A Framework and Coordination Technologies for Peer-to-peer based Decentralised Workflow Systems

by

Jun Yan

B.Eng. (SEU)
M.Eng. (SEU)

A thesis submitted to
School of Information Technology
Swinburne University of Technology

for the degree of
Doctor of Philosophy

August, 2004
To my wife, Xiaohong, my son, Benjamin, and my parent
Abstract

This thesis investigates an innovative framework and process coordination technologies for peer-to-peer based decentralised workflow systems. The aim of this work is to address some of the unsolved problems in the contemporary workflow research rudimentally from an architectural viewpoint. The problems addressed in this thesis, i.e., bad performance, vulnerability to failures, poor scalability, user restrictions, unsatisfactory system openness, and lack of support for incompletely-specified processes, have become major obstacles for wide deployment of workflow in real-world. After an in-depth analysis of the above problems, this thesis reveals that most of these problems are mainly caused by the mismatch between application nature, i.e., distributed, and system design, i.e., centralised management. Thus, the old-fashioned client-server paradigm which is conventionally used in most of today’s workflow systems should be replaced with a peer-to-peer based, open, collaborative and decentralised framework which can reflect workflow’s distributed feature more naturally.

Combining workflow technology and peer-to-peer computing technology, SwinDeW which is a genuinely decentralised workflow approach is proposed in this thesis. The distinguished design of SwinDeW removes both the centralised data repository and the centralised workflow engine from the system. Hence, workflow participants are facilitated by automated peers which are able to communicate and collaborate with one another directly to fulfil both build-time and run-time workflow functions. To achieve this goal, an innovative data storage approach, known as “know what you should know”, is proposed, which divides a process model into individual task partitions and distributes each partition to relevant peers properly according to the capability match. Based on such a data storage approach, the novel
mechanisms for decentralised process instantiation, instance execution and execution monitoring are explored. Moreover, SwinDeW is further extended to support incompletely-specified processes in the decentralised environment. New technologies for handling incompletely-specified processes at run-time are presented.

The major contributions of this research are an innovative, decentralised workflow system framework and corresponding process coordination technologies for system functionality. Issues regarding system performance, reliability, scalability, user support, system openness, and incompletely-specified process support are discussed deeply. Moreover, this thesis also contributes the SwinDeW prototype which implements and demonstrates this design and functionality for proof-of-concept purposes. With these outcomes, performance bottlenecks in workflow systems are likely to be eliminated whilst increased resilience to failure, enhanced scalability, better user support and improved system openness are likely to be achieved with support for both completely- and incompletely-specified processes. As a consequence, workflow systems will be expected to be widely deployable to real world applications to support processes, which was infeasible before.
Acknowledgements

I sincerely express my deepest gratitude to my supervisor, Associate Professor Yun Yang, for his seasoned supervision and continuous encouragement throughout the course of this work, and for his careful reading and appraisal of drafts of this thesis. Without his consistent support, I would not have been able to complete this manuscript. I also thank my associate supervisor, Dr. Gitesh K. Raikundalia, for his valuable advice, suggestions and support.

I thank Swinburne University of Technology and the School of Information Technology for offering me a full Research Scholarship throughout my doctoral program. I also thank the Research Committee of the School of Information Technology for research publication funding support and for providing me with financial support to attend conferences.

My thanks also go to staff members, research students and research assistants at CICEC for their help, suggestions, friendship and encouragement, in particular, Neroli, Finlay, Phil Joyce, Wei Lai, Jun Shen, Boris Wu, Lukman Setiawan, and Jonathan Derham.

Last but not least, I am deeply grateful to my wife, Xiaohong, my son, Benjamin, my parents, and my parents-in-law for their love, understanding, patience, encouragement, sacrifice and help.
Declaration

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 thesis. To the best of my knowledge, this thesis contains no material previously published or written by another person except where due reference is made in the text of the thesis.

