, retrieve and update reasoning rules, e.g., situation specification rules for inferring situations from interaction events.

Table 7.5: Rule APIs for managing reasoning rules

<table>
<thead>
<tr>
<th>No</th>
<th>Rule APIs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>getAllRuleIDs()</td>
<td>Returns all the rule IDs as a list</td>
</tr>
<tr>
<td>2</td>
<td>getRule(rID)</td>
<td>Returns the rule with ID rID as a string</td>
</tr>
<tr>
<td>3</td>
<td>addRule(R)</td>
<td>Adds a rule R to the reasoning module and returns its rule ID</td>
</tr>
<tr>
<td>4</td>
<td>deleteRule(rID)</td>
<td>Deletes the rID rule from the reasoning module</td>
</tr>
<tr>
<td>5</td>
<td>updateRule(rID,R)</td>
<td>Replaces the rID rule with the new rule R</td>
</tr>
</tbody>
</table>

To obtain all the specified reasoning rules, the application first invokes the getAll-
RuleID operation which returns a list of rule IDs. For each of these IDs, the application invokes getRule to retrieve the associated rule. To add a new rule to the reasoning module, the application invokes the addRule operation and passes a rule as a string. The deleteRule operation receives a rule ID and delete the corresponding rule. The updateRule operation replaces an existing rule with a new rule.

7.2 SCIM Prototype Implementation

We have implemented a SCIM prototype, in Java 2 Platform Standard Edition (J2SE). The following sections present the details of this implementation.

7.2.1 Implementing Ontologies

We have used the Protégé tool\textsuperscript{2} to create the SCOnto, SACOnto and IntEO ontologies. Fig. 7.3 shows a screenshot of the concepts in the SCOnto ontology (left panel), the granularity specification of the Chatting concept, the inference rules for the Father-Daughter and Busy concepts, and the semantic description of the Relationship concept (from top to bottom in the right panels). SCOnto contains 126 classes, 8 object properties, 39 data properties and 619 axioms.

Users’ privacy policies are stored in their SACOnto knowledge base. The policy statement is formatted in Manchester OWL syntax\textsuperscript{14} and stored as a string. The Manchester OWL syntax has been designed for writing OWL-DL class expressions in natural language. Fig. 7.4 shows a screenshot of the SACOnto ontology and Policy#1 written in Manchester OWL syntax. SACOnto contains 168 classes, 15 object properties, 41 data properties and 667 axioms.

The Protégé screenshot in Fig. 7.5 shows the concepts (left panel) and object properties (top right panel) in the IntEO ontology. IntEO contains 144 classes, 19 object properties, 38 data properties, 48 individuals and 623 axioms.

\textsuperscript{2}http://protege.stanford.edu/
Figure 7.3: A screenshot of SCONto in Protégé
CHAPTER 7. SocioPlatform ARCHITECTURE AND PROTOTYPE IMPLEMENTATION

Figure 7.4: A screenshot of SACOnto and an example of formatting policy (Policy#1) in Manchester Syntax [14], using Protégé
Figure 7.5: A screenshot of IntEO in Protégé
7.2.2 Implementing Adapters

We have written adapters for Facebook, LinkedIn, Twitter, and Google calendar using restfb 1.6.7, linkedin-j 1.0.415, twitter4j 2.2.4, and gdata-calendar 2.0 respectively, to fetch users’ social data. These adapters implement the information acquisition APIs and are managed by the acquisition manager module.

We follow the OAuth 2.0 protocol for authorization in fetching users’ social data. OAuth is an open standard protocol which allows users to share their private resources (e.g., profiles, friend lists) stored on one site with another site without having to hand out their credentials, typically supplying tokens instead of user name and password. Each token grants access to a specific site (e.g., Facebook) for specific resources (e.g., friend list and/or status updates) and for a defined duration (e.g., the next 2 hours).

We have created applications (see Fig. 7.6), called SCIMiddlewareF, SCIMiddlewareL, SCIMiddlewareT and SCIMiddlewareG, on Facebook, LinkedIn, Twitter, and Google respectively. Through these applications SCIM acquires access tokens and gets authoriza-

---

3http://restfb.com/
4http://code.google.com/p/linkedin-j/
5http://twitter4j.org/en/index.html
6http://code.google.com/p/gdata-java-client/
7http://oauth.net/2/
tion from users to fetch their social data from those sites.

### 7.2.3 Implementing Rule APIs and Reasoning Module

The reasoning module implementation has three main aspects: (1) implementation of the description logic based reasoning component, (2) implementation of the rule management APIs and (3) implementation of the situation inference component, as shown in Fig. 7.7.

The DL based reasoning component employs DL reasoners (listed in the Section 7.2.4) to execute reasoning rules, described in the Section 5.2.1, 5.2.2 and 5.3.1 for reasoning about different types of social context information such as friend relationships, family relationships and situations. The rule management component implements the rule management APIs (described in the Section 7.1.6) which allow domain experts to add, delete, retrieve and update the reasoning rules corresponding to the situation inference component and the DL based reasoning component.
The situation inference component combines complex event processing (CEP) and semantic matching in order to infer users’ situations at (near) real time from their interaction events. The information acquisition layer collects interaction events from the different social interaction platforms such as Facebook, Email, and instant messaging. The rule translation unit translates situation rules specified by the domain experts or application developers, to the executable format that compliant with the CEP engine.

The complex event processing engine is responsible for identifying situations by checking the situation specification rules (SSRs) defined by domain experts or application developers, and for inserting the identified situation into the SConto knowledge base. We employ the open source software Drools 5.5\footnote{http://www.jboss.org/drools/drools-fusion.html}, which includes Drools expert and Drools fusion, for implementing the CEP engine. There are many advantages of using Drools. Firstly, Drools provides an integrated platform for processing events, executing rules, and checking temporal correlation among events. Secondly, the Drools engine is implemented based on the RETE algorithm\footnote{[82]}. RETE is an efficient pattern matching algorithm which is asymptotically independent from the number of events or facts that is being inserted to the rules engine as long as the number of rules matched by these events are fixed. The RETE algorithm allows the Drools engine to maintain a working memory of the results of partial rule matches across time. Re-evaluation of each condition in the rules is no longer necessary, as the engine knows which conditions might possibly change for each event, and only those are re-evaluated. This improves the efficiency of the situation reasoning platform (see experimental result in Fig. 9.5). Thirdly, Drools supports user-defined functions in its rule language. This enables us to realise the constraints with ONTOEQ in the interaction event patterns using a function call. For example, the following constraint in the IE2 interaction event pattern

\[ \text{hasType ONTOEQ “PostingSomeThing”} \]

is realised as

\[ \text{semanticMatching(hasType,"PostingSomeThing")} \]

The domain expert specifies situation rules by following the human-readable situation specification language (SituLang), described in the Section 5.4.6. The rule translation
unit automatically translates the situation rules (*.dslr) specified in the SituLang to the Drools executable format (*.drl). We have written a rule translation scripts (see Appendix D), namely RuleTrans.dsl, using the language construct provided by the Drools. For instance, Fig. 7.8 shows the transformed watching game situation rule (presented in the Section 5.4.7) in the Drools executable format. The rule in this figure also presents how we realise and implement ONTOEQ of our situation specification language and support the semantic matching of the interaction event attributes in complex event processing for inferring situations.

```java
@global IEEventKB kbIns; // an instance of the IntEO KB
@rule "Watching Game"
  @when
  $IE1:InteractionEvent(hasType="PostingCurrentLocation" &&
    eval(kbIns.semanticMatching(atPlace,"Stadium")),
    personName:from,place:atPlace)
  $IE2:InteractionEvent(eval(kbIns.semanticMatching(hasType,"PostingSomeThing")) &&
    eval(kbIns.ontologicalMatch(about,"Sports")) &&
    has occurred "after" $IE1)
  @then
  kbIns.notifySITU("Person":personName=" is watching game at the "+place);
@end
```

Figure 7.8: The Drools executable format of the Watching game situation rule

The CEP engine uses the semantic matching functionality to match event patterns (in situation rules) with incoming interaction event streams. The semantic matching is an ontological query on the IntEO. Fig. 7.9 presents the implementation of the semantic matching function in Java. It is a boolean function and returns true if the value of the attrValue parameter is a subclass (in the IntEO) of the value of the constant parameter. Like the query processing module, we have used the derivo 1.09 SPARQL-DL query engine with the DL reasoners for implementing the semantic matching function.

At runtime, the CEP engine continuously receives users’ interaction events from social media and/or SIM, and incrementally matches the interaction event patterns specified in the situation rules. The engine invokes the semantic matching function if it is stated in the situation rules. Once a rule is matched, the situation is identified and inserted into SCOnto. The socially-aware applications use these situations to provide services to the users.

9http://www.derivo.de/en/resources/sparql-dl-api/
7.2.4 Implementing Policy APIs and Privacy Policy Module

We have implemented a set of Policy APIs (as listed in Table 7.2) that allow users’ applications to add, delete, retrieve and update their privacy policies. The privacy policy module (see Fig. 7.2) implements these APIs. We have used the OWL API 3\(^{10}\) to implement a knowledge base for SACOnto, and to manage this knowledge base, i.e., adding, deleting and updating privacy policies.

The application developer can develop a web-, mobile- or desktop-based graphical user interface based on these APIs for users to manage their policies. To visualize all the specified policies to an user, the application first invokes the `getAllPolicyIDs` operation which returns a list of policy IDs. For each of these IDs, the application invokes `getPolicyStatement` to retrieve the associated policy statement. These two operations are basically queries over the SACOnto knowledge base. Thus, like query APIs, we have used SPARQL-DL (see Section 7.2.5 for details) to represent queries related to these operations. To add a new policy to the user’s SACOnto knowledge base, the application invokes the `addPolicy` operation and passes a policy statement as a string. The `addPolicy` function (see Fig. 7.10 for a Java code snapshot) implements the steps described in Section 6.1.2. We have used the DLQuery parser in OWL API 3 to convert the string policy statement into an OWL-DL class expression. Finally, we insert the expression into the knowledge base. For the `deletePolicy` operation, first we delete the policy expression

\(^{10}\)http://owlapi.sourceforge.net/
from its equivalent class which is a subclass of the Access class. Then, we delete the pID individual from the Policy class and its link to the policy statement. For the updatePolicy operation, we follow the steps of the deletePolicy operation followed by an addPolicy operation.

```java
public String addPolicy(String strPolicy) {
    //transform policy into class expression
    OWLClassExpression owlPolicy = parseClassExpression(strPolicy);
    OWLClass decisionClass = getDecisionClass(strPolicy);
    // adding policy as an equivalent class of the corresponding decision class
    OWLEquivalentClassesAxiom ax = datafac.getOWL EquivalentClassesAxiom(OWLClass, owlPolicy);
    manager.addAxiom(SACOnto, ax);
    String pID = getPolicyID(strPolicy);
    // checking consistency
    boolean consistent = reasoner.isConsistent();
    if (consistent) {
        // adding policy makes SACOnto inconsistent, thus delete policy and return null
        deletePolicy(pID);
        return null;
    } else { //adding policy makes SACOnto consistent, thus return policy id
        return pID;
    }
}
```

Figure 7.10: A Java code snapshot for the addPolicy function

The privacy policy module also implements the executePolicy function that follows the steps described in Section 6.1.3. This function checks users’ privacy preferences and returns the access decision. To implement this function, we have used reasoners compliant with the OWL2 DL. In particular, for evaluation purposes, we have incorporated five different DL reasoners: Pellet 2.3.0,11 HermiT 1.3.5,12 TrOWL 0.8.1,13 Fact++ 1.5.2,14 and RacerPro 2.1.15. We have also used these reasoners in checking inconsistency during policy specification, executing reasoning rules (see Section 7.2.3) and applications’ queries (see Section 7.2.5).

### 7.2.5 Implementing Query APIs and Query Processing Module

The query processing module (see Fig. 7.2) implements the query APIs (described in the Section 7.1.5) to answer applications’ queries. We adopt a description logic (DL) based

---

11http://clarkparsia.com/pellet/
12http://hermit-reasoner.com/
13http://trowl.eu/
14http://owl.man.ac.uk/factplusplus/
15http://www.racer-systems.com/products/racerpro/
query language, namely SPARQL-DL [173], for query processing and have used the derivicio 1.0 SPARQL-DL query engine with the above mentioned DL reasoners. Recently, SPARQL-DL was introduced as a rich query language for OWL 2, which is a distinct subset of SPARQL (a RDF based language), tailored to ontology specific queries. Therefore, different aspects of context such as the existence of a relationship, granularity level of a current situation, and so on, can be answered easily in a simpler fashion. For example, queries related to the No. 1 and No. 19 APIs in the Table 7.3 can be formatted in SPARQL-DL as follows:

No.1# isAColleagueOfB(A,B)
PREFIX sc:<http://www.ict.swin.edu.au/Ontology/SCOnto#>
ASK{
    PropertyValue(A, sc:hasRelationship, ?rel1)
    PropertyValue(B, sc:hasRelationship, ?rel2)
    SameAs(?rel1,?rel2)
    DirectType(?rel1, sc:Colleague)}

No.19# getSituAtGranularity(A,gLevel)
PREFIX sc:<http://www.ict.swin.edu.au/Ontology/SCOnto#>
SELECT ?situ{
    PropertyValue(A, sc:hasSituation, ?situIns)
    DirectType(?situName, ?situIns)
    Annotation(strictSubClassOf(?situName,?situ),
        sc:granularityLevel, gLevel)}

The codification of the No. 19 query operation in Java is shown in the Fig. 7.11

7.3 Architecture and Design of the SIM

The Social Interaction Management (SIM) architecture comprises two main modules: the Model Execution Module (MEM) and the Management Module (MM), as shown in Fig. 7.12. The next two sections describe these two modules in details.
Figure 7.11: A Java code snapshot in Eclipse that implements \textit{getSituationGranularity(A,gLevel)} operation

Figure 7.12: Architecture of the social interaction management (SIM) platform
7.3.1 The Model Execution Module

The Model Execution Module supports instantiating both the domain-centric social interaction models (DSIMs) and player-centric social interaction models (PSIMs). As described in Section 6.2.1, these interaction models capture interaction-oriented relationships among collaborative actors in an interaction-centric socially-aware application. Therefore, at a given moment, multiple social interaction model instances may exist in parallel.

The MEM maintains a representation of all the functional elements of the domain-centric and player-centric social interaction models, called \texttt{DSIMFunctional\_\textsubscript{runtime}} and \texttt{PSIMFuctional\_\textsubscript{runtime}}, respectively. The \texttt{DSIMFunctional\_\textsubscript{runtime}} represents Social Roles, Social Relationships, Interactions, Conversations, Obligations and Operational Parameters. In addition, it contains a request handler, a response handler and a message queue. The \texttt{DSIMFunctional\_\textsubscript{runtime}} is able to (1) handle requests received from players (i.e., applications); (2) allocate requests into a message queue; (3) forward messages to corresponding social roles; (4) evaluate conditions (i.e., conversation and obligation) specified in the relationships; (5) send request to relevant social roles and then to players.

The \texttt{PSIMFuctional\_\textsubscript{runtime}} contains all the components of the \texttt{DSIMFunctional\_\textsubscript{runtime}}. In addition, it contains a special type of social role, called the coordinator role. The \texttt{PSIMFuctional\_\textsubscript{runtime}} bounds to one or more social roles in the \texttt{DSIMFunctional\_\textsubscript{runtime}}(s). In the \texttt{PSIMFuctional\_\textsubscript{runtime}}, all the incoming messages first forwarded to the coordinator role. After evaluating conditions specified in the relationships, the request message is forwarded to the coordinator player. The message is processed further based on the decision of the coordinator player. For instance, generate a reply message and send it to the player from where the message has come.

Both the \texttt{PSIMFuctional\_\textsubscript{runtime}} and \texttt{DSIMFunctional\_\textsubscript{runtime}} maintain the state of their corresponding entities such as social roles, social relationships and the composite (as a whole). As the message passes through these entities, the state is changed according to the state machine described in Section 6.2.2.4.
7.3.2 The Management Module

The Management module supports the runtime adaptation of the instantiated social interaction model. Thus, the management components, called DSIMManagement@Runtime and PSIMManagement@Runtime, are instantiated for each of the DSIMFunctional@Runtime and PSIMFunctional@Runtime. Both of these management components implement basic management operations provided by the organiser interface. In Section 6.2.2.3, we have discussed these operations and classified them as structure, parameter and state related operations. The In Adaptation sub-component of the management component implements the structure and parameter related management operations, while the State Management sub-component implements state related operations.

In addition to these sub-components (i.e., In Adaptation and State Management), the management component contains the Cross Adaptation sub-component which supports the adaptation across social interaction models and implements the algorithms described in the Section 6.2.3. The Cross Adaptation sub-component of the DSIMManagement@Runtime implements the computeRoleCentricRel algorithm, while the Cross Adaptation sub-component of the PSIMManagement@Runtime implements the triggerRoleAcquisition, triggerRoleRelinquishment and triggerUpdateRelationship algorithms. Moreover, the interactions between the Cross Adaptation sub-components of DSIMManagement@Runtime and PSIMManagement@Runtime follow the adaptation propagation protocol described in the Section 6.2.3.

The Management module also supports the platform-level management, i.e., to create, retrieve, delete, deploy and undeploy social interaction models dynamically. These APIs, as listed in the Table 7.6, allows a user to perform administration level management. The platform management component implements these APIs.

7.4 SIM Prototype Implementation

The Social Interaction Management (SIM) prototype can be divided into three main parts: the Deployment Layer, Functional Interaction Layer and the Management Layer, as shown in Fig. 7.13. Collectively these three parts provide the runtime environment for supporting
Chapter 7. SocioPlatform Architecture and Prototype Implementation

Table 7.6: Platform level management APIs

<table>
<thead>
<tr>
<th>No</th>
<th>APIs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>createDSIM(...)</td>
<td>Creates a DSIM</td>
</tr>
<tr>
<td>2</td>
<td>createPSIM(...)</td>
<td>Creates a PSIM that contains only a coordinator role</td>
</tr>
<tr>
<td>3</td>
<td>deleteModel(...)</td>
<td>Deletes a social interaction model</td>
</tr>
<tr>
<td>4</td>
<td>deployModel(...)</td>
<td>Deploys a model in the platform</td>
</tr>
<tr>
<td>5</td>
<td>undeployModel(...)</td>
<td>Undeploys a running model from the platform</td>
</tr>
<tr>
<td>6</td>
<td>getModel(...)</td>
<td>Returns the xml descriptor of a model</td>
</tr>
<tr>
<td>7</td>
<td>uploadModel(...)</td>
<td>Uploads a model descriptor file</td>
</tr>
</tbody>
</table>

mediated social interactions and its adaptation in interaction-centric socially-aware applications.

![Figure 7.13](image)

Figure 7.13: Layers of the social interaction management (SIM) prototype

The Model Execution Module belongs to the Functional Interaction Layer, while the Management Module belongs to the Management Layer. These two layers together are called the SIM-Core. One of the key consideration behind the SIM-Core implementation is to have a runtime environment that is independent of the underlying deployment environment. The main reason for this design principle is to make the SIM-Core reusable among an array of different deployment technologies. Besides, the deployment technologies and standards may evolve over time. For example, apart from SOAP/Web services [19][20], which is the currently supported technology, other alternative technolo-
gies such as Remote Method Invocation (RMI) \cite{133,147}, RESTful Services \cite{78,144}, XML/RPC \cite{29}, Mobile Web services \cite{174} do exist. Therefore, it is important to have SIM-Core implemented in a deployment-technology neutral manner.

As discussed in Section 6.2.1, we adopt and extend the Role Oriented Adaptive Design (ROAD) approach \cite{59} in modelling social interactions. Therefore, it is a logical choice to adopt the ROAD implementation for realising and implementing the SIM platform. Moreover, the current ROAD implementation considers the aspect of independently implementing the runtime environment from the underlying deployment environment, which is an important requirement to implement the SIM-Core as discussed above.

The current ROAD implementation\cite{16} provides a runtime environment, called the ROADfactory \cite{17}, and a deployment environment, called the ROAD4WS \cite{121}. However, we could not employ the ROAD implementation directly for several reasons. Firstly, the ROADfactory supports the instantiation and management of the ROAD composites (the DSIMs and PSIMs in our case), involving roles (social roles in our case), contracts (social relationships in our case) and message routing among different roles based on declarative interaction terms and contractual rules. However, it does not have the concept of the obligation and conversation and their realisation in the message routing mechanism. Moreover, the state machines for the composite and its elements are not implemented. Secondly, the ROADfactory was not implemented to recognise the domain-centric and player-centric views of the ROAD composite. Thirdly, the management operations (organiser role interface) provided by the ROADfactory are limited, in particular, the state, obligation, conversation, operational parameter and cross adaptation related management operations are not supported. Furthermore, the ROADfactory does not provide support for the platform level management.

We address these limitations in implementing the corresponding modules of the SIM-Core by adopting and extending the ROADfactory. In particular, the first and second limitations are addressed in implementing the model execution module (Section 7.4.1), while the third limitation is addressed in implementing the management module (Sec-
CHAPTER 7. SocioPlatform Architecture and Prototype Implementation

7.4.2

The ROAD4WS [12] is a Web service based deployment container for the ROAD-factory. Currently, it supports only the SOAP/Web services technology. The ROAD4WS extends the Apache Axis2 runtime [2, 145] to support the service composite deployment, which is in addition to Axis2’s existing service deployment. Importantly, the changes are introduced as extensions, and no changes are made to the Axis2 code base. This ensures the compatibility with Axis2 code base. The extension is implemented as another layer on top of Axis2, has additional capabilities beyond the behaviour of Axis2-core [112].

In summary, the ROAD4WS is a deployment environment for the ROADfactory and supports SOAP message based interaction among the players/actors. As our implementation of the SIM-Core adheres to the ROADfactory architecture, we can adopt the ROAD4WS without any major modification for the deployment of the SIM-Core. The details are discussed in Section 7.4.3.

Sections 7.4.4 presents a tool to compose DSIMs and PSIMs from social interaction information in SConTo KB in SCIM, while Section 7.4.5 shows how SIM generates and provides interaction events to SCIM.

7.4.1 Implementing the Model Execution Module

The Model Execution Module (MEM) maintains the functional elements of the runtime models. The main classes associated with the MEM are shown in Fig. 7.14. The element types DSIMFunctional@Runtime, PSIMFunctional@Runtime, social role, social relationship, obligation rule, conversation rule, operational parameter and player binding are maintained by the MEM. These are called the functional element types of the runtime models. During runtime, new elements of these types can be added, existing elements can be modified or removed. A runtime model connects multiple role-players and mediates interaction among these players by routing messages from one player to another player. That is, an interaction between players are realised as a message flow between them.

The implementation of the Model Execution Module (MEM) uses JAXB 2.0 [18] and
XML Schema [23]. The DSIM and PSIM models are loaded to the MEM as JAXB 2.0 bindings. JAXB helps the generation of classes and interfaces of runtime models automatically using an XML schema. The XML Schema-based implementation also facilitates change to the model.

JAXB 2.0 supports the translation of an XML representation to Java Classes and vice versa. These processes usually are known as unmarshalling and marshalling [139, 143]. Correspondingly, two classes for Unmarshalling and Marshalling have been implemented. In the process of unmarshalling, an XML document is converted to Java content objects or a tree of objects. Conversely, the marshalling is the reverse process that converts a tree of Java content objects back to an XML document. Using the process of

---

**Figure 7.14:** Class Diagram: Model Execution Module implementation
unmarshalling, the framework creates runtime models based on a DSIM or PSIM descriptor file as shown in Fig. 7.15. The process of marshalling is used to take a snapshot of the current description of a runtime model to be saved as an XML file.

The MEM includes the model parser and binder (MPB) component which is generated by the JAXB 2.0 binding compiler based on the DSIM schema (xsd) and PSIM schema (xsd) (see Appendix E). The MPB is used to generate the DSIMFunctional@Runtime and PSIMFunctional@Runtime models based on a descriptor such as ConvoyDSIM.xml and Car#1PSIM.xml. The descriptor file must conform to the DSIM or PSIM schemas, as appropriate. Therefore, any tool that generates the descriptor file should ensure that the output conforms to the DSIM or PSIM schemas.

We use the Drools rule language [46] to implement social interactions related constraints such as conversations and obligations. From the implementation point of view, there are many advantages of integrating Drools as the rule engine [46] [7]. Apart from the advantages discussed in the Section 7.2.3, Drools allows dynamically injecting facts (Java objects) and rules (Drools rules) into its Stateful Knowledge Session [30]. The Stateful Knowledge Session is long-lived and can be iteratively modified during the runtime (in contrast to its counterpart, the Stateless Knowledge Session). Thus, we have used the Stateful Knowledge Session for executing conversation and obligation rules.

Fig. 7.14 shows the classes associated with the Drools implementation. As shown, any rule base implementation to be integrated with the MEM has to implement the Rela-
tionshipRuleBase interface. Accordingly, the ObligationRuleBase and ConversationRuleBase implement the interface using the Drools plugin for the MEM to enforce the obligations and conversations using Drools rule. A social relationship maintains its associated ObligationRuleBase and ConversationRuleBase during the runtime. New rules can be added and existing rules can be removed from the rule base. Eventually, the rule base is deleted when no longer required, i.e., when the social relationship is removed. When a social interaction, i.e., a message, needs to be evaluated against the specified constraints, i.e., conversations and/or obligations, the corresponding conversation rules and/or obligations rules in their respective rule base are fired via the fireRule() method.

Drools rules are condition-action rules [46]. Therefore, a Drools rule contains two main parts, i.e., the when and then parts, which are also refereed to as the Left Hand Side (LHS) and Right Hand Side (RHS).

- The when part specifies a certain condition to trigger the rule. This condition may be evaluated to true or false based on the properties of facts in the Stateful Knowledge Session [7]. Here, a fact means a Java object, such as a message inserted into the Stateful Knowledge Session.

- The then part specifies what needs to be done if the condition is evaluated to true. The action part is written here, which also might be subject to a further checking of a condition.

Fig. 7.16 shows a sample Drools rule that interprets the obligation o1 in the ConvoyDSIM (presented in the Section 6.2.1.1). The Rule names should be unique in a rule base. Thus, we use the id of the obligation as the name of its corresponding rule, for example, here o1 is used as the name of the rule. The obligation o1 defines that the notifyPositionUpdate interaction (id="i1") occur in every 10 seconds. Thus, the when part of the rule first identifies the target interaction by checking its id as: MessageReceivedEvent(id="i1"). The MessageReceivedEvent is a fact type (Java class) that interprets the arrival of a message to a social relationship. A MessageReceivedEvent fact, i.e., an instance of the MessageReceivedEvent class, is inserted into the rule base when the constraints related to that message needs to be evaluated. The then part of the rule evaluates the constraints. In this part, the isObliged function is invoked, which returns
true if the obligation is satisfied, otherwise false. In the case of a false return value, an alert message is passed to the respective role/player.

```java
// an instance of the obligation rule base
global ObligationRuleBase oRBins;
// time of the last notifyPositionUpdate interaction
global long lastIntrTime = 0;
rules "o1"
  when
    // notifyPositionUpdate interaction
    $msg:MessageReceivedEvent(id="i1")
  then
    long currentTime = System.currentTimeMillis();
    if(!oRBins.isObligated($msg, currentTime, lastIntrTime)){
      $msg.setAlert("Obligation is violated");
    }
    lastIntrTime = currentTime;
end
```

Figure 7.16: A Drools rule for the obligation o1

7.4.2 Implementing the Management Module

The Management Module (MM) supports the adaptation of the runtime models instantiated by the MEM module. The main classes associated with MM are shown in Fig. 7.17. The MM comprises four main classes: Platform Management, OrganiserImp, DSIMCrossAdaptation, PSIMCrossAdaptation, DSIMManagement@Runtime and PSIMManagement@Runtime.

The Platform Management class implements the platform level management operations listed in the Table 7.6. The OrganiserImp class implements the Organiser role interface including state, structure and parameter related management operations, described in the Section 6.2.2.3. The DSIMCrossAdaptation and PSIMCrossAdaptation classes implement cross adaptation related operations. In particular, the DSIMCrossAdaptation implements computeRoleCentricRel while the PSIMCrossAdaptation implements roleAcquisition, roleRelinquishment and updateRelationship. Both of the DSIMManagement@Runtime and PSIMManagement@Runtime uses OrganiserImp to support state management, behaviour and parametric adaptation in their respective functional model. In addition, the DSIMManage-
ment@Runtime and PSIMManagement@Runtime use the DSIMCrossAdaptation and PSIMCrossAdaptation, respectively, to support adaptation across functional models.

7.4.3 The Deployment Layer

The current implementation of the SIM supports the Web service based deployment. We exploit ROAD4WS [121] for the Web service based deployment. It uses the Apache Axis2 [2] as the Web service engine (e.g., Apache Tomcat web server [3]). The issues of transport layer protocols, message processing and security are handled by the Axis2 engine. This ensures that the implementation conforms to the existing Web services standards such as SOAP [19] and WSDL [20].

Once ROAD4WS is installed a new directory called road_composites is created under AXIS2_HOME. In order to deploy a DSIM or PSIM model, the XML descriptor
such as ConvoyDSIM.xml or Car#1PSIM.xml is placed inside this road_composites directory. This can be done before or while the server (Tomcat) is running. Once the model descriptor is deployed, the XML file is automatically picked up by ROAD4WS and uses the MEM and MM to instantiate both the functional (DSIMFunctional@Runtime or PSIMFunctional@Runtime) and management (DSIMManagement@Runtime or PSIMManagement@Runtime) part of the model. ROAD4WS creates the service interfaces (both the functional and management interface) for each and every role based on the functional and management part of the instantiated model. This service interfaces are exposed according to the WSDL 2.0 standard [20]. The service name is created by combining the model and social role name, following the pattern modelname_socialrolename. Each and every message signature of the social interactions is exposed as a WSDL operation. For example, all the interactions directed from the Leading Car social role in the ConvoyDSIM, i.e., positionUpdate, routeUpdate and notifyMechIssue, are exposed as operations of the convoydsim_leadingcar service, as shown in Fig. 7.18. Similarly, all the state, behaviour and parameter related management operations are exposed as WSDL operations. Fig. 7.19 shows the partial management operations exposed as operation of the convoydsim_organizer service. These interfaces allow any WSDL 2.0 compatible client to invoke the operations.

Figure 7.18: LeadingCar social role (in ConvoyDSIM) related interactions are exposed as web service operations
7.4.4 Composing Runtime Social Interaction Models using SIMbuilder

As part of the SocioPlatform, we have implemented a web-based tool, called SIMbuilder, for the software engineer to build DSIM and PSIM models from the social interaction information stored in the SCOnto knowledge base. Fig. 7.20 shows the basic steps for composing such models using SIMbuilder tool (Fig. 7.21).

SIMbuilder retrieves interaction-centric relationships and corresponding social roles from the SCOnto knowledge base in SCIM, and presents to the user (e.g., a software engineer). The tool allows user to create a DSIM by explicitly selecting a set of social roles or social relationships for that model. For the case of social role based model creation, when a social role is selected for a model, all the social roles and relationships (including their interactions, conversations and obligations) those are connected with the selected social role are automatically added to the DSIM. For the case of social relationship based model creation, when a social relationship is selected for the model, just that relationship and its related social roles are added to the model. Once the selection is done, tool automatically composes the DSIM and creates an XML descriptor, which is compliant with the DSIM schema (see Appendix E.1).
7.4.5 Generating and Providing Interaction Events

SIM instantiates runtime social interaction models (i.e., DSIMs and PSIMs) of a collaborative application to mediate social interactions among collaborative actors/users. Users of the collaborative application interact with each other using their application through these runtime models. To allow the reasoning module of the SCIM to infer situations applications use, SIM generates an interaction event for each social interaction and sends it to the reasoning module, as shown in Fig. 7.22.

An interaction event is formatted based on the IntEO ontology presented in Section 5.4.1. Let us consider that during convoy the following car driver, i.e., Car#2, sends the convoyBreak("food") message to the leading car driver at 1:10PM, and after a while, say 1:13PM, Car#2 driver receives the positiveResponse("food") message from the leading
Figure 7.21: SIMbuilder tool for composing DSIMs and PSIMs
car driver, i.e., Car#1. As a result, SIM generates two interaction events $ie_1$ and $ie_2$ as follows:

$$ie_1: \text{InteractionEvent(platform="CooperativeConvoy", fromRole="FollowingCar", toRole="LeadingCar", from="Car#2", to="Car#1", hasType="Proposing", about="Food", atTime=TemporalEntity(start=1:10PM, duration=0ms))}$$

$$ie_2: \text{InteractionEvent(platform="CooperativeConvoy", fromRole="LeadingCar", toRole="FollowingCar", from="Car#1", to="Car#2", hasType="Agreeing", about="Food", atTime=TemporalEntity(start=1:13PM, duration=0ms))}$$

Reasoning module matches these interaction events against the event patterns in situation rule to identify situations. An example is given in Section 8.3.6.1.

7.5 Summary

In this chapter, we have presented the implementation of the SocioPlatform. This platform assists the developers in building socially-aware applications by acquiring, storing, managing and provisioning social context information, providing a runtime envir-
CHAPTER 7. SocioPlatform Architecture and Prototype Implementation

Environment for mediating social interactions, and managing their runtime adaptation. The SocioPlatform comprises two parts: social context information management (SCIM) and social interaction management (SIM).

SCIM implements a number of adapters for acquiring different types of social context information such as social role, social relationship, social interaction and situation from various sources including Facebook, LinkedIn, Twitter and Google Calendar. It provides a knowledge base that exploits SCONto to store such information, allows application developers or domain experts to define inference rules for reasoning about more meaningful and abstract social context information. Furthermore, it provides a query interface for applications to access users’ social context information. In this regard, it also implements a knowledge base using SACOnto, provides a policy interface for users to specify their privacy policies, and enforces users’ policies in accessing their social context information. In summary, SCIM reduces developers’ workload in building socially-aware applications, particularly, data-centric socially-aware applications, by acquiring, managing and provisioning social context information.

SIM provides a runtime environment with adaptation support for developing interaction-centric socially-aware applications. It supports adaptation and mediates social interactions among users of an interaction-centric socially-aware applications. It allows the application developers to externalise inter-user relationships from the applications and model them explicitly. SIM aids the application development by providing a runtime environment to execute the social interaction models, and managing runtime adaptation of these models to cope with the changes in user requirements and environments.
Chapter 8

Case Studies: Social Context-Aware Applications

This chapter describes two case studies involving the development of two prototypical social context-aware applications, in short socially-aware applications, using our novel approach to modelling, reasoning about and managing social context information (SCI), and the associated SocioPlatform, introduced in the preceding four chapters. These case studies revolve around the application scenarios that have been discussed in Chapter 2 and have been used as running examples throughout this thesis. The goal of these case studies is to demonstrate the use and applicability of our proposed framework in developing two different types of socially-aware applications. At the same time, these case studies highlight the benefits and strengths of the framework, and also validate the functionalities of the SocioPlatform.

The first case study presents the design and implementation process of developing a data-centric socially-aware application. This case study uses the phone call interruptions scenario (discussed in Section 2.1.1) as an example for demonstrating how our proposed framework assists software engineers in designing and implementing such an application, called socially-aware phone call (SPCall), by acquiring, modelling, reasoning about, managing and provisioning users’ diverse social context information from various sources. In particular, this case study highlights the functionalities provided by the Social Context In-
formation Management (SCIM) component of the SocioPlatform and demonstrates the use of these functionalities in addressing the challenges and requirements of developing SPCall (as identified in Section 2.1.2).

The second case study presents the design and implementation process of developing an interaction-centric socially-aware application. This case study uses the cooperative convoy application scenario (discussed in Section 2.2.1) as an example for demonstrating how our proposed framework assists software engineers in designing and implementing such an application, called socially-aware telematics (SocioTelematics), by modelling and managing interaction relationships and their runtime adaptation, to cope with the changes in requirements and environments. In particular, this case study highlights the functionalities provided by the Social Interaction Management (SCM) component of the SocioPlatform and demonstrates the use of these functionalities in addressing the challenges and requirements of developing SocioTelematics (as identified in Section 2.2.2).

The principal contributions of this chapter are twofold. First, the challenges and process associated with the design and implementation of two different types of socially-aware applications are illustrated by examples. Second, the principal research contributions of this thesis (i.e., the aforementioned approach to modelling, reasoning about and managing social context information including runtime adaptation support, and the functionalities of the SocioPlatform) are validated.

The discussions of this chapter repeat the various fragments of case studies that have been used throughout the discussion in previous chapters. This chapter, however, aims to present the case studies in a comprehensive manner to demonstrate how the social context information modelling, reasoning and management approach can be systematically applied to developing two different types of socially-aware applications.

The structure of the chapter is as follows. Section 8.1 presents a design and implementation process of developing socially-aware applications using our proposed framework and the associated SocioPlatform. A case study of developing and using a data-centric socially-aware application, namely SPCall, is presented in Section 8.2, while Section 8.3 presents a case study of developing and using an interaction-centric socially-aware application, namely SocioTelematics.
8.1 Design and Implementation Process

Fig. 8.1 shows the design and implementation process for socially-aware applications. Steps 1 to 5 belong to the design process while the step 6 belongs to the implementation process.

In the design process, steps 1 to 4 are applicable for both the data-centric and interaction-centric socially-aware applications, while step 5 is mainly used for the interaction-centric socially-aware applications. During design process, in step 1, social context information of application’s interest is identified through application requirements analysis. The SCONT, i.e., upper social context ontology, is adopted and extended based on the identified social context information. For instance, in the SPCall application design process, the SCONT is adopted and extended by incorporating a interaction-oriented social relationship ontology. In step 3, description logic based reasoning rules related to SCONT are modified (e.g., unnecessary rules are deleted, new rules are added) and IntEO, i.e., interaction event upper ontology, is extended.
(e.g., application specific concepts are added). In step 4, SACOnto, i.e., socially-aware access control ontology, is revisited and modified based on the application requirements. As mentioned above, step 5 is mainly used for the interaction-centric socially-aware applications, to compose both the domain- and player-centric social interaction models (DSIMs and PSIMs) from the connection-oriented social relationships information captured in SCOnto. In this step, software engineers use the SIMbuilder tool to compose these DSIMs and PSIMs. All of these generated artifacts in the design process, i.e., ontologies, rules and models, are used by the SocioPlatform to provide runtime support to the applications.

In the implementation process, i.e., in step 6, software engineers or application developers use APIs provided by the SocioPlatform to implement socially-aware applications. Sections 8.2.4 presents the implementation of the SPCall application, while Section 8.3.5 presents the implementation of the SocioTelematics application.

8.2 Case Study #1: The SPCall Application

The SPCall application is a data-centric socially-aware application, where the data, i.e., social context information such as social relationships and situations, is the basis of the application’s behaviour.

The SPCall application aims to reduce phone call interruptions and considers both the caller and callee perspectives. The application allows caller to know the situations of the intended callee to check whether it is suitable time to call. Accessing the callee’s situations is also subject to the callee’s privacy policies. In this regard, the application allows a person to specify access control polices considering his/her situations at the time of access request and the social relationship with the requester. On the other hand, the application allows a callee to specify her filtering preference on incoming phone calls considering her current situations and the relationships with the caller.

This case study exercises the design and implementation process described in the previous section, to develop the SPCall application. The application exploits social context information provided by the SocioPlatform to support social context-aware
behaviour, i.e., filter incoming phone calls based on social context information. This case study validates the framework’s functionality of modelling and reasoning connection-oriented (people-centric) social relationships, social roles and situations, acquiring and managing such social context information from different online social networks, and preserving privacy in accessing information.

The remaining sections of this case study are organised as follows. Sections 8.2.1 and 8.2.2 validate a number of functionalities of our proposed framework by utilising these functionalities in the SPCall application development to achieve its design features. In this regard, Section 8.2.1 presents design goals and constraints related to the development of SPCall application. The design features to address the design goals and constraints, and how our proposed framework supports these design features are described in Sections 8.2.2. By following the guidelines presented in Section 8.1, Section 8.2.3 presents the design process of the SPCall application, while Section 8.2.4 describes a prototype implementation of the SPCall client for Android. The artifacts, i.e., ontology models and reasoning rules, that are generated in the design process are used by the SocioPlatform. At runtime, SocioPlatform acquires social context information from online social networks and reason about social context information. These are presented in Sections 8.2.5 and 8.2.6 respectively. Finally, Section 8.2.7 illustrates the use of SPCall client together with the social context information provided by the SocioPlatform for two use case scenarios.

8.2.1 Design Goals and Constraint

As discussed in Section 2.1.2, the requirements analysis of the SPCall application identifies two main design goals: (1) minimizing mobile phone interruptions by automatically filtering/handling incoming phone calls based on user preferences that consider user’s social context information such as social relationships and situations, and (2) improving situation awareness by supplying callers with situation information of the intended callee that will lead to a suitable decision on making phone calls. However, supplying a callee’s situation information to a caller raises a serious concern regarding privacy.
8.2.2 Design Features

The above two design goals and a design constraint can be addressed by three design features. Below we describe these design features and how our proposed framework supports these features.

Provisioning social context information – To address the first design goal, the SPCall application needs to be provisioned with its user’s social context information. However, users’ social context information is distributed all over the web, emerging from and fragmented across many different proprietary applications such as Facebook, LinkedIn, and Twitter. Thus, it is a challenge for application developers as they must collect information from different sources and wade through a lot of irrelevant information to obtain the social context information of interest to the targeted functionality. Our proposed framework addresses this challenge by providing an approach to modelling, representing and reasoning about different types of social context information, and platform support for acquiring, storing and accessing such social context information (see, for example, Sections 4.3, 5.2 and 7.1). It has already been recognised that separating the process of acquiring contextual information from actual applications is key to facilitating application development and maintenance [69, 100]. In our framework, the SocioPlatform separates the process of acquiring social context information from the actual applications and provides the design feature of provisioning social context information, which subsequently enhances the application development process.

Allowing a person to access others’ situation information – To address the second design goal, the SPCall application needs to allow its user to access others’ situation information so that the user can decide whether it is an appropriate situation to make a phone call. Therefore, it is not a good design decision to manage a user’s social context information locally in his/her mobile device as technically it is hardly possible for other users to access that information. In our proposed framework, the SocioPlatform supports this design feature (i.e., allows a person to access others’ situation information) as the platform separates the acquisition and management of users’ social context information from the actual applications and
can be deployed as a Web service in a cloud. As a result, the SPCall application can collect others’ situation information from the SocioPlatform. Therefore, a person can access other persons’ situation information using the SPCall application running in a mobile device having internet connectivity.

**Allowing owners to control their information access** – The users’ social context information is inherently sensitive. But for the SPCall application, a user needs to share his/her situation information with other users. This information sharing raises serious concerns regarding the privacy and access control over users’ situation information. Thus, the application should allow its users to retain control over who has access to her situation information under which conditions. In our framework, as discussed in Section 6.1, we propose a way to preserve the information owner’s privacy by allowing the owners to fine-tune the granularity of information access and to specify access control policies. Therefore, our framework addresses this constraint by allowing information owners to specify their privacy preferences, i.e., who can access what information (here user situation) under which conditions. The SocioPlatform implements the proposed privacy preserving technique and enforces users’ privacy preferences when disclosing their situation information to others (see Section 7.2.4). Therefore, the SPCall application can meet this constraint through the SocioPlatform.

8.2.3 Design Process

By following the design process described in Section 8.1, we build artifacts, *i.e.*, ontologies and rules, for the SPCall application. In Section 2.1.2 the requirements of the SPCall application are analysed (step ①), where we have pointed out that the people-centric social relationships (e.g., father-child, mother-child, husband-wife, etc.) and situations (e.g., meeting, seminar, watching game in a stadium, etc.) are the key social context information for the SPCall application. Thus, in step ②, we adopt and extend the upper SConto (see Section 4.2) by incorporating a people-centric social relationship ontology (described in Section 4.3.2.1) and a situation ontology (described in Section 4.3.4). The people-centric social relationship ontology includes family, friend and work related social relationships such as father-son, graduate school friend and
student-supervisor, while the situation ontology includes business and non-business related situations such as meeting, seminar, lecture and watching game in a stadium. Fig. 8.2 shows an excerpt of the extended SCONto for the SPCall application.

In step 3, we further enrich SCONto by incorporating description logic based reasoning rules for deriving social relationships and situations. These reasoning rules have described in Section 5.2 and 5.3.1. A complete list of these reasoning rules are given in Appendix A. To infer situations (e.g., user is watching game in a stadium) from user’s interaction events information on Facebook, we adopt and extend IntEO (see Section 5.4.1) by incorporating a domain-specific interaction event ontology for Facebook (see Section 5.4.3). Fig. 8.3 shows an excerpt of the extended IntEO for the SPCall application.

Using SituLang (see Section 5.4.5), we specify a situation inference rule to detect the ‘watching game in a stadium’ situation (see Section 5.4.7). The SPCall application requires people-centric social relationship and situation based access control, as discussed in Section 2.1.2. Our access control policy model, socially-aware access control ontology (SAConto), considers both the social relationship and situation. Thus, in step 4, we adopt SAConto (Fig. 6.1) without any modification.

In summary, this design process builds the customised and application specific models and rules, i.e., artifacts for the SPCall application. These artifacts include a
**Figure 8.3:** A excerpt of the extended IntEO for SPCall application

SCOnto, an IntEO, a number of situation specification rules and a SACOnto. SocioPlatform uses these artifacts to build a knowledge base for managing users' social context information.

### 8.2.4 Implementing SPCall Client for Android

We have developed the SPCall client for Android mobile devices (step 6). The SPCall application provides functionality both caller and callee. Using the application a caller can obtain the situation of an intended callee to check whether it is a suitable time to call. On the other hand, to reduce interruptions, a callee can specify her preferences on incoming calls such as ring, vibrate, reject, or reject and send situation, based on her current situation and the relationship with the caller.

Fig. 8.4 shows some screenshots of the application. When the application starts first time, the user needs to sign in to her SocioPlatform account (see Fig. 8.4a). After that the user needs to specify her call filtering preferences using the preference editor (Fig. 8.4b). The user can specify privacy preferences regarding access to her social context information using the access control preference editor (Fig. 8.4d). The situation awareness editor (Fig. 8.4c) allows the user to know the situation of an intended callee before making a
Figure 8.4: Screen shots of the $SPCall$ client application for Android
phone call.

*SocioPlatform* (SCIM component) assists the application developer in developing socially-aware phone call applications by collecting and representing the users’ social context information from different sources and providing a set of query APIs for the applications to access that information based on the users’ privacy preferences.

In developing the *SPCall* application we, from the caller perspective, have used the `getSituation(callee)` query API to obtain the situation of an intended callee and then provide that information to the caller. From the callee perspective, to implement the call filtering functionality, we have used the `getAllRelationshipsName(me, inComingCallNum)` and `getSituation(me)` query APIs to obtain the relationships between the caller and callee, and the current situation of the callee. Then, based on the specified filtering preferences in the application, it decides whether to ring, vibrate, reject, or reject and send situation information at a specific granularity. In the case of a “send situation at a specific granularity” decision, the application invokes the `getSituAtGranularity(me, gLevel)` query API to obtain the situation information of the callee at the specified granularity level and then sends it to the caller.

### 8.2.5 Acquiring Social Context Information and Populating Knowledge Base

Let us consider that *SocioPlatform* (SCIM component) acquires Alice’s social context information from different OSNs and stores it into her *SCon* knowledge base. As illustrated in Fig. 8.5a, SCIM acquires social context information from Facebook that Tom is Alice’s graduate school friend (see Table 8.1 for specification in the Turtle syntax [88]), and Jack is Alice’s father (Table 8.2). As illustrated in Fig. 8.5b, SCIM collects social context information from LinkedIn that Bob is Alice’s colleague (Table 8.3) and John is her supervisor (Table 8.4).

### 8.2.6 Reasoning about Social Context Information

After acquiring the information, SCIM classifies Alice’s *SCon* knowledge base using a DL reasoner which infers the following social context information: (i) Tom is Alice’s
Figure 8.5: SocioPlatform acquires social context information from OSNs
### Chapter 8. Case Studies: Social Context-Aware Applications

**Table 8.1:** Tom is Alice’s graduate school friend (Turtle syntax)

<table>
<thead>
<tr>
<th>Object</th>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>:Alice</td>
<td>rdf:type</td>
<td>:Me</td>
</tr>
<tr>
<td>:Tom</td>
<td>rdf:type</td>
<td>:Person</td>
</tr>
<tr>
<td>:Alice-Tom</td>
<td>rdf:type</td>
<td>:GraduateSchoolFriend</td>
</tr>
<tr>
<td>:Alice-Tom</td>
<td>:relates</td>
<td>:Alice</td>
</tr>
<tr>
<td>:Alice-Tom</td>
<td>:relates</td>
<td>:Tom</td>
</tr>
<tr>
<td>:Alice</td>
<td>:relatedTo</td>
<td>:Tom</td>
</tr>
</tbody>
</table>

**Table 8.2:** Jack is Alice’s Father (Turtle syntax)

<table>
<thead>
<tr>
<th>Object</th>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>:Alice</td>
<td>rdf:type</td>
<td>:Me</td>
</tr>
<tr>
<td>:Jack</td>
<td>rdf:type</td>
<td>:Person</td>
</tr>
<tr>
<td>:Alice-Jack</td>
<td>rdf:type</td>
<td>:SocialRelationship</td>
</tr>
<tr>
<td>father</td>
<td>rdf:type</td>
<td>:FamilySocialRole</td>
</tr>
<tr>
<td>:Jack</td>
<td>:plays</td>
<td>:father</td>
</tr>
<tr>
<td>:Alice</td>
<td>:gender</td>
<td>female</td>
</tr>
</tbody>
</table>

**Table 8.3:** Bob is Alice’s Colleague (Turtle syntax)

<table>
<thead>
<tr>
<th>Object</th>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>:Alice</td>
<td>rdf:type</td>
<td>:Me</td>
</tr>
<tr>
<td>:Bob</td>
<td>rdf:type</td>
<td>:Person</td>
</tr>
<tr>
<td>:Alice-Bob</td>
<td>rdf:type</td>
<td>:Colleague</td>
</tr>
<tr>
<td>:Alice-Bob</td>
<td>:relates</td>
<td>:Alice</td>
</tr>
<tr>
<td>:Alice-Bob</td>
<td>:relates</td>
<td>:Bob</td>
</tr>
<tr>
<td>:Alice</td>
<td>:relatedTo</td>
<td>:Bob</td>
</tr>
</tbody>
</table>

**Table 8.4:** John is Alice’s Supervisor (Turtle syntax)

<table>
<thead>
<tr>
<th>Object</th>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>:Alice</td>
<td>rdf:type</td>
<td>:Me</td>
</tr>
<tr>
<td>:John</td>
<td>rdf:type</td>
<td>:Person</td>
</tr>
<tr>
<td>:Alice-John</td>
<td>rdf:type</td>
<td>:Student-Supervisor</td>
</tr>
<tr>
<td>:Alice-John</td>
<td>:relates</td>
<td>:Alice</td>
</tr>
<tr>
<td>:Alice-John</td>
<td>:relates</td>
<td>:Bob</td>
</tr>
<tr>
<td>:Alice</td>
<td>:relatedTo</td>
<td>:John</td>
</tr>
</tbody>
</table>
friend as the graduate school friend is a sub-class of friend (see Table 8.5 and Fig. 8.5a); 
(ii) The relationship between Alice and Jack is Father-Daughter according to the inference 
rule 5.1 and their relationship is a kind of the Family relationship as the Father-Daughter 
relationship is a subclass of the Family relationship (see Table 8.6 and Fig. 8.5a).

**Table 8.5:** Tom is Alice’s friend as the education-based friend is a sub-class of 
friend (Turtle syntax)

<table>
<thead>
<tr>
<th>:EducationBasedFriend</th>
<th>rdf:subClassOf</th>
<th>:Friend</th>
</tr>
</thead>
<tbody>
<tr>
<td>:Alice-Tom</td>
<td>rdf:type</td>
<td>:Friend</td>
</tr>
</tbody>
</table>

**Table 8.6:** The relationship between Jack and Alice is Father-Daughter which is a kind 
of Family relationship (Turtle syntax)

<table>
<thead>
<tr>
<th>:Alice-Jack</th>
<th>rdf:type</th>
<th>:Father-Daughter</th>
</tr>
</thead>
<tbody>
<tr>
<td>:Father-Daughter</td>
<td>rdf:subClassOf</td>
<td>:Family</td>
</tr>
<tr>
<td>:Alice-Jack</td>
<td>rdf:type</td>
<td>:Family</td>
</tr>
</tbody>
</table>

### 8.2.7 SPCall Use Scenarios

Let us consider that Alice, an university graduate student, uses the SPCall application to 
filter her incoming phone calls. She specifies her call filtering preferences in the SPCall as:

i. If my situation is *meeting* or *seminar*, and a call comes from *friend*, the action is to reject;

ii. If my situation is *meeting* or *seminar*, and a call comes from *family*, the action is to reject and forward my status at granularity level 2 (Busy);

iii. If my situation is *meeting* or *seminar*, and a call comes from *colleague*, the action is to reject and forward status at granularity level 3 (*i.e.*, meeting or seminar);

iv. If my situation is *meeting* or *seminar*, and a call comes from the *supervisor*, the action is to vibrate;

v. If my situation is a type of non-business situation, the action is ring.
We further assume that Alice specifies the following Policy#1 and Policy#2 (described in Section 6.1.2) as her privacy preferences regarding access to her status information which are stored in Alice’s SACOnto knowledge base.

Policy#1:

\[
\text{Granted} \equiv \text{Access} \sqcap \\
\exists \text{decision}(\text{Requester} \sqcap \exists \text{asksFor.Situation} \sqcap \\
\exists \text{hasRelationship.Colleague} \sqcap \\
\exists \text{relatedTo.}(\text{Owner} \sqcap \exists \text{hasSituation.Meeting}))
\]

Policy#2:

\[
\text{GrantedInGL5} \equiv \text{Access} \sqcap \\
\exists \text{decision}(\text{Requester} \sqcap \exists \text{asksFor.Situation} \sqcap \\
\exists \text{hasRelationship.Student-Supervisor} \sqcap \\
\exists \text{relatedTo.}(\text{Owner} \sqcap \exists \text{hasSituation.Chatting}))
\]

Based on the above settings we can analyse several scenarios:

**Scene#1:** Once on a Monday 2:30PM, Alice is in a meeting with her supervisor. One of her colleagues, Bob, wants to talk with Alice. Using the SPCall, Bob requests for Alice’s situation. When the request comes to SCIMS, it first deduces Alice’s situation using time instant, as described in Section 5.3.1, and finds that Alice is now supposed to be in a meeting. SCIMS updates Alice’s SACOnto knowledge base with her situation being ‘Meeting’. After that SCIMS checks Alice’s privacy preferences (stored in the SACOnto KB) regarding Bob’s request by following the steps specified in section 6.1.3, and finds that Policy#1 gives permission to send her situation to Bob. Thus, Bob gets the reply about Alice’s situation being ‘Meeting’ and decides not to make a phone call.

**Scene#2:** During the same period of time, Alice receives a phone call from her father
Jack. Alice’s $SPCall$ application sends requests to SCIMS to ask for Alice’s situation and her relationships with the caller. As in scene#1, SCIMS identifies that Alice is in meeting. SCIMS executes the query against Alice’s $SCOnto$ knowledge base to identify her relationships with the caller and finds that the relationship is ‘Father-Daughter’ which is a type of ‘Family’ relationship. Upon receiving the situation and relationship information from SCIMS, Alice’s $SPCall$ application checks with the Alice’s filtering preferences and decides to reject the call and send Alice’s situation at granularity level 2 according to the preference number (ii). The application then requests from SCIMS the situation at granularity level 2 and gets ‘Busy’ as the reply. Finally, the application sends the situation, ‘Busy’, as a message to Jack.

8.3 Case Study #2: The SocioTelematics Application

The SocioTelematics application is an interaction-centric socially-aware application, where agreements and constraints, about social interactions among collaborative actors, are the basis of the application.

The SocioTelematics application aims to allow two or more vehicle drivers to form a cooperative convoy by supporting their social interactions. Such social interactions are based on predefined agreements and constraints that characterise the interaction relationships between the players, such as drivers. In complex and changing environments, such agreements and constraints, and thus interaction relationships are subject to change. Thus, the behaviour of the application needs to be adapted to cope with the changes.

This case study exercises the design and implementation process described in Section 8.1 to develop the SocioTelematics application. The application uses the runtime environment and adaptation management functionalities of SocioPlatform to facilitate interactions and to cope with changes in requirements and environments. This case study validates our proposed approach to modelling runtime social interactions and managing their runtime adaptation.

The remaining sections of this case study are organised as follows. Section 8.3.1 presents design goals and constraints of developing the SocioTelematics application. A
set of design features (provided by our proposed framework) to address these design constraints is presented in Section 8.3.2. Section 8.3.3 presents the design process of developing the SocioTelematics application. Section 8.3.4 describes how the SocioPlatform creates a runtime environment for the SocioTelematics to use by using the artifacts that are generated in design process. Section 8.3.5 describes an implementation of the SocioTelematics client application for Android. Finally, a number of use case scenarios of the SocioTelematics application are described in Section 8.3.6.

8.3.1 Design Goals and Design Constraints

The requirements analysis of the SocioTelematics application (described in Section 2.2.2) identifies three main design goals: (1) the application should support social interactions and enforce users’ agreements in such interactions, (2) the application should provide a coordinated view of the different interactions and allows its users to perform coordination in an automated manner, and (3) the application needs to support adaptation in interaction relationships at runtime. There exists a constraint related to the third design goal – perform adaptation in a safe manner without affecting the interactions.

8.3.2 Design Features

The above three design goals and the design constraint can be addressed by four design features. Below we describe how our proposed framework supports these design features.

Modelling runtime social interactions from a domain perspective – To address the first design goal, it is required to model and represent users’ agreements related to their social interactions. This model should have a runtime representation to enforce constraints and agreements while social interactions are happening. Our proposed framework provides a role based approach to model and represent users’ agreements related to their social interactions, namely the domain-centric social interaction model (DSIM) (see Section 6.2.1.1). Our SocioPlatform provides a runtime environment to instantiate the DSIM and to facilitate social interactions complying with the model specification (see Section 7.3.1).
Modelling runtime social interaction from a player perspective – To address the second design goal, it is required to model and represent social interactions from a player (user) perspective, to allow the player to specify his/her coordination preferences, and to support execution of these preferences. Our proposed framework provides an approach to model social interaction from a player perspective that represents a player’s coordinated view of all his/her social interactions, namely the player-centric social interaction model (PSIM) (see Section 6.2.1.2). Like for a DSIM, the SocioPlatform also provides a runtime environment for instantiating a PSIM (see Section 7.3.1).

Supporting runtime adaptation – To address the third design goal, i.e., allow one to modify runtime social interaction models, it is required to support runtime adaptation in social interaction models. However, social interaction models are interrelated with each other. For instance, one or more player-centric social interaction models might be associated with a domain-centric social interaction model. Thus, a change in a domain-centric social interaction model needs to be propagated to the associated player-centric social interaction model(s). Our framework provides a mechanism for adaptation in a social interaction model (see Section 6.2.2.2) and adaptation propagation across social interaction models (see Section 6.2.3). The SocioPlatform implements this mechanism and provides the adaptation management functionality for applications to use (see Section 7.4.2).

Runtime model states and safe change – To address the design constraint, i.e., performing adaptation without affecting social interactions, it requires the safe change. Our framework defines safe change based on the state of each entity, i.e., social role, social relationship and interaction model as a whole. An entity can be adapted safely only when the entity is in the IDLE state (see Section 6.2.2.4). The SocioPlatform manages the states of the runtime models and their entities, ensures safe adaptation to the runtime models without affecting social interactions.

It has already been recognised that the model based development approach is key to facilitating the application development process and managing runtime adaptation [85], [159], [179]. In our approach, inter-actor relationships in an application are (externalised from the application implementation and) modelled explicitly using DSIMs and PSIMs,
their runtime (execution) environment is generated, and adaptation is managed by the \textit{SocioPlatform}.

### 8.3.3 Design Process

By following the design process described in Section 8.1, we build artifacts, i.e., DSIMs and PSIMs, for the \textit{SocioTelematics} application. Section 8.3.3.1 presents step 2 of the design process that extends SCOnto to model and represent social context information related to this case study. Section 8.3.3.2 presents step 3 of the design process that extends IntEO and specifies a rule to infer a situation from drivers’ social interactions at runtime. As step 5 of the design process, Section 8.3.3.3 shows how to compose domain- and player-centric social interaction models from the extended \textit{SCOnto} using SIMbuilder.

#### 8.3.3.1 Extending SCOnto and Specifying Social Interactions

In Section 2.2.2 the requirements of the \textit{SocioTelematics} application are analysed (step 1), where we have identified that the interaction-oriented social relationships are the basis for the \textit{SocioTelematics} application. To model and represent such social context information of interest, in step 2, we adopt and extend the upper SCOnto (see Section 4.2) by incorporating the cooperative convoy related social role ontology (presented in Section 4.3.1), interaction-oriented social relationship ontology (presented in Section 4.3.3) and driving-in-convoy ontology (described in Section 4.3.4). Fig. 8.6 shows an excerpt of the extended SCOnto.

We specify the social interactions and their related conversations, obligations and operational parameters based on the agreements among the collaborative actors in the cooperative convoy scenario, as discussed in Section 4.3.3. Each social interaction is represented as an instance of the \textit{SCOnto:SocialInteraction}. For example, Table 8.7 presents a social interaction (written in Turtle syntax [64]) which represents the agreement – “upon receiving a route request from a driver of a rented car, a travel guide service should send a route notification in 15 seconds”. This social interaction (\textit{si1}) consists of two basic interactions (\textit{i7} and \textit{i8}), a conversation (\textit{c1}), an obligation (\textit{o3}) and an operational Parameter (\textit{p3}).
Figure 8.6: An excerpt of the extended SCONto for SocioTelematics case study
$i_7$ interaction represents the route request from the rented car social role to the travel guide social role, while the $i_8$ interaction captures the route notification from the travel guide to the rented car. The $c_1$ conversation specifies a sequence between $i_7$ and $i_8$, i.e., $i_7$ leadsTo $i_8$, which states that after the $i_7$ interaction there will eventually be an $i_8$ interaction. The $c_1$ conversation is also subject to obligation $o_3$. This obligation constrains the time duration between $i_7$ and $i_8$, which is maximum 15 seconds, as represented by the $p_3$ operational parameter.

Similarly, we specify all other interactions, conversations, obligations, and operational parameters based on the actors’ agreements in the cooperative convoy (see Appendix B). At the end of this step of the design process, a SCOnto knowledge base is developed that contains the interaction-oriented relationships information among the collaborative actors in a cooperative convoy scenario.

### 8.3.3.2 Extending IntEO and Specifying Situation Reasoning Rules

In step 3 of the design process, to infer situations (e.g., drivers are hungry) we adopt and extend IntEO by incorporating the Time, InteractionType and Topic ontologies for the cooperative convoy scenario, as presented in Section 5.4.2. We reuse the social role and interaction-oriented social relationship ontologies for the cooperative convoy (as shown in Fig. 8.6) in extending IntEO. Fig. 8.7 shows an excerpt of the extended IntEO for the cooperative convoy. This ontology serves as a vocabulary for us to specify situations. Using this IntEO and the human readable situation specification language (presented in Section 5.4.5), we define a hungry situation as follows:

```
when
An atomic interaction event IE1 occurred in "CooperativeConvoy"
which is a message communication, type "Proposing" and
from role "CooperativeConvoy" and discussing about "Food" and
(at time "Breakfast" or at time "Lunch" or at time "Dinner")

A complex interaction event CIE1 observed between IE1 and IE2,
```
Table 8.7: A Social Interaction (si1) that consists of two basic Interactions (i7 and i8), a Conversation (c1), a Obligation (o3) and an Operational Parameter (p3). The social interaction can be read as – upon receiving a route request from a driver of a rented car, a travel guide service should send a route notification in 15 seconds

<table>
<thead>
<tr>
<th>Class</th>
<th>Instance</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>si1</td>
<td>rdf:type</td>
<td>:SocialInteraction</td>
</tr>
<tr>
<td>si1</td>
<td>:hasIntrac</td>
<td>i7</td>
</tr>
<tr>
<td>si1</td>
<td>:hasIntrac</td>
<td>i8</td>
</tr>
<tr>
<td>si1</td>
<td>:hasConv</td>
<td>:c1</td>
</tr>
<tr>
<td>si1</td>
<td>:hasOblg</td>
<td>:o3</td>
</tr>
<tr>
<td>si1</td>
<td>:hasOpParam</td>
<td>:p3</td>
</tr>
<tr>
<td>i7</td>
<td>rdf:type</td>
<td>:Interaction</td>
</tr>
<tr>
<td>i7</td>
<td>:msgSignature</td>
<td>&quot;routeRequest&quot;</td>
</tr>
<tr>
<td>i7</td>
<td>:fromRole</td>
<td>:RentedCar</td>
</tr>
<tr>
<td>i7</td>
<td>:toRole</td>
<td>:TravelGuide</td>
</tr>
<tr>
<td>i8</td>
<td>rdf:type</td>
<td>:Interaction</td>
</tr>
<tr>
<td>i8</td>
<td>:msgSignature</td>
<td>&quot;routeNotification&quot;</td>
</tr>
<tr>
<td>i8</td>
<td>:fromRole</td>
<td>:TravelGuide</td>
</tr>
<tr>
<td>i8</td>
<td>:toRole</td>
<td>:RentedCar</td>
</tr>
<tr>
<td>p3</td>
<td>rdf:type</td>
<td>:OperationalParameter</td>
</tr>
<tr>
<td>p3</td>
<td>:paramName</td>
<td>&quot;maxTimeDelay&quot;</td>
</tr>
<tr>
<td>p3</td>
<td>:value</td>
<td>15</td>
</tr>
<tr>
<td>p3</td>
<td>:timeUnit</td>
<td>SEC</td>
</tr>
<tr>
<td>o3</td>
<td>rdf:type</td>
<td>:Obligation</td>
</tr>
<tr>
<td>o3</td>
<td>:timer</td>
<td>:duration</td>
</tr>
<tr>
<td>o3</td>
<td>:relatedOp</td>
<td>:p3</td>
</tr>
<tr>
<td>c1</td>
<td>rdf:type</td>
<td>:Conversation</td>
</tr>
<tr>
<td>c1</td>
<td>:intrac1</td>
<td>:i7</td>
</tr>
<tr>
<td>c1</td>
<td>:intrac2</td>
<td>:i8</td>
</tr>
<tr>
<td>c1</td>
<td>:hasSequence</td>
<td>:leadsTo</td>
</tr>
<tr>
<td>c1</td>
<td>:relatedOblg</td>
<td>:o3</td>
</tr>
</tbody>
</table>
where interaction event IE2 is a message communication, type "Agreeing" and from role "CooperativeConvoy" and discussing about "Food", has occurred "after" IE1

then Notify "Hungry"

Figure 8.7: An excerpt of the extended IntEO for the cooperative convoy

In summary, in this step, a cooperative convoy application specific IntEO is build and a rule to recognise the hungry situation is specified.

8.3.3.3 Composing Domain- and Player-centric Social Interaction Models

In step 5 of the design process, we use SCONto that has developed in Section 8.3.3.1 and the SIMbuilder tool, to compose both the domain-centric social interaction models (DSIMs) and player-centric social interaction models (PSIMs). In Section 7.4.4 we have described steps for composing such DSIMs and PSIMs using the SIMbuilder tool. We use social roles as the basis to compose the DSIMs. We compose three DSIMs: ConvoyDSIM, BudgetDSIM and AVISDSIM. The ConvoyDSIM consists of the LeadingCar and FollowingCar
social roles. Thus, all the interactions, conversations, obligations and operational parameters related to these social roles, as captured in the SCOnto, are automatically added to the ConvoyDSIM by the SIMbuilder. See Appendix C.1.1 for the complete specification of the ConvoyDSIM. Similarly, we compose the BudgetDSIM and AVISDSIM. The BudgetDSIM consists of the RentedCar, RentalCompany, TravelGuide and RoadsideAssistance social roles, while the AVISDSIM consists of the RentedCar, RentalCompany, TrafficManagement and RoadsideAssistance social roles. The complete specifications of the BudgetDSIM and AVISDSIM are presented in Appendices C.1.2 and C.1.3, respectively.

We use the SIMbuilder to further compose PSIMs from the Car#1 and Car#2 perspectives, namely Car#1PSIM and Car#2PSIM. SIMbuilder takes the ConvoyDSIM, BudgetDSIM and AVISDSIM as inputs and automatically generates these two PSIMs. Appendices C.2.1 and C.2.2 present the complete specification of Car#1PSIM and Car#2PSIM, respectively. At the end of this step 5, five social interaction models – ConvoyDSIM, BudgetDSIM, AVISDSIM, Car#1PSIM, and Car#2PSIM – have been built, which we will use to generate the runtime execution environment for SocioTelematics.

### 8.3.4 Instantiating Models and Creating Runtime Environment

Section 8.3.3 has described steps 1 to 5 of the design process that build artifacts such as SCOnto, IntEO, a situation inference rule, ConvoyDSIM, BudgetDSIM, AVISDSIM, Car#1PSIM and Car#2PSIM for developing the SocioTelematics application. The SocioPlatform uses these artifacts, i.e., models and rules, to create a runtime environment for the application to use. In particular, SIM instantiates DSIMs and PSIMs, and creates a runtime environment, i.e., DSIMFunctional@Runtime, DSIMManagement@Runtime, PSIMFunctional@Runtime, and PSIMManagement@Runtime, for mediating social interactions (see Section 7.3.1). SIM also generates a Web service interface for the SocioTelematics application to invoke. On the other hand, SCIM uses SCOnto, IntEO and the situation inference rule to create a knowledge base and semantic event processing environment for inferring users’ situations from interaction events. SCIM exploits the rule translation script (RuleTrans.dsl) (provided in Appendix D) to convert the situation inference rule, specified in Section 8.3.3.2, to the Drools executable format.
8.3.5 Implementing *SocioTelematics* Client for Android

We have implemented the *SocioTelematics* application for Android mobile devices and Tablets (step 6). To support map, we use Google Map APIs for Android. To interact with the *SocioPlatform* and process SOAP messages, we use ksoap2 [11] – a lightweight SOAP library for the Android platform. The application executes a number of tasks running as background services:

- **Location sharing service** – Listen to the location update from specified location provider (e.g., GPS, Network) and send the position to the other cars as `positionUpdate` interaction messages.

- **Message fetching service** – Pull the received messages from the queue in a proxy running on the server.

- **Message processing service** – Extract information from the fetched message, evaluate and fire coordination rules, and update the interaction history accordingly.

- **Map viewer service** – Periodically update the cars’ (both leading and following) positions on the map viewer so that drivers can see each other’s positions.

- **Text to speech conversion service** – To reduce driver distraction, this service provides voice notification (message name and contents) when a message comes.

The application has four fundamental panels such as *settings*, *convoy*, *interaction* and *adaptation* (see Fig. 8.8). When the application starts, the driver needs to configure the application using the *settings panel* (see Fig. 8.8a) by providing a user ID (to recognise the car) and password. Using the user’s credential, the application invokes the `getRoleList` organiser role function at the server side, fetches all social roles associated to its user and populates those roles in the main screen. Also based on each social role in the role list, the application invokes the `getInteractionList` function and accumulates associated interaction definitions for each social role. This means that the application is not tightly coupled with the social roles played by the car in a cooperative convoy and also the company from which the car has been rented. Thus, it serves the purpose of multi-modal use, *i.e.*, without any internal logic change this application can be used in a car from any...
Figure 8.8: Some screenshots of the SocioTelematics application for Android
rental company as well as by a driver playing either the leading car or following car social role.

During driving, for the driver to perform the additional tasabirks in the cooperative convoy (e.g., forwarding information) as agreed may cause distraction. To facilitate such collaboration but with less distraction, the application allows the driver to specify coordination rules for, say, forwarding information. In the settings panel, a driver can select an incoming interaction message and its corresponding outgoing message(s) as his/her coordination preferences. For example, the following car driver specifies a coordination preference as: “when a road block notification is received from the roadside assistance service (as the roadBlockNotification interaction), forward it to the leading car (as the notifyRoadBlock interaction)”. Fig. 8.9 shows this coordination rule specified in the SocioTelematics application. Thus, when the application receives a message that matches with an incoming message in the coordination preferences, the application (as a coordinator player) sends the corresponding outgoing message(s) immediately on behalf of the driver. Also, the application provides voice notification (message name and contents) when a message comes.

Fig. 8.8b shows the convoy panel of the application. The map of the screen shows the cars’ positions in the cooperative convoy. The application uses GPS to show its car position on the map and notifies that position to others (as the notifyPosition interaction) using the location sharing service. The application gets the other car’s position once it receives a notifyPosition message from the other car. Also it shows a warning message as pop-up, for example, if the distance between the two cars violates their agreed distance.
CHAPTER 8. CASE STUDIES: SOCIAL CONTEXT-AWARE APPLICATIONS

The interaction panel (see Fig. 8.8d) of the application allows a driver to perform interactions and displays: the role list – associated with the driver of the car, the interactions list – list of interactions corresponding to the selected role in the role list, and the interaction log – interactions so far occurred. Section 8.3.6.1 presents some interaction examples.

The adaptation panel (see Fig. 8.8d) allows the leading car driver to perform adaptation on the ConvoyDSIM. The current version of the application provides an interface for the organiser player, e.g., the driver of the leading car, to perform/trigger three structural adaptation: create a new position (i.e., social role) for a new car to join in the convoy, join a car to the convoy and remove a car from the convoy. As required, options to trigger other types of structural adaptation can easily be incorporated to the interface. A number of parametric adaptation can be performed through the settings panel. The current version provides options to set the position update frequency and maximum desired distance. Here, it is worth noting that using the application interface the organiser player only triggers or requests adaptation, while the actual adaptation is performed on the server side. Section 8.3.6.2 illustrates a number of adaptation scenarios.

8.3.6 SocioTelematics Use Scenarios

This section describes a number of use cases for the SocioTelematics application. Section 8.3.6.1 presents use cases where drivers in the cooperative convoy use the application to perform social interactions. Section 8.3.6.2 describes use cases where drivers use the SocioTelematics application to perform runtime adaptation.

8.3.6.1 Performing Social Interactions

At runtime, the drivers of the cooperative convoy use the SocioTelematics application to perform social interactions adhering with their agreements and constraints. Through these interactions, drivers perform tasks and achieve their goals.

For instance, during convoy, if any driver needs to stop for some reason, e.g., food, fuel, etc., he/she can propose it to other drivers using the convoyBreak(reason) social interaction. Other drivers can reply with a positiveResponse or negativeResponse social inter-
action. These interactions are mediated by the SIM of the SocioPlatform. SIM generates an interaction event for each social interaction and insert that interaction event to the reasoning module of the SCIM. The complex event processing engine of the reasoning module incrementally matches the inserted interaction events with the event patterns in the situation inference rule. When a rule is matched, the situation is identified and inserted to the SCONto knowledge base.

Assume that the following car driver sends the convoyBreak("food") message to the leading car driver and after a while receives the positiveResponse("food") message. As a result, SIM generates two interaction events ie1 and ie2 as follows:

\[
\text{ie1: InteractionEvent(platform="CooperativeConvoy", fromRole="FollowingCar", hasType="Proposing", about="Food", atTime=TemporalEntity(start=1:10PM, duration=0ms))}
\]

\[
\text{ie2: InteractionEvent(platform="CooperativeConvoy", toRole="LeadingCar", hasType="Agreeing", about="Food", atTime=TemporalEntity(start=1:13PM, duration=0ms))}
\]

Also assume that the inference rule to identify the hungry situation, as described in Section 8.3.3.2, is running in the reasoning module. Therefore, the hungry situation is identified by the above two interaction events, and is stored into the SCONto knowledge base. As a result of this situation, the SocioTelematics application of the leading car sends a nearby restaurant address request as a addressRequest("restaurant") interaction to the travel guide service. Once it receives the restaurant addresses as a responseAddress interaction, the convoy panel highlights those addresses on the map. The driver can choose any of the restaurant and drive to there for food.

8.3.6.2 Performing Runtime Adaptation

In this section, we consider a number of use cases that require adaptation in the ConvoyDSIM and the associated PSIMs. The organiser player of the ConvoyDSIM, here the driver of the leading car, triggers the adaptation by using the SocioTelematics application, while the adaptation request is executed and managed by the SocioPlatform. Fig. [8.10] il-
Illustrates the modifications of the runtime DSIM and PSIMs as adaptation is carried out.

- To add a new car, say Car#3, as a following car in the convoy, the leading car driver (the ConvoyDSIM organiser player) presses the respective button in the adaptation panel (Fig. 8.8d) of the SocioTelematics application that invokes the following management methods (see Fig. 8.10(a)): (1) `addSocialRole("FC2")` – to add another following car role, namely FC2, to the ConvoyDSIM, (2) `addSRelationship("LC", "FC2", relName)` – to add a relationship between the LC and FC2 social roles, (3) `addInteraction, addConversation, addObligation` and `addOperationalParam` – to add functional and non-functional information to the relationship as required, and finally (4) `bindRolePlayer` – to bind the third vehicle to the FC2 role. Because the `bindRolePlayer` is invoked, the ConvoyDSIM (as part of the cross adaptation process) computes the role-centric relationship of the FC2 (say Rr5) using the Algorithm 1 (see Section 6.2.3) and invokes the `triggerRoleAcquisition` method in Car#3PSIM with

![Figure 8.10: An illustration of the runtime adaptation in the ConvoyDSIM and associated PSIMs to cope with changes in user requirements and environments](image-url)
Chapter 8. Case Studies: Social Context-Aware Applications

FC2 and Rr5 as parameters. As a result, the Car#3PSIM updates its structure by adding the FC2 role and the Rr5 role-centric relationship between the FC2 and Coordinator roles (see Fig. 8.10). As a result of these sequence of adaptations, Car#3PSIM is linked with the ConvoyDSIM, and thus using the SocioTelematics application driver of Car#3 can perform interactions with other car drivers of the convoy through the Car#3PSIM.

- If the leading car (Car#1) breaks-down, the ConvoyDSIM organiser player (the leading car driver) presses the respective button in the adaptation panel of the SocioTelematics application that invokes the unbindRolePlayer method to unbind Car#1PSIM from the LC. As a consequence, the ConvoyDSIM (as part of the cross adaptation process) invokes the triggerRoleRelinquishment("LC") method in Car#1PSIM. As a result, the Car#1PSIM updates its structure by deleting the LC role and the relationship between the LC and Coordinator roles (see Fig. 8.10). Furthermore, to assign a following car (say Car#2) to play the leading car role, the leading car driver presses the respective button that invokes unbindRolePlayer("FC", urlCar#2PSIM) followed by bindRolePlayer("LC",urlCar#2PSIM) to first unbind Car#2PSIM from the following car role and then bind it to the leading car role. As a result, the ConvoyDSIM (as part of the cross adaptation process) invokes the triggerRoleRelinquishment and triggerRoleAcquisition methods respectively which ultimately updates the Car#2PSIM by removing the FC role and its associated relationship (see Fig. 8.10) followed by adding the LC role and its associated relationship (see Fig. 8.10). As a result of these sequence of adaptations, the Car#1PSIM has detached from the ConvoyDSIM, and thus using the SocioTelematics application, the driver of Car#1 will no longer be able to perform interactions with other car drivers.

- The heavy raining situation can be detected by Car#2’s (LC) wet-sensor and an adaptation rule (defined in section 6.2.2.3) embedded in the leading car’s SocioTelematics application can trigger adaptation automatically or the driver of the leading car can trigger adaptation manually by pressing a button in the application. This adaptation request invokes the updateOperationalParam(r1,maxDesiredDistance,600m) method in ConvoyDSIM to change the
maxDesiredDistance parameter value from 1km to 600m. Both the Car#2PSIM and Car#3PSIM are related to the R4 relationship. Thus, the ConvoyDSIM (as part of the cross adaptation process) invokes the triggerUpdateRelationship method in Car#2PSIM and Car#3PSIM to update the maxDesiredDistance value in both of these PSIMs (see Fig. 8.10).

8.4 Summary

In this chapter, we have presented two case studies of developing and using social context-aware applications. In this regard, we have presented a design and implementation process of developing such applications. We have also described a number of use case scenarios to illustrate the usefulness of these applications. The first case study is a data-centric socially-aware phone call application, namely SPCall, while the second case study is an interaction-centric socially-aware application, namely SocioTelematics.

On one hand, these two case studies have highlighted various challenges in terms of design goals and constraints, for developing two different types of socially-aware applications. The design features provided by our framework and the SocioPlatform to address those challenges are also illustrated.

On the other hand, these case studies have exercised our approach to designing and implementing socially-aware applications. Subsequently, they have validated the principal contributions of this thesis, i.e., an approach to modelling, reasoning about and managing social context information including runtime adaptation support.
Chapter 9

Experimental Evaluation

In this chapter, we evaluate the SocioPlatform presented in Chapter 7 and the two socially-aware applications (SPCall and SocioTelematics) developed in Chapter 8.

As discussed in Chapter 7, the SocioPlatform provides high-level support for developing socially-aware applications by acquiring, reasoning and provisioning different types of social context information and managing runtime adaptation. It comprises two main components, social context information management (SCIM) and social interaction management (SIM), which provide different functionalities or services to the applications. The SCIM provides social context information acquisition, reasoning and query processing functionalities, while the SIM provides a runtime environment for mediating social interactions (i.e., checking interaction constraints and routing interaction messages) based on the runtime social interaction models, and supports their runtime adaptation.

In order to evaluate the SocioPlatform, we setup an experimental environment (Section 9.1). We evaluate the performance of each of the aforementioned core functionalities provided by the SCIM and SIM. We evaluate the information acquisition, reasoning and query processing performance of SCIM using real data from Facebook, LinkedIn, Twitter and Google Calendar (Section 9.2). We quantify the adaptation overhead of SIM, and evaluate the performance and resource consumption of SIM related to mediating social interactions (Section 9.3).

Furthermore, to demonstrate the real-world applicability and feasibility of the SPCall
and SocioTelematics applications that have been developed using our framework and the SocioPlatform, we evaluate the applications’ performance and resource consumption using real-life experiments. Section 9.4 evaluates the SocioTelematics application, while Section 9.5 presents the evaluation of the SPCall application.

9.1 Experimental Setup

The SocioPlatform is deployed and evaluated on a machine with a Core 2 Duo E8400 3 GHz processor, 3 GB RAM, Windows XP professional edition SP3. We used Java 1.6, Drools 2 [8], Tomcat 7.0.21 [3] and Axis2 1.6.1 [2] in this experiment.

To evaluate the SPCall and SocioTelematics applications in real-life experiment, it is required to deploy the SocioPlatform on a machine with a real IP-address so that the applications running on a mobile device can communicate with the SocioPlatform over the Internet. Thus, for the experiments to evaluating applications (Sections 9.5 and 9.4), we have deployed the SocioPlatform on a standard windows based Apache Tomcat Web server in the Amazon Elastic Compute Cloud (Amazon EC2)[1].

9.2 Evaluating SCIM

SCIM is a knowledge management component of the SocioPlatform, providing information acquisition, reasoning and query processing functionalities. The reasoning functionality includes both the description logic based social context information (SCI) reasoning and real time complex interaction event stream processing based situation reasoning. Sections 9.2.1, 9.2.2 and 9.2.3 present the performance evaluation of SCI acquisition, description logic based SCI reasoning and query processing over SCI, respectively.

The purpose of these experiment was to quantify the performance of SCIM using data of social media users of having different number of friends in their accounts in online social networks – ranging from the maximum allowed count (i.e., 5000 connections for Facebook) to the average number (around 130 for Facebook) and further to small counts (less than 40 connections). In these experiments, we choose eight social media

users who cover these desired cases and have accounts in Facebook, LinkedIn, Twitter and Google Calendar. Note that the users gave us permission to collect their social data. The SCONto and its related reasoning rules, as discussed in Section 7.2.1, have been used in these experiments.

To evaluate the performance of real time interaction event stream processing based situation reasoning (Section 9.2.4), the data set, i.e., situation rules and interaction event streams, should fulfill some criteria. For instance, a specific number of interaction events in an interaction event stream should be matched with a specific number of situation reasoning rules. It is very difficult to collect interaction event streams from social media with such criteria. Thus, to evaluate the performance of complex event processing based situation reasoning, we have used simulated data that comply with the desired criteria but actually represent interaction events in social media.

9.2.1 Performance Evaluation for Information Acquisition

To assess the performance of the information acquisition functionality of SCIM, we evaluate the time cost of acquiring information using two metrics: (i) time required to fetch raw data from a source – to quantify the network overhead; (ii) Preprocessing and storing time – time required to extract the information of interest from fetched data, transform it to the suitable format, and insert it into the SCONto knowledge base (KB) (uses OWL-DL data model) – to quantify the performance of inserting information into the SCONto KB of SCIMS.

For each Facebook account of the eight experimental subjects, we ran the experiment 50 times. Fig. 9.1a and Fig. 9.1b show the result where error bars depict standard deviation. For the highest number of friends (i.e., 5000), the average time to fetch information from Facebook was 12 sec (see Fig. 9.1a) and the average time for preprocessing and inserting information into knowledge base was around 700 ms (see Fig. 9.1b), which is an acceptable result, since this information acquisition is usually performed off-line and might not be required to be updated very frequently. In general, the results in Fig. 9.1a and Fig. 9.1b show that time requires to acquire social context information increases with increasing size of information.
Chapter 9. Experimental Evaluation

Figure 9.1: Time taken by the Information acquisition module in SCIMS to acquire social context information from Facebook

(a) Fetching information from Facebook

(b) Inserting information into Knowledge Base
### Table 9.1: Social context information collected from users’ Facebook accounts

<table>
<thead>
<tr>
<th>Subjects</th>
<th>Connections</th>
<th>People-centric</th>
<th>Object-centric</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Family</td>
<td>Friend</td>
<td>Work</td>
</tr>
<tr>
<td>User1</td>
<td>35</td>
<td>5(3)</td>
<td>37(5)</td>
<td>2(1)</td>
</tr>
<tr>
<td>User2</td>
<td>171</td>
<td>0(0)</td>
<td>152(3)</td>
<td>0(0)</td>
</tr>
<tr>
<td>User3</td>
<td>297</td>
<td>12(3)</td>
<td>202(3)</td>
<td>32(2)</td>
</tr>
<tr>
<td>User4</td>
<td>338</td>
<td>1(1)</td>
<td>326(1)</td>
<td>12(1)</td>
</tr>
<tr>
<td>User5</td>
<td>387</td>
<td>5(4)</td>
<td>143(2)</td>
<td>18(1)</td>
</tr>
<tr>
<td>User6</td>
<td>580</td>
<td>1(1)</td>
<td>502(3)</td>
<td>54(1)</td>
</tr>
<tr>
<td>User7</td>
<td>813</td>
<td>3(2)</td>
<td>809(4)</td>
<td>33(1)</td>
</tr>
<tr>
<td>User8</td>
<td>5000</td>
<td>0(0)</td>
<td>3126(2)</td>
<td>21(1)</td>
</tr>
</tbody>
</table>

The number reported in parentheses is relationship types in each category.
Table 9.1 reports the summary of the social context information that we have collected from these users’ Facebook accounts, including the users’ total connections (a connection is a link between two users), the number of connections in different categories of relationships and the number of relationship types in each category (reported in parentheses). It is worth noting that a connection could be a member of multiple relationship types. For example, a connection between two users could be a college friend, graduate school friend and also they might live in same city. The connections of User3 are categorized into 18 types of relationships while the connections are categorized into 4 types for User4. Thus, the information fetching (Fig. 9.1a) and inserting (Fig. 9.1b) time for User3 have taken longer than User4 even though User3’s number of connections is less than User4’s number of connections.

Similarly, using SCIM, we have also collected these users’ other type of social context information such as research-topic based relationship, following-follower relationship, and situations, from LinkedIn, Twitter and Google Calendar. At the end of this information acquisition experiment, SCIM has built eight knowledge bases for the eight users. Table 9.2 reports the details of these knowledge bases, including the number of inserted axioms\(^2\) and the size (in kilobyte) of each knowledge base. In the table, we can see that the number of axioms in User3’s KB is more than the number of axioms in User4’s KB as User3 has more classified relationships than User4.

### 9.2.2 Performance Evaluation for Description Logic based Reasoning

To measure the SCIM’s performance on classifying and reasoning, we have used the eight knowledge bases that have been built in the above experiments. Fig. 9.2 shows the results of knowledge base initialisation and classification time using five different reasoners. In general, the initialisation and classification time increases with increasing sizes of the knowledge base. The results show that TrOWL performs very well for a small sized KB, but as the KB size increases the computation cost increases dramatically. However, Fact++ outperforms consistently from small to large sized KBs. Table 9.2 reports the number of inferred axioms newly created after classifying each KB and the overall size of the KB.

\(^2\)OWL uses statements to represent knowledge, which are called axioms
Table 9.2: Knowledge bases (KBs) have been built for experimental subjects before and after classification and reasoning.

<table>
<thead>
<tr>
<th>KB No.</th>
<th>Experimental Subjects</th>
<th>Asserted Axioms count (after inserting SCI into KB)</th>
<th>KB size in kilobyte</th>
<th>Inferred axioms count</th>
<th>Overall KB size in kilobyte</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>User1</td>
<td>657</td>
<td>528</td>
<td>90</td>
<td>116</td>
</tr>
<tr>
<td>2</td>
<td>User2</td>
<td>1513</td>
<td>1401</td>
<td>234</td>
<td>267</td>
</tr>
<tr>
<td>3</td>
<td>User3</td>
<td>2167</td>
<td>2576</td>
<td>358</td>
<td>430</td>
</tr>
<tr>
<td>4</td>
<td>User4</td>
<td>1822</td>
<td>2557</td>
<td>287</td>
<td>400</td>
</tr>
<tr>
<td>5</td>
<td>User5</td>
<td>2489</td>
<td>2907</td>
<td>410</td>
<td>493</td>
</tr>
<tr>
<td>6</td>
<td>User6</td>
<td>3389</td>
<td>4756</td>
<td>616</td>
<td>772</td>
</tr>
<tr>
<td>7</td>
<td>User7</td>
<td>4583</td>
<td>5422</td>
<td>829</td>
<td>1011</td>
</tr>
<tr>
<td>8</td>
<td>User8</td>
<td>24245</td>
<td>32381</td>
<td>4499</td>
<td>5318</td>
</tr>
</tbody>
</table>
Figure 9.2: Reasoning time for various sizes of KBs using five different reasoners (i.e., HermiT, Pellet, RacerPro, TrOWL and Fact++)

9.2.3 Performance Evaluation for Query Processing

To measure the SCIM’s performance on query processing, we measure the time to answer a set of queries – `isAFamilyOfB`, `getAllRelationshipsName` and `getSituAtGranularity`. The eight knowledge bases (see Table 9.2) that have been built in previous experiments using users’ social context information acquired from Facebook, LinkedIn, Twitter and Google calendar, are used for performance evaluation.

Figure 9.3: SCIM’s performance on query processing (QP) – time taken to answer a set of queries (`isAFamilyOfB`, `getAllRelationshipsName` and `getSituAtGranularity`) for various sizes of KBs using five different reasoners (i.e., HermiT, Pellet, RacerPro, TrOWL and Fact++)
Fig. 9.3 shows the results of query execution time for these KBs using the same five different reasoners. The results show that HermiT and Pellet perform well and take almost the same amount of time to process queries, while TrOWL outperforms all other reasoners (including HermiT and Pellet), from small to large sized KBs.

9.2.4 Performance Evaluation for Interaction Event Stream based Situation Reasoning

To evaluate the performance of the Interaction Event Stream based Situation Reasoning, we have conducted a series of experiments. The goal of these experiments was to quantify the platform performance in processing interaction event streams (IESs), executing situation specification rule (SSR) sets and detecting situations. In these experiments, we have used the TrOWL reasoner in semantic matching, as the experimental results presented in Section 9.2.3 showed that the TrOWL reasoner outperforms, i.e., takes comparatively less time to perform matching, over the RacerPro, Fact++, HermiT and Pellet reasoners.

We have measured the situation reasoning time with increasing sizes of SSR sets where each rule in the rule sets consists of six simple interaction event patterns (IEPs) and three to five complex interaction event patterns (CIEPs). Each IEP uses two to eight ontological properties. We have created three interaction event streams (IESs) where each IES comprised 1000 interaction events (IEs). These three IESs match all rules in the SSR sets exactly one, fifty and hundred times respectively. This means that for a SSR set with 500 rules, an IES that matches each rule 50 times will detect 25000 situations (i.e., 500x50). For each of the three IESs, we execute each set of SSRs 1000 times. The results in Fig. 9.4 show that for a given number (1, 50 or 100) in which each rule is fired, the total time to fire a set of SSRs increases approximate linearly with the size of the SSR set. For a particular size of SSR set, the time taken to fire the rule set increases as the number times each rule is fired increases. Moreover, the results also show that for a particular number of detected situations, an SSR set of smaller size takes less time compared to an SSR set of larger size. For instance, as illustrated in Fig. 9.4, both the case where each of 500 SSRs fire 100 times and the case where each of 1000 SSRs fire 50 times, detect 50,000 situations. However, the former case takes less time than the latter, in detecting
the same number of situations.

![Performance evaluation results for a fixed size (1000) of interaction event stream (IES) with increasing situation specification rule (SSR) set sizes and event firing occurrences](image)

Figure 9.4: Performance evaluation results for a fixed size (1000) of interaction event stream (IES) with increasing situation specification rule (SSR) set sizes and event firing occurrences

We have also measured the situation reasoning time for 1000 SSRs with increasing sizes of IESs (or the volumes of the interaction events). We have created a set of IESs where each IES matches each rule of the 1000 SSRs exactly once, i.e., regardless the size of the IES, each experimental IES will detect 1000 situations. The error bars in Fig. 9.5 show that for 1000 SSRs and 1000 detected situations, the execution time does not vary much with the sizes of the IESs and lies between 3.94 and 3.97 seconds (on average). That is, for a fixed size of SSR set and a fixed number of detected situations, the size of the inserted IES does not have much impact on the platform performance in terms of computation time.

### 9.3 Evaluating SIM

SIM is a mediated social interactions management component of the SocioPlatform. It provides a runtime environment for the interaction-centric socially-aware applications to support mediated social interactions based on runtime social interaction models (i.e., DSIMs

230
and PSIMs). It also supports runtime adaptation of these social interaction models. The experimental evaluation of SIM had two main goals: (1) to measure the overhead of adaptation operations on runtime social interaction models (Section 9.3.1), and (2) to measure the resource consumption and system performance of the SIM, related to mediating social interactions, under different workloads (Section 9.3.2).

### 9.3.1 Quantifying Adaptation Overhead

To quantify the adaptation overhead, we deployed 100 domain-centric social interaction models (DSIMs) where each DSIM consists of 10 social roles connected in a ring topology using 10 social relationships. Each relationship is comprised of 6 interactions, 6 conversations and 6 obligations. Using SIMbuilder, we created 10 PSIMs for 10 players where each player plays a role in each of the 100 DSIMs. Thus, each PSIM contained 100 social roles and 100 role-centric relationships. We executed each of the structural and parametric adaptation operations 1000 times over the 100 DSIMs. We measured the time from the moment the adaptation was requested, to the moment Axis2 updated the services. The box plots in Fig. 9.6 and Fig. 9.7 show the summary of the results where the horizontal line inside each of the boxes represents the median. The results show that
the deletion operations (e.g., \text{delSocialRole}, \text{delSRelationship}, and \text{unbindRP}) take less time compared to the addition operations (i.e., \text{addSocialRole}, \text{addSRelationship}, and \text{bindRP}).

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig9.6a.png}
\caption{Structural adaptation}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig9.6b.png}
\caption{Parametric adaptation}
\end{figure}

Fig. 9.6a and Fig. 9.6b present the time required to perform different structural and parametric adaptation in a social interaction model at runtime. The results show that the structural adaptation takes more time than the parametric adaptation. Among the structural adaptation operations, adding a social relationship (\text{addSRelationship}) in the model takes the longest time, around 68 millisecond (ms) (on average), as it needs to update
the configuration of two social roles, whereas deleting a social role (\textit{delSocialRole}) takes the least time, around 5 milliseconds. For different \textit{parametric} adaptation operations, the required time is related to rule injections and deletions in the Drools engine and lies between 162 and 486 microseconds.

The box plots in Fig. 9.7 illustrate the \textit{adaptation overhead} results across a DSIM and a PSIM. Fig. 9.7a shows the total time required for the \textit{bindRP} (bind role-player) and \textit{unbindRP} (unbind role-player) structural adaptation. Fig. 9.7b shows the time required for each step in the \textit{bindRP} adaptation, including the time to add a URL to a social role (\textit{addURLtoSR}), to compute a role centric relationship (\textit{compRoleCenRel}) using Algorithm 1, to send a request to a PSIM (\textit{sendReqToPSIM}), and to execute the role acquisition method (\textit{exeRoleAcqu}). Fig. 9.7c shows the time required for each step in the \textit{unbindRP} adaptation, including the time to delete an URL from a role (\textit{delURLfromSR}), to send a request to a PSIM (\textit{sendReqToPSIM}), and to execute the relinquishment method (\textit{exeRoleRelq}). The results show that on average the \textit{bindRP} and \textit{unbindRP} operations take 100ms and 33ms, respectively, which we believe is an acceptable overhead in collaborative applications.

\subsection*{9.3.2 Evaluating Resource Consumption and System Performance}

To evaluate the SIM’s resource consumption and performance related to mediating social interactions, we deployed social interaction models with an increasing number of social roles and relationships, \textit{i.e.}, from 100 roles and 100 relationships to 500 roles and 500 relationships. We also deployed the players (one for each social role) with each player sending 12 messages per minute (msgs/min). The SIM’s workload increases with the number of active players playing roles in a running model, \textit{i.e.},

\[ \text{workload} = \text{number of active players} \times \text{message sending rate} \]

For instance, the workload becomes 1200 msgs/min for a model with 100 roles where all 100 players of these roles are active, and the workload increases to 2400 msgs/min for 200 roles. The results (average of 100 runs with 10 minutes each run) in Fig. 9.8a show that the SIM’s CPU consumption increases linearly with increasing workloads or sizes of the running social interaction model.
Figure 9.7: Adaptation across social interaction models caused by binding and unbinding role player structural adaptation
CHAPTER 9. EXPERIMENTAL EVALUATION

Figure 9.8: SIM’s resource consumption and performance related to mediating social interactions

(a) SIM’s CPU consumption for different workloads

(b) Comparing T1 test case with two benchmarks B1 and B2 in terms of CPU usage
To quantitatively evaluate the SIM’s CPU consumption, we compared the SIM’s CPU consumption with two benchmark workloads (B1 and B2). In B1, we created 100 very simple services where each of these services had an operation `addNumbers` that could return the sum of two numbers. Similarly in B2, we created 100 services where each of them had an operation `sendMessage` that could receive a message from a client and pass it on to another client. The service in B2 is very similar to the SIM as it also routes messages. However, SIM performs some additional tasks, i.e., checks the messages against constraints defined in the relationships. We also deployed a client for each of the 100 services of these benchmarks, where a client invokes the corresponding service at a rate of 12 requests/minute. We deployed B1 and B2 separately in the Axis2 web service engine and ran the experiment 100 times (10 minutes each time). For both B1 and B2, the server workload was 1200 requests/minute, i.e., the same as the T1 test case of SIM (see Fig. 9.8a). Fig. 9.8b shows that on average the SIM’s CPU consumption is a little higher than B1 and B2 due to the additional constraints evaluation.

To measure the performance, we computed the time taken by the SIM to process an incoming message and route it from one (source) role to another (destination) role. The box plots in Fig. 9.9 (right) (zoom in view of Fig. 9.9 (left) between 20 and 80 milliseconds processing time) shows that on average SIM takes 32 milliseconds to process and route a message regardless of the SIM workload. However, the most extreme value of the processing time (see Fig. 9.9 (left)) increases with increasing SIM workload.

### 9.4 Evaluating the SocioTelematics Application

In this section, we present the experimental results that demonstrate the feasibility and applicability of the SocioTelematics application with respect to the resource consumption and performance. In this experiment, the SocioPlatform was deployed in the Amazon EC2, while the SocioTelematics was running on an Android Samsung Galaxy Tab phone (ARMv7 800MHz processor, RAM 444MB, External storage 4GB) with 3G connection (used in a car). Thus, the experimental setup was a client-server architecture, as illustrated in Fig. 9.10.

---

3http://aws.amazon.com/ec2/
**Figure 9.9:** SIM’s performance results are illustrated using the box plots - the left figure depicts the most extreme value in outliers, the right figure (zoom in view of the left figure) highlights the average processing time.

**Figure 9.10:** Client-server architecture of the SocioTelematics application.

- A SocioTelematics application is installed in both the Car#1 and Car#2.
- Using the application drivers’ of Car#1 and Car#2 are sharing their GPS positions and interacting with each other over 3G network.
We deployed the domain-centric and player-centric social interaction models related to the cooperative convoy scenario, i.e., BudgetDSIM, AVISDSIM, ConvoyDSIM, Car#1PSIM and Car#2PSIM, to the SocioPlatform. The SIM component of the SocioPlatform takes these social interaction models as input and dynamically generates Web services with each social role being exposed as a service and each interaction term exposed as an operation in the service (as discussed in Section 7.4.3). SIM manages and mediates social interactions by routing the messages, coming from the SocioTelematics client application (i.e., player) running in a car, to the destination endpoint. For each of the player a proxy service is maintained at the server to store all the incoming messages directed to that player. Given the devices’ limited computation power and battery life – the proxy service allows the player (client application running in a device) to communicate asynchronously with the server (social interaction models) via SOAP interfaces.

Section 9.4.1 benchmarks the resource consumption of the SocioTelematics application on an Android Samsung Galaxy Tab phone. The results demonstrate that the application can be deployed on an off-the-shelf smart phone or Tablet with limited resource and computation power. Section 9.4.2 quantifies the performance of both the functional and management operations of the SocioTelematics application with respect to the communication latency between applications via the server. The results show that the application’s performance is acceptable in a cooperative convoy.

### 9.4.1 Evaluating Resource Consumption

We have profiled the resource consumption of the SocioTelematics prototype application on an Android Samsung Galaxy Tab, for CPU, memory (RAM), and storage. Fig. 9.11a and 9.11b show the application’s CPU and memory consumptions, respectively, over five minutes while the application was fetching messages from the SIM in every three seconds, updating Google maps view in every five seconds, and sending messages (i.e., performing interactions) once in every second.

Table 9.3 presents a comparison (average over a 5 minute run) of SocioTelematics with two widely used similar (in terms of service used) Android Applications - Gmail
and Google Maps. Gmail is an application that has a background service to interact with a web server for fetching and sending emails while the Google Maps uses a location service to show position on the map view. This comparison is performed using the built-in task manager application of Samsung Galaxy Tab. The results in Table 9.3 show that SocioTelematics consumes reasonable resources and is competitive with existing applications in terms of resource efficiency.

Table 9.3: SocioTelematics vs. benchmark applications – CPU, Memory and Storage usage

<table>
<thead>
<tr>
<th>Applications</th>
<th>CPU</th>
<th>Memory</th>
<th>Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>SocioTelematics</td>
<td>9.3%</td>
<td>9.60MB</td>
<td>520KB</td>
</tr>
<tr>
<td>Benchmark</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gmail</td>
<td>13%</td>
<td>15.27MB</td>
<td>1050KB</td>
</tr>
<tr>
<td>Google Maps</td>
<td>21%</td>
<td>25.80MB</td>
<td>2560KB</td>
</tr>
</tbody>
</table>

9.4.2 Quantifying Performance

We have examined both the functional and adaptation performance of the SocioTelematics using two cars in a cooperative convoy in 50 kilometres of driving.

To quantify the functional performance, we have evaluated the communication latency between applications. Fig. 9.12a shows that the application takes an average 1100 milliseconds to fetch a message from the server over 3G network. Fig. 9.12b shows that the server takes an average 250 milliseconds to route a message from a proxy to another proxy via Car#1PSIM, ConvoyDSIM and Car#2PSIM, on the server. It is worth noting that the communication latency is highly variable and generally depends on the network traffic and the server working load. In our testbed, we have also found that to pass a message from one client application to another client application takes around 1.6 seconds (given the fetching message interval is one second). This means that the application running in a car can display the other car’s position every 1.6 seconds which we believe is an acceptable time in a cooperative convoy scenario.

To quantify the application’s adaptation overhead, we have performed a number of adaptation operations, including add a car to the convoy and remove a car from the convoy. The results in Table 9.4 show that given the 1.12 seconds communication latency
Figure 9.11: Resource consumption of the *SocioTelematics* while it was fetching messages from the SIM in every three seconds, updating Google maps view in every five seconds, and sending messages (i.e., performing interactions) once in every second.
Figure 9.12: Communication latency
between the application and the server, on average the time to add and remove a car to and from the convoy at runtime takes 1.33 seconds (i.e., 1.121+0.209 seconds) and 1.167 seconds (i.e., 1.121+0.046 seconds), respectively. It takes around 0.42 milliseconds to modify the maxDesiredDistance operational parameter. We believe these adaptation times are acceptable in a cooperative convoy.

Table 9.4: Time required to perform adaptation operations

<table>
<thead>
<tr>
<th>Operations</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send an adaptation request from the SocioTelematics application to the Amazon server over a 3G network</td>
<td>1.121 sec</td>
</tr>
<tr>
<td>Add a new car to the ConvoyDSIM at runtime, i.e., addSocialRole, addSRelationship and bindRP</td>
<td>0.209 sec</td>
</tr>
<tr>
<td>Remove a car from the ConvoyDSIM at runtime, i.e., unbindRP, delSRelationship and delSocialRole</td>
<td>0.046 sec</td>
</tr>
<tr>
<td>Modify the maxDesiredDistance operational parameter in ConvoyDSIM</td>
<td>0.422 ms</td>
</tr>
</tbody>
</table>

9.5 Evaluating the SPCall Application

In this section, we present experimental results that demonstrate the feasibility and applicability of the SPCall application with respect to the resource consumption and performance. Like the client-server experimental setup for evaluating SocioTelematics, in this experiment, the SocioPlatform was deployed in the Amazon EC2, while the SPCall was running on the same Android Samsung Galaxy Tab phone with 3G internet connection.

Section 9.5.1 benchmarks the resource consumption of the SPCall application on an Android Samsung Galaxy Tab phone. The results demonstrate that the application can be deployed on an off-the-shelf smart phone and Tablet. Section 9.5.2 quantifies the performance of the SPCall application with respect to the communication latency between the application and the server. We observe that the application is able to acquire information from SocioPlatform and make decision in time before the call is forwarded to the voice-mail of the callee.
9.5.1 Evaluating Resource Consumption

In this experiment, we compare the SPCall application with an Android default dialler application and three highly downloaded (more than 10,000 times) and ratted (greater than 4 out of 5) Android applications that also aim to filter incoming calls. The dialer (some Android version named Phone) is a default Android application comes with every Android phone. It allows a person to make a phone call and manages phone call logs, i.e., incoming and outgoing calls. However, it does not provide automatic call filtering functionality. The other three proprietary Android applications (i.e., CallFilter[4], Call Control[5] and Call Blocker[6]) provide different features to its users. One common feature is filtering incoming phone calls based on user specified call filtering preferences. These applications only allow users to consider time in day and phone number for defining call filtering preferences. However, unlike SPCall, none of these applications able to recognise social relationships with the caller and allow users to consider such relationships for defining their call filtering preferences. Moreover, none of these applications have considered the caller perspective like the SPCall. That is, among all the applications mentioned above, only the SPCall allows a caller to know the situation of the intended callee to decide suitable time to call.

We have measured the resource consumption of these applications on an Android Samsung Galaxy Tab, for CPU, memory (RAM), and storage. Table 9.5 presents a comparison (average of 100 phone calls receiving attempts). The results show that SPCall consumes reasonable resources and is competitive with existing applications in terms of resource efficiency.

Table 9.5: SPcall vs. benchmark applications – CPU, Memory and Storage usage

<table>
<thead>
<tr>
<th>Applications</th>
<th>CPU</th>
<th>Memory</th>
<th>Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPCall</td>
<td>7.8%</td>
<td>4.6MB</td>
<td>527KB</td>
</tr>
<tr>
<td>Benchmark</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Android Dialler</td>
<td>11.2%</td>
<td>38.73MB</td>
<td>748KB</td>
</tr>
<tr>
<td>Call Filter</td>
<td>7.1%</td>
<td>3.6MB</td>
<td>1023KB</td>
</tr>
<tr>
<td>Call Control</td>
<td>17.7%</td>
<td>12MB</td>
<td>2365KB</td>
</tr>
<tr>
<td>Call Blocker</td>
<td>21%</td>
<td>23MB</td>
<td>6307KB</td>
</tr>
</tbody>
</table>

9.5.2 Quantifying Performance

In this experiment, we have used the SCOnto knowledge base that has been built in previous experiment for the user3. The knowledge base contains a reasonable number of different types of social context information such as 12 instances of family relationships, 202 instances of friend relationships, 32 instances of work relationships and 187 instances of living address based relationships (see Table 9.1). When a phone call comes, the SPCall application communicates with the SCIM of the SocioPlatform to collect users’ social context information such as social relationships and situations, and behaves according to the specified filtering preferences. In this experiment, we have defined 100 call filtering preferences including the five preferences that are described in the use case scenarios (see Section 8.2.7).

Table 9.6 presents the details performance result (an average of 100 runs) of the call filtering process. Major portion of the total time is taken for communication between the SPCall client and the SocioPlatform server, which was around 1106 milliseconds. It is worth noting that the communication time mainly depends on the internet speed of the mobile and the network traffic at the time of communication, and nothing related to the implementation of the SPCall client and the SocioPlatform. In the server side (running on the Amazon EC2), it took 13.64 milliseconds to process queries which are basically for answering the social relationships between the caller and callee, and answering the current situation of the callee. Once the SPCall client application, running on the mobile, receives the social relationship and situation information, it starts to check call filtering preferences. In our case, it took 7.4 milliseconds to check 100 filtering preferences. In total, it took around 1127 milliseconds to take a decision whether to ring or vibrate or reject or forward to the voice-mail. For the call filtering application, this time is very critical as if it takes longer to take a decision the call is automatically forwarded to the voice mail. In our experiment, in 7 out of 100 test runs, application could not successfully take decision on time. There are two possible options to overcome this limitation. First, using high speed mobile internet. Second, increasing the time limit to forwarding a call to voice mail.
Table 9.6: Call filtering performance of the \textit{SPCall} application

<table>
<thead>
<tr>
<th>Task</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPCall client to SocioPlatform (server) communication latency</td>
<td>1106 ms</td>
</tr>
<tr>
<td>Time required to process queries (server side)</td>
<td>13.64 ms</td>
</tr>
<tr>
<td>Time required to check call filtering preferences (SPCall client)</td>
<td>7.4 ms</td>
</tr>
<tr>
<td>Total</td>
<td>1127.04 ms</td>
</tr>
</tbody>
</table>

9.6 Summary

In this chapter, we have evaluated the SocioPlatform for developing social context-aware applications, using a series of experiments at two different fronts.

\textit{Firstly}, we have quantified the performance of the core functions provided by the SocioPlatform. As appropriate, in some cases, we have also measured the resource consumption. To evaluate the performance of some functions such as social context information acquisition, DL based reasoning and query processing, we have used real data from online social networks to evaluate these functions in a real world context. To evaluate the performance of other functions such as real time interaction events based situation reasoning, mediating social interactions and supporting runtime adaptation, we have used simulated data and application scenario based models, as appropriate. We have also quantified resource consumption and compared with two benchmark workloads related to mediating social interactions. The overall results of the experiment have shown that the performance and resource consumption of the SocioPlatform is acceptable with respect to the data volume size and platform workload.

\textit{Secondly}, we have quantified performance and measured resource consumption of two socially-aware applications using real-life experiment. These socially-aware applications, \textit{i.e., SocioTelematics} and \textit{SPCall}, have been developed using our proposed framework and the SocioPlatform. We have also compared the resource consumption of these applications with other existing similar applications. The results of this experiment show that the performance of these applications is acceptable in real-world scenario and the resource usage of these applications is reasonable.

In summary, the results of these experiments demonstrate the real-world applicability
and feasibility of developing socially-aware applications using our framework.
Chapter 10

Conclusion

We have presented, in this thesis, a framework for social context-aware applications, which includes an approach for modelling, reasoning about and managing social context information and a platform, namely SocioPlatform, for assisting the development of such applications. The motivation of this research stems from the need to address the limitations of existing approaches as well as the new challenges faced by the design and implementation of socially-aware applications in terms of social context information provisioning and adaptability. The outcome of this study is a novel framework for the social context-aware applications.

In this chapter, we first highlight the important contributions this thesis makes to the field of pervasive social computing, and then discuss some related topics for future investigation.

10.1 Contributions

In general, the aim of this thesis was to address the challenges associated with the engineering of social context-aware applications, by developing a framework consisting of both sound conceptual foundations and a supporting software platform. The framework differs significantly from the previously proposed solutions described in Chapter 3, such as Prometheus [124], as it is founded on a set of well-integrated approaches...
to modelling, reasoning about and managing social context that allows context to be described at varying levels of abstraction with its semantics, deducing more meaningful and interesting social context information, and managing its acquisition, provisioning and runtime adaptation. The framework makes a number of important research contributions, which are summarised in the remainder of this section.

I. An ontology based social context model that includes ontologies for social role, social relationship, social interaction and situation. In order to capture both the general and domain-specific concepts, we have provided both an upper and domain-specific ontologies. The upper ontology captures the basic concepts abstracted from the analysis of socially-aware applications and different sources of social context information. This upper ontology can be shared, reused and adapted to various applications. This can be customised to represent concepts in different forms for different users. It is also extensible to allow for the incorporation of new concepts and the specialisation of concepts and constraints for a particular application or domain, which we refer to as application- or domain-specific ontologies. As examples, domain-specific ontologies for family, educational organisation and cooperative convoy have been presented. One of the key characteristics of our modelling approach is that we model social context information, e.g., social relationship, as first-class entities rather than representing them as a generic link between people (as in existing approaches such as [182], discussed in Chapter 3). This way of modelling allows us to benefit from description logic (DL) in classifying and reasoning about social context information at different levels of abstraction based on the properties of social context.

II. An ontology based approach to reasoning about social context information. Socially-aware applications may require some social context information that cannot be acquired directly from the possible sources, but can be inferred from the acquired basic information. For example, an exact social relationship between an individual and a person can be derived from the individual’s gender information and the social role of the person who is related with the individual, acquired from Facebook. Moreover, new social relationships (e.g., best friend) and situations (e.g., busy), as required by applications can be deduced and abstracted from the exist-
ing concepts. We have adopted an ontology based approach to deriving, deducing and abstracting social context information considering their structural/parametric, time-stamp and abstraction aspects. To date, OWL2-DL has been the most practical choice for most ontological applications as it supports maximum expressiveness while retaining computational completeness and decidability [154]. As such, we have used the OWL2-DL class constructors, semantics and property characteristics to represent and specify reasoning rules. The main advantage of their use is that a description logic reasoner can be used for automatically reasoning about social context information in the knowledge base.

III. A novel approach to inferring users’ situations from their interaction events. The novelties of this approach lie in considering users’ interaction events in inferring situations and combining semantic matching and temporal aspects of interaction events in specifying and deducing situations. In doing so, an interaction event ontology is proposed to capture users’ interaction activities in different social interaction platforms. To capture both the general and domain-specific concepts, the ontology includes an upper interaction ontology, namely IntEO, which can be customised and specialised to form domain-specific ontologies. As examples, domain-specific ontologies for Facebook and Cooperative Convoy have been presented. To allow domain experts to specify situations based on interaction event patterns and their correlations, a situation specification language, both for the technical and non-technical persons, is proposed that integrates the IntEO with temporal relations, where IntEO serves as a vocabulary for domain experts to specify the attribute values in an interaction event pattern. The key features of this approach are that (1) the proposed situation language is expressive enough to cover the temporal aspects of situation specification and reasoning as the language incorporates all possible types of temporal correlations, and (2) it simplifies the situation rules by incorporating an interaction event ontology in the situation specification language.

IV. An ontology based privacy policy model and a language to specify privacy preferences. Users’ social context information is inherently sensitive. In pervasive social computing environments, users might need to allow others to access their social context information for greater benefits but this may also compromise their privacy.
Thus, users should be able to *retain control over who* has access to their personal information under *which* conditions. Our proposed policy model provides intuitive support by considering *social role, social relationship* and *situation* information when defining privacy preferences, and allowing owners to fine-tune the *granularity of accessed information*. In this regard, we have also provided a mechanism to check *inconsistency* in policy specification and to *enforce* users’ policies while accessing their social context information. The main advantages of our approach are that a description logic reasoner can be used for *detecting* automatically inconsistencies in the specification of policies, and *executing* policies.

V. An approach to modelling runtime social interactions. We have taken a role based approach to modelling runtime social interactions as it has been evaluated as being very useful in modelling entities, their relationships, functionality and interactions [49]. Our approach utilises the notion of roles for functional abstractions. Among the many different approaches to role modelling, one common way to develop role based models is to use organisational concepts [60]. We adopt the view of an organisation being a composition structure of dynamic social roles in order to model social interactions at runtime. The essences of our approach are the *externalisation* of the interaction relationships from the applications and the *explicit* modelling of such relationships from *domain-* and *player-perspectives*. The *domain-centric model* captures a *collaborative view* of the interaction relationships among the users/actors whereas the *player-centric model* captures an individual’s *coordinated view* of all its interactions (across different domains). The player-centric model allows a user to specify his/her *coordination preferences* and perform coordination in an *automated* manner. These runtime models are realised through a platform, and are used to *mediate social interactions* between users in interaction-centric socially-aware applications.

VI. An approach to managing runtime adaptation in social interaction models. Runtime social interaction models capture predefined agreements and constraints, *i.e.*, interaction relationships, among collaborative users in interaction-centric socially-aware applications. In complex and changing environments, such interaction relationships, and thus social interaction models, need to be *adapted* because of
the changes in users’ requirements and environments. To manage such adaptation: (1) we have identified different types of changes (i.e., user requirements and environments) and the various adaptations (i.e., parametric and structural) to deal with such changes; (2) we have proposed adaptation and management operations, and presented possible ways (i.e., manually and rule-based) to perform adaptations using those operations; (3) we have introduced social interaction states to support adaptation in a safe manner; and (4) finally, we have considered inherent dependencies among player- and domain-centric social interaction models, and proposed a protocol with associated algorithms for cross-model adaptation (i.e., adaptation propagation). The essence of our adaptation approach is the separation of functional and management operations, which facilitates the management of dynamic relationships between collaborative users, and supports their adaptation to cope with the continuous changes in their requirements and environments.

VII. Finally, a platform, namely SocioPlatform, to provide high-level support for developing socially-aware applications. The SocioPlatform integrates the aforementioned theoretical contributions into a system architecture and subsequently a platform by incorporating such capabilities as the acquisition, management and querying of social context information, providing an environment for executing social interaction models and managing their runtime adaptation, and presenting a set of APIs to developers to build socially-aware applications.

Overall, our framework for social context-aware applications addresses the key challenges of designing and implementing novel socially-aware applications, i.e., social context modelling, reasoning and management. In this thesis, we have presented how these key challenges can be addressed by our novel framework that includes techniques for modelling, reasoning about and managing different types of social context information, and platform for better supporting the development of socially-aware applications.

10.2 Future Work

Much work can be done, in the pervasive social computing area, and more specifically, to further enhance the framework. The following are three possible research directions for
further investigation.

**Usability Study** – Empirical usability evaluation of the proposed framework and the case study applications is important for several reasons. *First*, it is crucial in identifying new usability problems, and in exploring appropriate solutions to these. *Second*, it provides important validation and feedback in terms the suitability of proposed social context modelling, reasoning and management techniques such as those proposed in this thesis. *Finally*, it will eventually lead researchers to better understand the most compelling uses of social context-awareness, enabling the development of applications that are capable of achieving widespread acceptance and use [98].

**Quality of Social Context** – An essential topic for future research is that of quality of social context. The quality of context (QoC) plays an important role in improving context-based adaptation and in ensuring the correct behaviour of context-aware applications and services. The usefulness problem arising in context-aware software as a result of imperfect context information is well recognised. A number of research efforts (e.g., [99], [79], [134]) have attempted to present QoC parameters and QoC measuring methods for context of a physical nature such as location and time. Parameters to define QoC from different perspectives have been proposed. To characterise imperfect context information, Henricksen et al. [99] proposed four QoC parameters: *unknown*, *ambiguous*, *imprecise* and *erroneous*. Manzoor et al. [134] defined *up-to-dateness*, *trust-worthiness*, *completeness*, *consistency* and *precision* as generic QoC parameters, and proposed techniques to evaluate these parameters. All of the above work identified that errors in context information mainly arise as a result of sensing. The work presented in this thesis addressed the issue of *up-to-dateness* in acquiring users’ situation information from social media. However, many other aspects related to the quality of social context such as imprecise and trust-worthiness yet need to be addressed.

**Social Signal Processing and Behaviour Awareness** – One of the goal of pervasive social computing is to provide services to the human by detecting human social context and recognising human social intentions. As such, behaviour awareness is identified as an important research direction in pervasive social computing [196].
Vinciarelli et al. [183] argue that next generation computing needs to include the essence of social intelligence – the ability to recognise human social signals and social behaviours like politeness and disagreement, in order to become more effective and efficient in dealing with user needs. For example, an audio player in a mobile phone can choose music to play based on its user’s current behaviour pattern and mood. Behaviour awareness aims to analyse various kinds of behavioural cues and social signals, in order to identify their features and meanings. One of the research issues in this direction would be how ontology-based approach can be exploited in modelling behavioural cues into computer systems, so that social signals can be effectively analysed by machines in order to recognise human social intentions.
Bibliography


[19] W3c, w3c soap specifications. [http://www.w3.org/TR/soap/](http://www.w3.org/TR/soap/)

[20] W3c, web services description language (wsdl). [http://www.w3.org/TR/wsd120](http://www.w3.org/TR/wsd120)


Bibliography

HotMobile ’10, pages 60–65, New York, NY, USA, 2010. ACM.


puting and Communications Workshops (PERCOM Workshops), PERCOM ’09, pages 1–8, Washington, DC, USA, 2009. IEEE Computer Society.


[116] M. Kabir, J. Han, A. Colman, and J. Yu. Scaas: A platform for managing adaptation


[144] C. Pautasso and E. Wilde. Restful web services: principles, patterns, emerging


WWW ’10, pages 981–990, New York, NY, USA, 2010. ACM.


Appendix A

Description Logic-based Reasoning
Rules and Constraints

A.1 Classification and Reasoning Rules

\[\text{Father-Son} \equiv \big( \exists \text{relates}(\text{Person} \cap \text{plays} \cdot \{\text{father}\}) \cap \\
\exists \text{relates}(\text{Me} \cap \text{gender} \cdot \{\text{male}\}) \big) \cup \\
\big( \exists \text{relates}(\text{Person} \cap \text{plays} \cdot \{\text{son}\}) \cap \\
\exists \text{relates}(\text{Me} \cap \text{gender} \cdot \{\text{male}\}) \big) \quad (A.1)\]

\[\text{Mother-Son} \equiv \big( \exists \text{relates}(\text{Person} \cap \text{plays} \cdot \{\text{mother}\}) \cap \\
\exists \text{relates}(\text{Me} \cap \text{gender} \cdot \{\text{male}\}) \big) \cup \\
\big( \exists \text{relates}(\text{Person} \cap \text{plays} \cdot \{\text{son}\}) \cap \\
\exists \text{relates}(\text{Me} \cap \text{gender} \cdot \{\text{female}\}) \big) \]
APPENDIX A. DESCRIPTION LOGIC-BASED REASONING RULES AND CONSTRAINTS

Mother-Daughter ≡ ( ∃relates(Person ⊓ plays · {mother}) ⊓
                    ∃relates(Me ⊓ gender · {female})) ⊔
                   (∃relates(Person ⊓ plays · {daughter}) ⊓
                    ∃relates(Me ⊓ gender · {female}))

GrandFather-GrandSon ≡ ( ∃relates(Person ⊓ plays · {grandfather}) ⊓
                    ∃relates(Me ⊓ gender · {male})) ⊔
                   (∃relates(Person ⊓ plays · {grandson}) ⊓
                    ∃relates(Me ⊓ gender · {male}))

GrandFather-GrandDaughter ≡ ( ∃relates(Person ⊓ plays · {grandfather}) ⊓
                    ∃relates(Me ⊓ gender · {female})) ⊔
                   (∃relates(Person ⊓ plays · {granddaughter}) ⊓
                    ∃relates(Me ⊓ gender · {male}))

GrandMother-GrandSon ≡ ( ∃relates(Person ⊓ plays · {grandmother}) ⊓
                    ∃relates(Me ⊓ gender · {male})) ⊔
                   (∃relates(Person ⊓ plays · {grandson}) ⊓
                    ∃relates(Me ⊓ gender · {female}))

GrandMother-GrandDaughter ≡ ( ∃relates(Person ⊓ plays · {grandmother}) ⊓
                    ∃relates(Me ⊓ gender · {female})) ⊔
                   (∃relates(Person ⊓ plays · {granddaughter}) ⊓
                    ∃relates(Me ⊓ gender · {female}))
APPENDIX A. DESCRIPTION LOGIC-BASED REASONING RULES AND CONSTRAINTS

Uncle-Nephew \equiv ( \exists \text{relates}(\text{Person} \sqcap \text{plays} \cdot \{uncle\}) \sqcap \\
\exists \text{relates}(\text{Me} \sqcap \text{gender} \cdot \{male\}))

\sqcup ( \\
\exists \text{relates}(\text{Person} \sqcap \text{plays} \cdot \{nephew\}) \sqcap \\
\exists \text{relates}(\text{Me} \sqcap \text{gender} \cdot \{male\}))

Uncle-Niece \equiv ( \exists \text{relates}(\text{Person} \sqcap \text{plays} \cdot \{uncle\}) \sqcap \\
\exists \text{relates}(\text{Me} \sqcap \text{gender} \cdot \{female\}))

\sqcup ( \\
\exists \text{relates}(\text{Person} \sqcap \text{plays} \cdot \{niece\}) \sqcap \\
\exists \text{relates}(\text{Me} \sqcap \text{gender} \cdot \{male\}))

Aunt-Nephew \equiv ( \exists \text{relates}(\text{Person} \sqcap \text{plays} \cdot \{aunt\}) \sqcap \\
\exists \text{relates}(\text{Me} \sqcap \text{gender} \cdot \{male\}))

\sqcup ( \\
\exists \text{relates}(\text{Person} \sqcap \text{plays} \cdot \{nephew\}) \sqcap \\
\exists \text{relates}(\text{Me} \sqcap \text{gender} \cdot \{female\}))

Aunt-Niece \equiv ( \exists \text{relates}(\text{Person} \sqcap \text{plays} \cdot \{aunt\}) \sqcap \\
\exists \text{relates}(\text{Me} \sqcap \text{gender} \cdot \{female\}))

\sqcup ( \\
\exists \text{relates}(\text{Person} \sqcap \text{plays} \cdot \{niece\}) \sqcap \\
\exists \text{relates}(\text{Me} \sqcap \text{gender} \cdot \{female\}))

Father_in_law-Son_in_law \equiv ( \exists \text{relates}(\text{Person} \sqcap \text{plays} \cdot \{father\_in\_law\}) \sqcap \\
\exists \text{relates}(\text{Me} \sqcap \text{gender} \cdot \{male\}))

\sqcup ( \\
\exists \text{relates}(\text{Person} \sqcap \text{plays} \cdot \{son\_in\_law\}) \sqcap \\
\exists \text{relates}(\text{Me} \sqcap \text{gender} \cdot \{male\}))
APPENDIX A. DESCRIPTION LOGIC-BASED REASONING RULES AND CONSTRAINTS

\[
\text{Father\_in\_law-Daughter\_in\_law} \equiv (\exists \text{relates(Person} \cap \text{plays} \cdot \{\text{father\_in\_law}\}) \cap \\
\exists \text{relates(Me} \cap \text{gender} \cdot \{\text{female}\})) \\
\cup (\exists \text{relates(Person} \cap \text{plays} \cdot \{\text{daughter\_in\_law}\}) \cap \\
\exists \text{relates(Me} \cap \text{gender} \cdot \{\text{male}\})
\]

\[
\text{Mother\_in\_law-Son\_in\_law} \equiv (\exists \text{relates(Person} \cap \text{plays} \cdot \{\text{mother\_in\_law}\}) \cap \\
\exists \text{relates(Me} \cap \text{gender} \cdot \{\text{male}\}) \\
\cup (\exists \text{relates(Person} \cap \text{plays} \cdot \{\text{son\_in\_law}\}) \cap \\
\exists \text{relates(Me} \cap \text{gender} \cdot \{\text{female}\})
\]

\[
\text{Mother\_in\_law-Daughter\_in\_law} \equiv (\exists \text{relates(Person} \cap \text{plays} \cdot \{\text{mother\_in\_law}\}) \cap \\
\exists \text{relates(Me} \cap \text{gender} \cdot \{\text{female}\}) \\
\cup (\exists \text{relates(Person} \cap \text{plays} \cdot \{\text{daughter\_in\_law}\}) \cap \\
\exists \text{relates(Me} \cap \text{gender} \cdot \{\text{female}\})
\]

\[
\text{Brother-Brother} \equiv (\exists \text{relates(Person} \cap \text{plays} \cdot \{\text{brother}\}) \cap \\
\exists \text{relates(Me} \cap \text{gender} \cdot \{\text{male}\}))
\]

\[
\text{Sister-Sister} \equiv (\exists \text{relates(Person} \cap \text{plays} \cdot \{\text{sister}\}) \cap \\
\exists \text{relates(Me} \cap \text{gender} \cdot \{\text{female}\}))
\]

\[
\text{Brother-Sister} \equiv (\exists \text{relates(Person} \cap \text{plays} \cdot \{\text{brother}\}) \cap \\
\exists \text{relates(Me} \cap \text{gender} \cdot \{\text{female}\}) \\
\cup (\exists \text{relates(Person} \cap \text{plays} \cdot \{\text{sister}\}) \cap \\
\exists \text{relates(Me} \cap \text{gender} \cdot \{\text{male}\})
\]

\[
\text{Weekends} \equiv \exists \text{inDateTime(DateTimeDescription} \cap \\
(\text{dayOfWeek} \cdot \{\text{Saturday}\} \cup \text{dayOfWeek} \cdot \{\text{Sunday}\}))
\]
APPENDIX A. DESCRIPTION LOGIC-BASED REASONING RULES AND CONSTRAINTS

Weekdays ≡ \exists inDateTime(\text{DateDescription} \sqcap \\
(\text{dayOfWeek} \cdot \{\text{Monday}\} \sqcup \text{dayOfWeek} \cdot \{\text{Tuesday}\} \\
\sqcup \text{dayOfWeek} \cdot \{\text{Wednesday}\} \sqcup \text{dayOfWeek} \cdot \{\text{Thursday}\} \\
\sqcup \text{dayOfWeek} \cdot \{\text{Friday}\}))

Holidays ≡ \text{NewYearDay} \sqcup \text{ChristmasDay} \sqcup \text{BoxingDay} \\
\sqcup \text{GoodFriday} \sqcup \text{LabourDay} \sqcup \text{AustraliaDay} \\
\sqcup \text{QueenBrithDay} \sqcup \text{ANZACDay} \sqcup \text{EsaterMonday} \\
\sqcup \text{MelbourneCupDay} \sqcup \text{SaturdayBeforeEasterSunday}

NewYearDay ≡ \exists inDateTime(\text{DateDescription} \sqcap \\
(\text{year} \cdot \{2013\} \sqcap \text{month} \cdot \{-01\} \sqcap \text{day} \cdot \{-01\}))

ChristmasDay ≡ \exists inDateTime(\text{DateDescription} \sqcap \\
(\text{year} \cdot \{2013\} \sqcap \text{month} \cdot \{-12\} \sqcap \text{day} \cdot \{-25\}))

EasterMonday ≡ \exists inDateTime(\text{DateDescription} \sqcap \\
(\text{year} \cdot \{2013\} \sqcap \text{month} \cdot \{-04\} \sqcap \text{day} \cdot \{-01\}))

BoxingDay ≡ \exists inDateTime(\text{DateDescription} \sqcap \\
(\text{year} \cdot \{2013\} \sqcap \text{month} \cdot \{-12\} \sqcap \text{day} \cdot \{-26\}))

GoodFriday ≡ \exists inDateTime(\text{DateDescription} \sqcap \\
(\text{year} \cdot \{2013\} \sqcap \text{month} \cdot \{-03\} \sqcap \text{day} \cdot \{-29\}))

LabourDay ≡ \exists inDateTime(\text{DateDescription} \sqcap \\
(\text{year} \cdot \{2013\} \sqcap \text{month} \cdot \{-03\} \sqcap \text{day} \cdot \{-11\}))

AustraliaDay ≡ \exists inDateTime(\text{DateDescription} \sqcap \\
(\text{year} \cdot \{2013\} \sqcap \text{month} \cdot \{-01\} \sqcap \text{day} \cdot \{-28\}))

QueenBrithDay ≡ \exists inDateTime(\text{DateDescription} \sqcap \\
(\text{year} \cdot \{2013\} \sqcap \text{month} \cdot \{-06\} \sqcap \text{day} \cdot \{-10\}))

ANZACDay ≡ \exists inDateTime(\text{DateDescription} \sqcap \\
(\text{year} \cdot \{2013\} \sqcap \text{month} \cdot \{-04\} \sqcap \text{day} \cdot \{-25\}))
**MelbourneCupDay**  \( \equiv \exists \text{inDateTime}(\text{DateTimeDescription} \sqcap \text{year} \cdot \{2013\} \sqcap \text{month} \cdot \{-11\} \sqcap \text{day} \cdot \{-5\}) \)

**SaturdayBeforeEasterSunday**  \( \equiv \exists \text{inDateTime}(\text{DateTimeDescription} \sqcap \text{year} \cdot \{2013\} \sqcap \text{month} \cdot \{-3\} \sqcap \text{day} \cdot \{-30\}) \)

**BusinessHours**  \( \equiv \text{Weekdays} \sqcap \neg \text{Holidays} \sqcap \exists \text{inDateTime}(\text{DateTimeDescription} \sqcap \text{hour} \cdot \{9\} \sqcap \text{hour} \leq \{16\}) \)

**LunchHours**  \( \equiv \text{inDateTime}(\text{DateTimeDescription} \sqcap \text{hour} \cdot \{12\}) \)

**WorkingHours**  \( \equiv \text{BusinessHours} \sqcap \neg \text{LunchHours} \)

**NonBusinessHours**  \( \equiv \neg \text{BusinessHours} \)
APPENDIX A. DESCRIPTION LOGIC-BASED REASONING RULES AND CONSTRAINTS

A.2 Constraints

\[
\begin{align*}
\text{Father}_in\_law-Son\_in\_law & \sqcap \text{Father}_in\_law-Daughter\_in\_law = \bot \\
\text{Mother}_in\_law-Son\_in\_law & \sqcap \text{Mother}_in\_law-Daughter\_in\_law = \bot \\
\text{Father}_in\_law-Child\_in\_law & \sqcap \text{Mother}_in\_law-Child\_in\_law = \bot \\
\text{Mother-Son} & \sqcap \text{Mother-Daughter} = \bot \\
\text{Father-Child} & \sqcap \text{Mother-Child} = \bot \\
\text{Grandfather-Grandson} & \sqcap \text{Grandfather-Granddaughter} = \bot \\
\text{Grandmother-Grandson} & \sqcap \text{Grandmother-Granddaughter} = \bot \\
\text{Grandfather-Grandchild} & \sqcap \text{Grandmother-Grandchild} = \bot \\
\text{Brother-Brother} & \sqcap \text{Brother-Sister} = \bot \\
\text{Brother-Brother} & \sqcap \text{Sister-Sister} = \bot \\
\text{Brother-Sister} & \sqcap \text{Sister-Sister} = \bot \\
\text{Father-Daughter} & \sqsubseteq \text{relates} \leq 1(\text{Person} \sqcap \text{plays} \cdot \text{father}) \\
\text{Mother-Son} & \sqsubseteq \text{relates} \leq 1(\text{Person} \sqcap \text{plays} \cdot \text{mother}) \\
\text{Mother-Daughter} & \sqsubseteq \text{relates} \leq 1(\text{Person} \sqcap \text{plays} \cdot \text{mother}) \\
\text{Father}_in\_law-Daughter\_in\_law & \sqsubseteq \text{relates} \leq 1(\text{Person} \sqcap \text{plays} \cdot \text{father}_in\_law) \\
\text{Mother}_in\_law-Daughter\_in\_law & \sqsubseteq \text{relates} \leq 1(\text{Person} \sqcap \text{plays} \cdot \text{mother}_in\_law)
\end{align*}
\]
A.2.1 Constraints Defined in SACOnto

\[
\begin{align*}
\text{Policy} & \sqsubseteq 1 \text{policyStatement} \\
\text{Granted} \sqcap \text{Denied} & \sqsubseteq \bot \\
\text{Granted} \sqcap \text{GrantedInGL1} & \sqsubseteq \bot \\
\text{Granted} \sqcap \text{GrantedInGL2} & \sqsubseteq \bot \\
\text{Granted} \sqcap \text{GrantedInGL3} & \sqsubseteq \bot \\
\text{Granted} \sqcap \text{GrantedInGL4} & \sqsubseteq \bot \\
\text{Denied} \sqcap \text{GrantedInGL1} & \sqsubseteq \bot \\
\text{Denied} \sqcap \text{GrantedInGL2} & \sqsubseteq \bot \\
\text{Denied} \sqcap \text{GrantedInGL3} & \sqsubseteq \bot \\
\text{Denied} \sqcap \text{GrantedInGL4} & \sqsubseteq \bot \\
\text{GrantedInGL1} \sqcap \text{GrantedInGL2} & \sqsubseteq \bot \\
\text{GrantedInGL1} \sqcap \text{GrantedInGL3} & \sqsubseteq \bot \\
\text{GrantedInGL1} \sqcap \text{GrantedInGL4} & \sqsubseteq \bot \\
\text{GrantedInGL2} \sqcap \text{GrantedInGL3} & \sqsubseteq \bot \\
\text{GrantedInGL2} \sqcap \text{GrantedInGL4} & \sqsubseteq \bot \\
\text{GrantedInGL3} \sqcap \text{GrantedInGL4} & \sqsubseteq \bot
\end{align*}
\]
### Appendix B

### Social Interactions in a Cooperative Convoy

**Table B.1:** Interaction (i1): Leading car notifies its position to the following car

<table>
<thead>
<tr>
<th>:i1</th>
<th>rdf:type</th>
<th>:Interaction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>:msgSignature</td>
<td>&quot;notifyPosition&quot;</td>
</tr>
<tr>
<td></td>
<td>:fromRole</td>
<td>:LeadingCar</td>
</tr>
<tr>
<td></td>
<td>:toRole</td>
<td>:FollowingCar</td>
</tr>
</tbody>
</table>

**Table B.2:** Interaction (i2): Following car notifies its position to the leading car

<table>
<thead>
<tr>
<th>:i2</th>
<th>rdf:type</th>
<th>:Interaction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>:msgSignature</td>
<td>&quot;notifyPosition&quot;</td>
</tr>
<tr>
<td></td>
<td>:fromRole</td>
<td>:FollowingCar</td>
</tr>
<tr>
<td></td>
<td>:toRole</td>
<td>:LeadingCar</td>
</tr>
</tbody>
</table>
**Table B.3:** Obligation (o1) and related Operational parameter (p1): Both the leading and following cars notify each other of their positions every 10 seconds

<table>
<thead>
<tr>
<th>:p1</th>
<th>rdf:type</th>
<th>OperationalParameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>:p1</td>
<td>:paramName</td>
<td>“posUpdateFreq”</td>
</tr>
<tr>
<td>:p1</td>
<td>:value</td>
<td>10</td>
</tr>
<tr>
<td>:p1</td>
<td>:timeUnit</td>
<td>“SEC”</td>
</tr>
<tr>
<td>:o1</td>
<td>rdf:type</td>
<td>:Obligation</td>
</tr>
<tr>
<td>:o1</td>
<td>:time</td>
<td>“duration”</td>
</tr>
<tr>
<td>:o1</td>
<td>:relatedOP</td>
<td>:p1</td>
</tr>
<tr>
<td>:i1</td>
<td>:hasObg</td>
<td>:o1</td>
</tr>
<tr>
<td>:i2</td>
<td>:hasObg</td>
<td>:o1</td>
</tr>
</tbody>
</table>

**Table B.4:** Obligation (o2) and related Operational parameter (p2): Maximum desired distance between the leading car and following car is 1 kilometer

<table>
<thead>
<tr>
<th>:p2</th>
<th>rdf:type</th>
<th>OperationalParameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>:p2</td>
<td>:paramName</td>
<td>“maxDesiredDistance”</td>
</tr>
<tr>
<td>:p2</td>
<td>:value</td>
<td>1</td>
</tr>
<tr>
<td>:p2</td>
<td>:distanceUnit</td>
<td>“KM”</td>
</tr>
<tr>
<td>:o2</td>
<td>rdf:type</td>
<td>:Obligation</td>
</tr>
<tr>
<td>:o2</td>
<td>:time</td>
<td>“duration”</td>
</tr>
<tr>
<td>:o2</td>
<td>:relatedOP</td>
<td>:p1</td>
</tr>
<tr>
<td>:i1</td>
<td>:hasObg</td>
<td>:o2</td>
</tr>
<tr>
<td>:i2</td>
<td>:hasObg</td>
<td>:o2</td>
</tr>
</tbody>
</table>

**Table B.5:** Interaction (i3): Following car sends ahead road block information to the leading car

<table>
<thead>
<tr>
<th>:i3</th>
<th>rdf:type</th>
<th>:Interaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>:i3</td>
<td>:msgSignature</td>
<td>“notifyRoadBlock”</td>
</tr>
<tr>
<td>:i3</td>
<td>:fromRole</td>
<td>:FollowingCar</td>
</tr>
<tr>
<td>:i3</td>
<td>:toRole</td>
<td>:LeadingCar</td>
</tr>
</tbody>
</table>

**Table B.6:** Interaction (i4): Leading car updates the route information to the following car

<table>
<thead>
<tr>
<th>:i4</th>
<th>rdf:type</th>
<th>:Interaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>:i4</td>
<td>:msgSignature</td>
<td>“routeUpdate”</td>
</tr>
<tr>
<td>:i4</td>
<td>:fromRole</td>
<td>:LeadingCar</td>
</tr>
<tr>
<td>:i4</td>
<td>:toRole</td>
<td>:FollowingCar</td>
</tr>
</tbody>
</table>
### Table B.7: Interaction (i5): Leading car notifies its mechanical issue to the following car

<table>
<thead>
<tr>
<th>:i5</th>
<th>rdf:type</th>
<th>:Interaction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>:msgSignature</td>
<td>&quot;notifyMechanicalIssue&quot;</td>
</tr>
<tr>
<td>:i5</td>
<td>:fromRole</td>
<td>:LeadingCar</td>
</tr>
<tr>
<td></td>
<td>:toRole</td>
<td>:FollowingCar</td>
</tr>
</tbody>
</table>

### Table B.8: Interaction (i6): Following car notifies its mechanical issue to the leading car

<table>
<thead>
<tr>
<th>:i6</th>
<th>rdf:type</th>
<th>:Interaction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>:msgSignature</td>
<td>&quot;notifyMechanicalIssue&quot;</td>
</tr>
<tr>
<td>:i6</td>
<td>:fromRole</td>
<td>:FollowingCar</td>
</tr>
<tr>
<td></td>
<td>:toRole</td>
<td>:LeadingCar</td>
</tr>
</tbody>
</table>

### Table B.9: Interaction (i7): a rented car sends a route request to a travel guide

<table>
<thead>
<tr>
<th>:i7</th>
<th>rdf:type</th>
<th>:Interaction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>:msgSignature</td>
<td>&quot;routeRequest&quot;</td>
</tr>
<tr>
<td>:i7</td>
<td>:fromRole</td>
<td>:RentedCar</td>
</tr>
<tr>
<td></td>
<td>:toRole</td>
<td>:TravelGuide</td>
</tr>
</tbody>
</table>

### Table B.10: Interaction (i8): a travel guide sends a route notification to a rented car

<table>
<thead>
<tr>
<th>:i8</th>
<th>rdf:type</th>
<th>:Interaction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>:msgSignature</td>
<td>&quot;routeNotification&quot;</td>
</tr>
<tr>
<td>:i8</td>
<td>:fromRole</td>
<td>:TravelGuide</td>
</tr>
<tr>
<td></td>
<td>:toRole</td>
<td>:RentedCar</td>
</tr>
</tbody>
</table>
**Table B.11:** Conversation (c1), and related Obligation (o3) and Operational Parameter (p3): Upon receiving a route request, travel guide should send a route notification by 15 seconds

| p3 rdf:type | :OperationalParameter |
| p3 :paramName | "maxTimeDelay" |
| p3 :value | 15 |
| p3 :timeUnit | SEC |
| o3 rdf:type | :Obligation |
| o3 :timer :duration |  |
| o3 :relatedOP | p3 |
| c1 rdf:type | :Conversation |
| c1 :intrac1 | i7 |
| c1 :intrac2 | i8 |
| c1 :hasSequence :leadsTo |  |
| c1 :relatedObg | o3 |

**Table B.12:** Interaction (i9): Driver of the rented car notifies a mechanical issue to the rental company

| i9 rdf:type | :Interaction |
| i9 :msgSignature | "notifyMechanicalIssue" |
| i9 :fromRole | :RentedCar |
| i9 :toRole | :RentalCompany |

**Table B.13:** Interaction (i10): Driver of the rented car requests for a mechanical assistance to the roadside assistance service

| i10 rdf:type | :Interaction |
| i10 :msgSignature | "mechanicalAssistanceRequest" |
| i10 :fromRole | :RentedCar |
| i10 :toRole | :RoadsideAssistance |
### Table B.14: Interaction (i11): A roadside assistance service responds on an assistance request

| :i11 | rdf:type         | :Interaction       |
| :---  | :msgSignature   | "assistanceResponse" |
| :---  | :fromRole       | :RoadsideAssistance |
| :---  | :toRole         | :RentedCar         |

### Table B.15: Conversation (c2), and related Obligation (o4) and Operational Parameter (p4): Upon receiving a mechanical assistance request, roadside assistance service should send an assistance response by 5 minutes

| :p4   | rdf:type     | :OperationalParameter |
| :---   | :paramName   | "maxTimeDelay"         |
| :---   | :value       | 5                       |
| :---   | :timeUnit    | MIN                     |
| :o4   | rdf:type     | :Obligation            |
| :---   | :timer       | :duration               |
| :---   | :relatedOP   | :p4                     |
| :c2   | rdf:type     | :Conversation          |
| :---   | :intrac1     | :i10                    |
| :---   | :intrac2     | :i11                    |
| :---   | :hasSequence | :leadsTo                |
| :---   | :relatedObg  | :o4                     |

### Table B.16: Interaction (i12): Roadside assistance service verifies car id with the rental company

| :i12  | rdf:type         | :Interaction           |
| :---   | :msgSignature   | "carIDVerificationRequest" |
| :---   | :fromRole       | :RoadsideAssistance    |
| :---   | :toRole         | :RentalCompany         |
# Appendix B. Social Interactions in a Cooperative Convoy

**Table B.17**: Interaction (i13): Rental company sends car id verification result to the roadside assistance

<table>
<thead>
<tr>
<th>:i13</th>
<th>rdf:type</th>
<th>:Interaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>:i13 :msgSignature</td>
<td>&quot;carIDVerificationResult&quot;</td>
<td>:RentalCompany</td>
</tr>
<tr>
<td>:i13 :toRole</td>
<td>:RentalCompany</td>
<td>:RoadsideAssistance</td>
</tr>
</tbody>
</table>

**Table B.18**: Conversation (c3): Upon receiving a car id verification request from a roadside assistance, the car rental company verifies the car id and sends the result back to the roadside assistance service

<table>
<thead>
<tr>
<th>:c3</th>
<th>rdf:type</th>
<th>:Conversation</th>
</tr>
</thead>
<tbody>
<tr>
<td>:c3 :intrac1</td>
<td>:i12</td>
<td>:i13</td>
</tr>
<tr>
<td>:c3 :intrac2</td>
<td>:i13</td>
<td></td>
</tr>
<tr>
<td>:c3 :hasSequence</td>
<td>:leadsTo</td>
<td></td>
</tr>
</tbody>
</table>

**Table B.19**: Interaction (i14): A driver of the rented car requests to a traffic management service to check whether a road is blocked to a destination

<table>
<thead>
<tr>
<th>:i14</th>
<th>rdf:type</th>
<th>:Interaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>:i14 :msgSignature</td>
<td>&quot;roadBlockCheckingRequest&quot;</td>
<td>:RentedCar</td>
</tr>
<tr>
<td>:i14 :fromRole</td>
<td>:RentedCar</td>
<td>:TrafficManagement</td>
</tr>
</tbody>
</table>
### Table B.20: Interaction (i15): A traffic management service responses on a road block checking request

<table>
<thead>
<tr>
<th>i15</th>
<th>rdf:type</th>
<th>Interaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>i15</td>
<td>:msgSignature</td>
<td>&quot;roadBlockCheckingResult&quot;</td>
</tr>
<tr>
<td>i15</td>
<td>:fromRole</td>
<td>:TrafficManagement</td>
</tr>
<tr>
<td>i15</td>
<td>:toRole</td>
<td>:RentedCar</td>
</tr>
</tbody>
</table>

### Table B.21: Conversation (c4), and related Obligation (o6) and Operational Parameter (p6): Upon receiving a road block checking request, traffic management service should send the checking result by 10 seconds

<table>
<thead>
<tr>
<th>p6</th>
<th>rdf:type</th>
<th>OperationalParameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>p6</td>
<td>:paramName</td>
<td>&quot;maxTimeDelay&quot;</td>
</tr>
<tr>
<td>p6</td>
<td>:value</td>
<td>10</td>
</tr>
<tr>
<td>p6</td>
<td>:timeUnit</td>
<td>SEC</td>
</tr>
<tr>
<td>o6</td>
<td>rdf:type</td>
<td>Obligation</td>
</tr>
<tr>
<td>o6</td>
<td>:timer</td>
<td>:duration</td>
</tr>
<tr>
<td>o6</td>
<td>:relatedOP</td>
<td>:p6</td>
</tr>
<tr>
<td>c4</td>
<td>rdf:type</td>
<td>Conversation</td>
</tr>
<tr>
<td>c4</td>
<td>:intrac1</td>
<td>:i14</td>
</tr>
<tr>
<td>c4</td>
<td>:intrac2</td>
<td>:i15</td>
</tr>
<tr>
<td>c4</td>
<td>:hasSequence</td>
<td>:leadsTo</td>
</tr>
<tr>
<td>c4</td>
<td>:relatedObg</td>
<td>:o6</td>
</tr>
</tbody>
</table>

### Table B.22: Interaction (i16): Following car proposes a convoy break to the leading car

<table>
<thead>
<tr>
<th>i16</th>
<th>rdf:type</th>
<th>Interaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>i16</td>
<td>:msgSignature</td>
<td>&quot;convoyBreak&quot;</td>
</tr>
<tr>
<td>i16</td>
<td>:fromRole</td>
<td>:FollowingCar</td>
</tr>
<tr>
<td>i16</td>
<td>:toRole</td>
<td>:LeadingCar</td>
</tr>
</tbody>
</table>

### Table B.23: Interaction (i17): Leading car agrees to the convoy break request

<table>
<thead>
<tr>
<th>i17</th>
<th>rdf:type</th>
<th>Interaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>i17</td>
<td>:msgSignature</td>
<td>&quot;positiveResponse&quot;</td>
</tr>
<tr>
<td>i17</td>
<td>:fromRole</td>
<td>:LeadingCar</td>
</tr>
<tr>
<td>i17</td>
<td>:toRole</td>
<td>:FollowingCar</td>
</tr>
</tbody>
</table>
### Appendix B. Social Interactions in a Cooperative Convoy

**Table B.24:** Interaction (i18): Address request from a rented car to a travel guide service

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>:i18</td>
<td>rdf:type</td>
<td>:Interaction</td>
</tr>
<tr>
<td>:i18</td>
<td>:msgSignature</td>
<td>&quot;addressRequest&quot;</td>
</tr>
<tr>
<td>:i18</td>
<td>:fromRole</td>
<td>:RentedCar</td>
</tr>
<tr>
<td>:i18</td>
<td>:toRole</td>
<td>:TravelGuide</td>
</tr>
</tbody>
</table>

**Table B.25:** Interaction (i19): Travel guide service sends address(es), as requested, to a rented car

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>:i19</td>
<td>rdf:type</td>
<td>:Interaction</td>
</tr>
<tr>
<td>:i19</td>
<td>:msgSignature</td>
<td>&quot;responseAddress&quot;</td>
</tr>
<tr>
<td>:i19</td>
<td>:fromRole</td>
<td>:TravelGuide</td>
</tr>
<tr>
<td>:i19</td>
<td>:toRole</td>
<td>:RentedCar</td>
</tr>
</tbody>
</table>
Appendix C

Social Interaction Models

C.1 Domain-centric Social Interaction Models (DSIMs)

C.1.1 ConvoyDSIM

DSIM ConvoyDSIM{
    SocialRole LeadingCar, FollowingCar;
    OrganiserRole Organiser;

    RolePlayerBinding org{
        RoleName Organiser;
        Player Car#1.SocioTelematics;
    };
    RolePlayerBinding b1{
        RoleName LeadingCar;
        Player Car#1PSIM;
    };
    RolePlayerBinding b2{
        RoleName FollowingCar;
        Player Car#2PSIM;
    };

    SocialRelationship R1{
        RelationshipName LeadingCar-FollowingCar;
        RoleA LeadingCar;
        RoleB FollowingCar;
        SocialInteraction i1{
            MsgSignature notifyPosition(float:latitude,float:longitude);
            Direction RoleBToRoleA;
        };
    };
}
RelatedObligation o1, o2;
};

SocialInteraction i2{
    MsgSignature notifyPosition(float:latitude, float:longitude);
    Direction RoleAToRoleB;
    RelatedObligation o1, o2;
};

SocialInteraction i3{
    MsgSignature notifyRoadBlock(String roadAddress);
    Direction RoleBToRoleA;
};

SocialInteraction i4{
    MsgSignature routeUpdate(String route);
    Direction RoleAToRoleB;
};

SocialInteraction i5{
    MsgSignature notifyMechanicalIssue(String issueDescription);
    Direction RoleAToRoleB;
};

SocialInteraction i6{
    MsgSignature notifyMechanicalIssue(String issueDescription);
    Direction RoleBToRoleA;
};

Obligation o1{
    Timer PERIOD;
    RelatedOperationalParam p1;
};

Obligation o2{
    RelatedOperationalParam p2;
};

OperationalParam p1{
    OPParamName posUpdateFreq;
    OPParamValue 10;
    TimeUnit SEC;
};

OperationalParam p2{
    OPParamName maxDesiredDistance;
    OPParamValue 1;
    DistanceUnit KM;
};

SocialInteraction i16{
    MsgSignature convoyBreak(String cause);
    Direction RoleBToRoleA;
};

SocialInteraction i17{
    MsgSignature positiveResponse(String cause);
    Direction RoleAToRoleB;
};
C.1.2 BudgetDSIM

**DSIM** BudgetDSIM{
  SocialRole RentedCar, RoadsideAssistance;
  SocialRole RentalCompany, TravelGuide;
  OrganiserRole Organiser;

  RolePlayerBinding org{
    RoleName Organiser;
    Player Budget;
  }

  RolePlayerBinding b3{
    RoleName RentedCar;
    Player Car#1PSIM;
  }

  RolePlayerBinding b4{
    RoleName RoadsideAssistance;
    Player RACV;
  }

  RolePlayerBinding b5{
    RoleName RentalCompany;
    Player Budget;
  }

  RolePlayerBinding b6{
    RoleName TravelGuide;
    Player TravelGuide.org;
  }

  RolePlayerBinding b3{
    RoleName OrganiserRole;
    Player Budget;
  }

  SocialRelationship R2{
    RelationshipName RentedCar-TravelGuide;
    RoleA RentedCar;
    RoleB TravelGuide;
    SocialInteraction i7{
      MsgSignature routeRequest(String sourceAddr, String destAddr,
                                String preference);
      Direction RoleAToRoleB;
    }

    SocialInteraction i8{
      MsgSignature routeNotification(String route);
      Direction RoleBToRoleA;
    }
}
SocialInteraction i8{
    MsgSignature addressRequest(String placeType);
    Direction RoleAToRoleB;
};

SocialInteraction i9{
    MsgSignature responseAddress(String addresses);
    Direction RoleBToRoleA;
};

Conversation c1{
    SocialInteractionX i7;
    SocialInteractionX i8;
    Sequence leadsTo; // i7 leadsTo i8
    RelatedObligation o3;
};

Obligation o3{
    Timer DURATION;
    RelatedOperationalParam p3;
};

OperationalParam p3{
    OPParamName maxTimeDelay;
    OPParamValue 15;
    TimeUnit SEC;
};

SocialRelationship R3{
    RelationshipName RentedCar-RentalCompany;
    RoleA RentedCar;
    RoleB RentalCompany;
    SocialInteraction i9{
        MsgSignature notifyMechanicalIssue(String issueDescription);
        Direction RoleAToRoleB;
    };
}

SocialRelationship R4{
    RelationshipName RentedCar-RoadsideAssistance;
    RoleA RentedCar;
    RoleB RoadsideAssistance;
    SocialInteraction i10{
        MsgSignature mechanicalAssistanceRequest(String issueDescription);
        Direction RoleAToRoleB;
    };
    SocialInteraction i11{
        MsgSignature assistanceResponse(String description);
        Direction RoleBToRoleA;
    };
}
C.1.3 AVISDSIM

**AVISDSIM**

**DSIM** AVISDSIM{
  SocialRole RentedCar, RoadsideAssistance;
  SocialRole RentalCompany, TrafficManagement;
  OrganiserRole Organiser;

  RolePlayerBinding org{
    RoleName Organiser;
  }

Player AVIS;
};
RolePlayerBinding b7{
    RoleName RentedCar;
    Player Car#2PSIM;
};
RolePlayerBinding b8{
    RoleName RoadsideAssistance;
    Player RACV;
};
RolePlayerBinding b9{
    RoleName RentalCompany;
    Player AVIS;
};
RolePlayerBinding b10{
    RoleName TrafficManagement;
    Player VicRoad.org;
};

SocialRelationship R6{
    RelationshipName RentedCar-TrafficManagement;
    RoleA RentedCar;
    RoleB TrafficManagement;
    SocialInteraction i14{
        MsgSignature roadBlockCheckingRequest(String sourceAddr,
            String destAddr);
        Direction RoleAToRoleB;
    }; SocialInteraction i15{
        MsgSignature roadBlockCheckingResult(String address);
        Direction RoleBToRoleA;
    }; Conversation c4{
        SocialInteractionX i14;
        SocialInteractionX i15;
        Sequence leadsTo;
        RelatedObligation o6;
    }; Obligation o6{
        Timer DURATION;
        RelatedOperationalParam p6;
    }; OperationalParam p6{
        OPParamName maxTimeDelay;
        OPParamValue 10;
        TimeUnit SEC;
    };
}
SocialRelationship R7{
  RelationshipName RentedCar-RentalCompany;
  RoleA RentedCar;
  RoleB RentalCompany;
  SocialInteraction i16{
    MsgSignature notifyMechanicalIssue(String issueDescription);
    Direction RoleAToRoleB;
  }
};

SocialRelationship R8{
  RelationshipName RentedCar-RoadsideAssistance;
  RoleA RentedCar;
  RoleB RoadsideAssistance;
  SocialInteraction i17{
    MsgSignature mechanicalAssistanceRequest(String issueDescription);
    Direction RoleAToRoleB;
  }
  SocialInteraction i18{
    MsgSignature assistanceResponse(String description);
    Direction RoleBToRoleA;
  }
};

Conversation c5{
  SocialInteractionX i17;
  SocialInteractionX i18;
  Sequence leadsTo; // i17 leadsTo i18
  RelatedObligation o7;
};

Obligation o7{
  Timer DURATION;
  RelatedOperationalParam p7;
};

OperationalParam p7{
  OPParamName maxTimeDelay;
  OPParamValue 8;
  TimeUnit MIN;
};

SocialRelationship R9{
  RelationshipName RoadsideAssistance-RentalCompany;
  RoleA RoadsideAssistance;
  RoleB RentalCompany;
  SocialInteraction i19{
    MsgSignature carIDVerificationRequest(String carID);
    Direction RoleAToRoleB;
  }
};
C.2 Player-centric Social Interaction Models (PSIMs)

C.2.1 Car#1PSIM

PSIM Car#1PSIM{
    SocialRole LeadingCar, RentedCar;
    CoordinatorRole Coordinator;
    OrganiserRole Organiser;

    RolePlayerBinding org{
        RoleName Organiser;
        Player Car#1.SocioTelematics;
    };
    RolePlayerBinding b1{
        RoleName LeadingCar;
        Player Car#1.SocioTelematics;
    };
    RolePlayerBinding b7{
        RoleName RentedCar;
        Player Car#1.SocioTelematics;
    };
    RolePlayerBinding b7{
        RoleName Coordinator;
        Player Car#1.CoordinationRules;
    };

    RoleCentricSocialRelationship Rr1{
        RelationshipName LeadingCar-Coordinator;
        RoleA LeadingCar;
        RoleB Coordinator;
        SocialInteraction i1{
            MsgSignature notifyPosition(float:latitude,float:longitude);
            Direction RoleBToRoleA;
            RelatedObligation o1, o2;
        };
        SocialInteraction i2{
            MsgSignature notifyPosition(float:latitude,float:longitude);
            Direction RoleAToRoleB;
            RelatedObligation o1;
        };
    }
}
APPENDIX C. SOCIAL INTERACTION MODELS

```java
SocialInteraction i3{  
    MsgSignature notifyRoadBlock(String address);  
    Direction RoleBToRoleA;
};
SocialInteraction i4{  
    MsgSignature routeUpdate(String route);  
    Direction RoleAToRoleB;
};
SocialInteraction i5{  
    MsgSignature notifyMechanicalIssue(String issueDescription);  
    Direction RoleAToRoleB;
};
SocialInteraction i6{  
    MsgSignature notifyMechanicalIssue(String issueDescription);  
    Direction RoleBToRoleA;
};
SocialInteraction i16{  
    MsgSignature convoyBreak(String cause);  
    Direction RoleBToRoleA;
};
SocialInteraction i17{  
    MsgSignature positiveResponse(String cause);  
    Direction RoleAToRoleB;
};
Obligation o1{  
    Timer PERIOD;  
    RelatedOperationalParam p1;
};
Obligation o2{  
    RelatedOperationalParam p2;
};
OperationalParam p1{  
    OPParamName posUpdateFreq;  
    OPParamValue 10;  
    TimeUnit SEC;
};
OperationalParam p2{  
    OPParamName maxDesiredDistance;  
    OPParamValue 1;  
    DistanceUnit KM;
};

RoleCentricSocialRelationship Rr2{  
    RelationshipName RentedCar-Coordinator;  
    RoleA RentedCar;  
    RoleB Coordinator;
}
```
Apprendix C. Social Interaction Models

SocialInteraction i7{
    MsgSignature routeRequest(String sourceAddr, String destAddr,
                               String preference);
    Direction RoleAToRoleB;
};
SocialInteraction i8{
    MsgSignature routeNotification(String route);
    Direction RoleBToRoleA;
};
SocialInteraction i9{
    MsgSignature notifyMechanicalIssue(String issueDescription);
    Direction RoleAToRoleB;
};
SocialInteraction i10{
    MsgSignature mechanicalAssistanceRequest(String issueDescription);
    Direction RoleAToRoleB;
};
SocialInteraction i11{
    MsgSignature assistanceResponse(String description);
    Direction RoleBToRoleA;
};
SocialInteraction i18{
    MsgSignature addressRequest(String placeType);
    Direction RoleAToRoleB;
};
SocialInteraction i9{
    MsgSignature responseAddress(String addresses);
    Direction RoleBToRoleA;
};
Conversation c1{
    SocialInteractionX i7;
    SocialInteractionX i8;
    Sequence leadsTo; // i7 leadsTo i8
    RelatedObligation o3;
};
Obligation o3{
    Timer DURATION;
    RelatedOperationalParam p3;
};
OperationalParam p3{
    OPParamName maxTimeDelay;
    OPParamValue 15;
    TimeUnit SEC;
};
Conversation c2{
    SocialInteractionX i10;
    SocialInteractionX i11;
}
Appendix C. Social Interaction Models

Sequence leadsTo; // i10 leadsTo i11
RelatedObligation o4;
};
Obligation o4{
    Timer DURATION;
    RelatedOperationalParam p4;
};
OperationalParam p4{
    OPParamName maxTimeDelay;
    OPParamValue 5;
    TimeUnit MIN;
};
Conversation c3{
    SocialInteractionX i12;
    SocialInteractionX i13;
    Sequence leadsTo; // i12 leadsTo i13
};
};

C.2.2 Car#2PSIM

PSIM Car#2PSIM{
    SocialRole FollowingCar, RentedCar;
    CoordinatorRole Coordinator;
    OrganiserRole Organiser;

    RolePlayerBinding org{
        RoleName Organiser;
        Player Car#2.SocioTelematics;
    }; 
    RolePlayerBinding b1{
        RoleName FollowingCar;
        Player Car#2.SocioTelematics;
    }; 
    RolePlayerBinding b2{
        RoleName RentedCar;
        Player Car#2.SocioTelematics;
    }; 
    RolePlayerBinding b3{
        RoleName Coordinator;
        Player Car#2.CoordinationRules;
    }; 

    RoleCentricSocialRelationship Rr3{
        RelationshipName FollowingCar-Coordinator;
    }
RoleA Coordinator;
RoleB FollowingCar;

SocialInteraction i1{
    MsgSignature notifyPosition(float:latitude,float:longitude);
    Direction RoleBToRoleA;
    RelatedObligation o1, o2;
};

SocialInteraction i2{
    MsgSignature notifyPosition(float:latitude,float:longitude);
    Direction RoleAToRoleB;
    RelatedObligation o1;
};

SocialInteraction i3{
    MsgSignature notifyRoadBlock(String roadAddress);
    Direction RoleBToRoleA;
};

SocialInteraction i4{
    MsgSignature routeUpdate(String route);
    Direction RoleAToRoleB;
};

SocialInteraction i5{
    MsgSignature notifyMechanicalIssue(String issueDescription);
    Direction RoleAToRoleB;
};

SocialInteraction i16{
    MsgSignature convoyBreak(String cause);
    Direction RoleBToRoleA;
};

SocialInteraction i17{
    MsgSignature positiveResponse(String cause);
    Direction RoleAToRoleB;
};

Obligation o1{
    Timer PERIOD;
    RelatedOperationalParam p1;
};

Obligation o2{
    RelatedOperationalParam p2;
};

OperationalParam p1{
    OPParamName posUpdateFreq;
    OPParamValue 10;
    TimeUnit SEC;
};

OperationalParam p2{
    OPParamName maxDesiredDistance;
    OPParamValue 1;
    DistanceUnit KM;
}
APPENDIX C. SOCIAL INTERACTION MODELS

RoleCentricSocialRelationship Rr4{
  RelationshipName RentedCar-Coordinator;
  RoleA RentedCar;
  RoleB Coordinator;
  SocialInteraction i14{
    MsgSignature roadBlockCheckingRequest(String sourceAddr, String destAddr);
    Direction RoleAToRoleB;
  }
  SocialInteraction i15{
    MsgSignature roadBlockCheckingResult(String address);
    Direction RoleBToRoleA;
  }
  SocialInteraction i16{
    MsgSignature notifyMechanicalIssue(String issueDescription);
    Direction RoleAToRoleB;
  }
  SocialInteraction i17{
    MsgSignature mechanicalAssistanceRequest(String issueDescription);
    Direction RoleAToRoleB;
  }
  SocialInteraction i18{
    MsgSignature assistanceResponse(String description);
    Direction RoleBToRoleA;
  }
  Conversation c4{
    SocialInteractionX i14;
    SocialInteractionX i15;
    Sequence leadsTo;
    RelatedObligation o6;
  }
  Obligation o6{
    Timer DURATION;
    RelatedOperationalParam p6;
  }
  OperationalParam p6{
    OPParamName maxTimeDelay;
    OPParamValue 10;
    TimeUnit SEC;
  }
  Conversation c5{
    SocialInteractionX i17;
    SocialInteractionX i18;
    Sequence leadsTo; // i17 leadsTo i18
    RelatedObligation o7;
  }
App: C. Social Interaction Models

};
Obligation o7{
    Timer DURATION;
    RelatedOperationalParam p7;
};
OperationalParam p7{
    OPParamName maxTimeDelay;
    OPParamValue 8;
    TimeUnit MIN;
};


Appendix D

Rule Translation Script

(RuleTrans.dsl)

This RuleTrans.dsl is used by the Drools engine to automatically translate situation rules written in human readable format to the Drools executable format.

[when][] an atomic interaction event {atomic_event_name}
occurred in "{source}" which is a message communication, {constraints} = ${atomic_event_name}:InteractionEvent({constraints}) from entry-point "{source}"

[when][] an atomic interaction event {atomic_event_name}
occurred which is a message communication, {constraints} = ${atomic_event_name}:InteractionEvent({constraints})

[when][] a complex interaction event {complex_event_name}
obscurred between {event1} and {event2} where event {event2}
occurred in "{source}" is a message communication, {constraints} = ${complex_event_name}:InteractionEvent({constraints}) from entry-point "{source}"

[when][] a complex interaction event {complex_event_name}
obscurred between {event1} and {event2} where event {event2} is
a message communication, {constraints} = ${complex_event_name}:InteractionEvent({constraints})

[when][] an atomic interaction event {atomic_event_name}
occurred in "{source}" = ${atomic_event_name}:InteractionEvent() from entry-point "{source}"
[when][] an atomic interaction event {atomic_event_name} occurred = ${\text{atomic_event_name}}: \text{InteractionEvent}() 

[when][] is less than or equal to<=

[when][] is less than<

[when][] is greater than or equal to>=

[when][] is greater than=>

[when][] is equal to===

[when][] equals===

[when][] from actor "{value}" = eval(kbIns.semanticMatching("fromActor","{value}"))

[when][] to actor "{value}" = eval(kbIns.semanticMatching("toActor","{value}"))

[when][] from role "{value}" = eval(kbIns.semanticMatching("fromRole","{value}"))

[when][] to role "{value}" = eval(kbIns.semanticMatching("toRole","{value}"))

[when][] related with "{value}" relationship = eval(kbIns.semanticMatching("related","{value}"))

[when][] discussing about "{value}" = eval(kbIns.semanticMatching("about","{value}"))

[when][] has occurred "before" ${\text{event2}}$ by '{value}' = this before[1ms,{'value'}] ${\text{event2}}$

[when][] has occurred "before" ${\text{event2}}$ between '{value1}' and '{value2}' = this before[{'value1'},{'value2'}] ${\text{event2}}$

[when][] has occurred "after" ${\text{event2}}$ by '{value}' = this before[1ms,{'value'}] ${\text{event2}}$

[when][] has occurred "after" ${\text{event2}}$ between '{value1}' and '{value2}' = this after[{'value1'},{'value2'}] ${\text{event2}}$

[when][] has occurred "coincides" ${\text{event2}}$ by '{value}' = this coincides[{'value'}] ${\text{event2}}$
[when][] has occurred "coincides" ${\text{event2}}$ by ‘{value1}’ and ‘{value2}’ = this coincides[{{value1},{value2}}] ${\text{event2}}$

[when][] has occurred "during" ${\text{event2}}$ by ‘{value}’ = this during[{{value}}] ${\text{event2}}$

[when][] has occurred "during" ${\text{event2}}$ by ‘{value1}’ and ‘{value2}’ = this during[{{value1},{value2}}] ${\text{event2}}$

[when][] has occurred "during" ${\text{event2}}$ by ‘{value1}’ , ‘{value2}’ , ‘{value3}’ and ‘{value4}’ = this during[{{value1},{value2},{value3},{value4}}] ${\text{event2}}$

[when][] has occurred "finishes" ${\text{event2}}$ by ‘{value}’=this finishes[{{value}}] ${\text{event2}}$

[when][] has occurred "finishedby" ${\text{event2}}$ by ‘{value}’=this finishedby[{{value}}] ${\text{event2}}$

[when][] has occurred "includes" ${\text{event2}}$ by ‘{value}’=this includes[{{value}}] ${\text{event2}}$

[when][] has occurred "includes" ${\text{event2}}$ by ‘{value1}’ and ‘{value2}’=this includes[{{value1},{value2}}] ${\text{event2}}$

[when][] has occurred "includes" ${\text{event2}}$ by ‘{value1}’ , ‘{value2}’ , ‘{value3}’ and ‘{value4}’=this includes[{{value1},{value2},{value3},{value4}}] ${\text{event2}}$

[when][] has occurred "meets" ${\text{event2}}$ by ‘{value}’=this meets[{{value}}] ${\text{event2}}$

[when][] has occurred "metby" ${\text{event2}}$ by ‘{value}’=this metby[{{value}}] ${\text{event2}}$

[when][] has occurred "starts" ${\text{event2}}$ by ‘{value}’=this starts[{{value}}] ${\text{event2}}$

[when][] has occurred "startedby" ${\text{event2}}$ by ‘{value}’=this startedby[{{value}}] ${\text{event2}}$

[when][] has occurred "overlaps" ${\text{event2}}$ by ‘{value}’=this overlaps[{{value}}] ${\text{event2}}$

[when][] has occurred "overlaps" ${\text{event2}}$ by ‘{value1}’ and ‘{value2}’=this overlaps[{{value1},{value2}}] ${\text{event2}}$
APPENDIX D. RULE TRANSLATION SCRIPT (RuleTrans.dsl)

[when][] has occurred "overlappedby" ${event2}$ by '{value}' = this overlappedby [ {value} ] ${event2}$

[when][] has occurred "overlappedby" ${event2}$ by '{value1}' and '{value2}' = this overlappedby [ {value1}, {value2} ] ${event2}$

[when][] has occurred "temp_op" ${event2}$ = this {temp_op} ${event2}$

[when][] and=&&

[when][] or=||
Appendix E

Schema Definitions

The schema definition of the deployable descriptor is given below. The schema definition is captured by the files ... These files can be visualised and navigated via suitable software tool such as Eclipse XSD Editor

E.1 DSIM Schema

```xml
<?xml version="1.0" encoding="UTF-8"?>
  <element name="DSIM">
    <complexType>
      <sequence>
        <element name="SocialRole" minOccurs="2" maxOccurs="unbounded">
          <complexType>
            <simpleContent>
              <extension base="string">
                <attribute name="id" type="string"/>
              </extension>
            </simpleContent>
          </complexType>
        </element>
        <element name="SocialRelationship" minOccurs="1" maxOccurs="unbounded">
          <complexType>
            <sequence>
              <element name="RoleA" type="string" minOccurs="1" maxOccurs="1"/>
            </sequence>
          </complexType>
        </element>
      </sequence>
    </complexType>
  </element>
</schema>
```

APPENDIX E. SCHEMA DEFINITIONS

<element name="RoleB" type="string" minOccurs="1" maxOccurs="1">
</element>
<element name="SocialInteraction" minOccurs="1" maxOccurs="unbounded">
<complexType>
<sequence>
<element name="MsgName" type="string" minOccurs="1" maxOccurs="1">
</element>
<element name="Direction" type="dsim:DirectionType" minOccurs="1" maxOccurs="1">
</element>
<element name="Type" type="string" minOccurs="0" maxOccurs="unbounded">
</element>
</sequence>
<attribute name="id" type="string">
</attribute>
</complexType>
</element>
<element name="OperationalParam" minOccurs="0" maxOccurs="unbounded">
<complexType>
<sequence>
<element name="ParamName" type="string" minOccurs="1" maxOccurs="1">
</element>
<element name="ParamValue" type="string" minOccurs="1" maxOccurs="1">
</element>
<element name="TimeUnit" minOccurs="0" maxOccurs="1">
</element>
</sequence>
<complexType>
<restriction base="dsim:TimeUnitType">
<enumeration value="MS">
</enumeration>
<enumeration value="SEC">
</enumeration>
<enumeration value="MIN">
</enumeration>
<enumeration value="HOUR">
</enumeration>
<enumeration value="DAY">
</enumeration>
</restriction>
</complexType>
</element>
<element name="DistanceUnit" minOccurs="0" maxOccurs="1">
<complexType>
<restriction base="string">
</restriction>
</complexType>
</element>
<enumeration value="MM"></enumeration>
<enumeration value="CM"></enumeration>
<enumeration value="M"></enumeration>
<enumeration value="KM"></enumeration>
</restriction>
</simpleType>
</element>
</sequence>
<attribute name="id" type="string"></attribute>
</complexType>
</element>
<element name="Obligation" minOccurs="0" maxOccurs="unbounded">
<complexType>
<sequence>
<element name="Timer" minOccurs="1" maxOccurs="1" type="dsm:TimerType"></element>
<element name="RelatedOPParameter" type="string" minOccurs="1" maxOccurs="1"></element>
</sequence>
<attribute name="id" type="string"></attribute>
</complexType>
</element>
<element name="Conversation" minOccurs="0" maxOccurs="unbounded">
<complexType>
<sequence>
<element name="SocialInteractionX" type="string" minOccurs="1" maxOccurs="1"></element>
<element name="SocialInteractionY" type="string" minOccurs="1" maxOccurs="1"></element>
<element name="SequenceType" minOccurs="1" maxOccurs="1"></element>
<simpleType>
<restriction base="string">
<enumeration value="leadsTo"></enumeration>
<enumeration value="precedes"></enumeration>
</restriction>
</simpleType>
<element name="RelatedObligation" type="string" minOccurs="1" maxOccurs="1">
</element>
</sequence>
<attribute name="id" type="string">
</attribute>
</complexType>
</element>

<element name="OrganiserRole" type="string" minOccurs="1" maxOccurs="1">
</element>
<element name="RolePlayerBinding" minOccurs="1" maxOccurs="unbounded">
<complexType>
<sequence>
<element name="RoleName" type="string"></element>
<element name="Player" type="string"></element>
</sequence>
</complexType></element>
</sequence>
<attribute name="id" type="string"></attribute>
<attribute name="name" type="string" use="required"></attribute>
</complexType></element>

<simpleType name="DirectionType">
<restriction base="string">
<enumeration value="RoleAToRoleB"></enumeration>
<enumeration value="RoleBToRoleA"></enumeration>
</restriction>
</simpleType>

<simpleType name="TimeUnitType">
<restriction base="string">
<enumeration value="MS"></enumeration>
<enumeration value="SEC"></enumeration>
<enumeration value="MIN"></enumeration>
<enumeration value="HOUR"></enumeration>
<enumeration value="DAY"></enumeration>
</restriction>
</simpleType>

<simpleType name="TimerType">
<restriction base="string">
<enumeration value="DURATION"></enumeration>
<enumeration value="PERIOD"></enumeration>
</restriction>
</simpleType>
E.2 PSIM Schema

<?xml version = "1.0" encoding = "UTF-8"?>

<element name="PSIM">
  <complexType>
    <sequence>
      <element name="SocialRole" minOccurs="1" maxOccurs="unbounded">
        <complexType>
          <simpleContent>
            <extension base="string">
              <attribute name="id" type="string"/>
            </extension>
          </simpleContent>
        </complexType>
      </element>
      <element name="CoordinatorRole" minOccurs="1" maxOccurs="1">
        <complexType>
          <simpleContent>
            <extension base="string">
              <attribute name="id" type="string"/>
            </extension>
          </simpleContent>
        </complexType>
      </element>
      <element name="RoleCentricSocialRelationship" minOccurs="1" maxOccurs="unbounded">
        <complexType>
          <sequence>
            <element name="RoleA" type="string" minOccurs="1" maxOccurs="1"/>
            <element name="RoleB" type="string" minOccurs="1" maxOccurs="1"/>
            <element name="SocialInteraction" minOccurs="1" maxOccurs="unbounded">
              <complexType>
                <sequence>
                  <element name="MsgName" type="string" minOccurs="1" maxOccurs="1"/>
                  <element name="Direction" type="dsim:DirectionType" minOccurs="1" maxOccurs="1"/>
                  <element name="Type" type="string" minOccurs="0" maxOccurs="unbounded"/>
                </sequence>
                <attribute name="id" type="string"/>
              </complexType>
            </element>
          </sequence>
        </complexType>
      </element>
    </sequence>
  </complexType>
</element>
</schema>
APPENDIX E. SCHEMA DEFINITIONS

minOccurs="0" maxOccurs="unbounded">
<complexType>
<sequence>
<element name="ParamName"
    type="string" minOccurs="1" maxOccurs="1">
</element>
<element name="ParamValue"
    type="string" minOccurs="1" maxOccurs="1">
</element>
<element name="TimeUnit"
    minOccurs="0" maxOccurs="1">
<complexType>
<restriction base="dsim:TimeUnitType">
<enumeration value="MS"/>
<enumeration value="SEC"/>
<enumeration value="MIN"/>
<enumeration value="HOUR"/>
<enumeration value="DAY"/>
</restriction>
</complexType>
</element>
<element name="DistanceUnit"
    minOccurs="0" maxOccurs="1">
<complexType>
<restriction base="string">
<enumeration value="MM"/>
<enumeration value="CM"/>
<enumeration value="M"/>
<enumeration value="KM"/>
</restriction>
</complexType>
</element>
</sequence>
<attribute name="id" type="string"/>
</complexType>
</element>
<element name="Obligation" minOccurs="0" maxOccurs="unbounded">
<complexType>
<sequence>
  <element name="Timer"
    minOccurs="1" maxOccurs="1" type="dsim:TimerType">
  </element>
</element>
<element
  name="RelatedOPParameter" type="string" minOccurs="1"
  maxOccurs="1">
</element>
</sequence>
<attribute name="id" type="string">
</attribute>
</complexType>
</element>
<element name="Conversation" minOccurs="0" maxOccurs="unbounded">
<complexType>
<sequence>
  <element name="SocialInteractionX" type="string" minOccurs="1"
    maxOccurs="1">
  </element>
  <element name="SocialInteractionY" type="string" minOccurs="1"
    maxOccurs="1">
  </element>
  <element name="SequenceType"
    minOccurs="1" maxOccurs="1">
    <simpleType>
      <restriction base="string">
        <enumeration value="leadsTo">
          </enumeration>
        <enumeration value="precedes">
          </enumeration>
      </restriction>
    </simpleType>
  </element>
  <element name="RelatedObligation" type="string" minOccurs="1"
    maxOccurs="1">
  </element>
</sequence>
<attribute name="id" type="string"></attribute>
</complexType>
</element>
<element name="OrganiserRole" type="string"
  minOccurs="1" maxOccurs="1">
</element>
<element name="RolePlayerBinding" minOccurs="1" maxOccurs="unbounded">
<complexType>
<sequence>
  <element name="RoleName" type="string"></element>
  <element name="Player" type="string"></element>
</sequence>
</complexType>
</element>

<complexType name="DirectionType">
  <restriction base="string">
    <enumeration value="RoleAToRoleB"/>
    <enumeration value="RoleBToRoleA"/>
  </restriction>
</complexType>

<complexType name="TimeUnitType">
  <restriction base="string">
    <enumeration value="MS"/>
    <enumeration value="SEC"/>
    <enumeration value="MIN"/>
    <enumeration value="HOUR"/>
    <enumeration value="DAY"/>
  </restriction>
</complexType>

<complexType name="TimerType">
  <restriction base="string">
    <enumeration value="DURATION"/>
    <enumeration value="PERIOD"/>
  </restriction>
</complexType>