Jun Yan

30 August, 2004
# Table of Contents

## Chapter 1 Introduction
1.1 Introduction to workflow management .......................................................... 1  
1.2 Key issues of this research .............................................................................. 4  
1.3 Overview of this thesis .................................................................................... 6

## Chapter 2 Literature review and requirements analysis
2.1 Research problems analysis ............................................................................ 8  
2.2 The WfMC’s workflow reference model .......................................................... 12  
2.3 Workflow approaches under aspects of client-server based distribution ..... 14  

2.3.1 Exotica/FMQM .......................................................................................... 14  
2.3.2 ADEPT .................................................................................................... 15  
2.3.3 Endeavors ............................................................................................... 16  
2.3.4 DartFlow ................................................................................................ 17  
2.3.5 METUFlow ............................................................................................ 18  
2.3.6 Discussion ............................................................................................... 18  

2.4 Decentralised workflow approaches based on other computing technologies  

2.4.1 PeCo ....................................................................................................... 19  
2.4.2 An architecture based on WWPD and WWP ......................................... 20  
2.4.3 RainMan .................................................................................................. 20  
2.4.4 Serendipity-II .......................................................................................... 21  
2.4.5 Matrix .................................................................................................... 21  
2.4.6 Discussion ............................................................................................... 22  

2.5 Workflow research related to incomplete process support  

2.5.1 Workflow Continuum ............................................................................... 23  
2.5.2 WASA ..................................................................................................... 25  
2.5.3 WORKWARE ........................................................................................... 25  
2.5.4 Pockets of Flexibility ............................................................................... 26
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.5.5 Discussion</td>
<td>27</td>
</tr>
<tr>
<td>2.6 Requirements analysis</td>
<td>27</td>
</tr>
<tr>
<td>2.7 Summary</td>
<td>29</td>
</tr>
<tr>
<td>Chapter 3 System framework and design</td>
<td>30</td>
</tr>
<tr>
<td>3.1 Peer-to-peer technology</td>
<td>30</td>
</tr>
<tr>
<td>3.2 Decentralised workflow architecture based on p2p</td>
<td>32</td>
</tr>
<tr>
<td>3.3 Peer structure and peer organisation</td>
<td>37</td>
</tr>
<tr>
<td>3.3.1 Peer structure</td>
<td>37</td>
</tr>
<tr>
<td>3.3.2 Peer organisation</td>
<td>41</td>
</tr>
<tr>
<td>3.4 Peer management service</td>
<td>43</td>
</tr>
<tr>
<td>3.4.1 Peer discovery service</td>
<td>43</td>
</tr>
<tr>
<td>3.4.2 Peer registration service</td>
<td>45</td>
</tr>
<tr>
<td>3.4.3 Capability configuration service</td>
<td>49</td>
</tr>
<tr>
<td>3.5 System initialisation</td>
<td>50</td>
</tr>
<tr>
<td>3.6 Summary</td>
<td>51</td>
</tr>
<tr>
<td>Chapter 4 Build-time functions</td>
<td>52</td>
</tr>
<tr>
<td>4.1 Issues of process representation</td>
<td>52</td>
</tr>
<tr>
<td>4.2 A graph-based process representation</td>
<td>55</td>
</tr>
<tr>
<td>4.3 Task notation</td>
<td>62</td>
</tr>
<tr>
<td>4.4 Process definition service</td>
<td>64</td>
</tr>
<tr>
<td>4.4.1 Partitions of a process</td>
<td>66</td>
</tr>
<tr>
<td>4.4.2 Distribution of data partition</td>
<td>68</td>
</tr>
<tr>
<td>4.4.3 Data maintenance</td>
<td>70</td>
</tr>
<tr>
<td>4.5 Summary</td>
<td>71</td>
</tr>
<tr>
<td>Chapter 5 Run-time functions</td>
<td>73</td>
</tr>
<tr>
<td>5.1 Process instantiation procedure</td>
<td>73</td>
</tr>
<tr>
<td>5.2 Task instantiation</td>
<td>77</td>
</tr>
<tr>
<td>5.3 Dynamic work allocation</td>
<td>80</td>
</tr>
<tr>
<td>5.3.1 Allocation policy based on round-robin scheduling</td>
<td>81</td>
</tr>
<tr>
<td>5.3.2 Allocation policy based on workload</td>
<td>81</td>
</tr>
<tr>
<td>5.3.3 Allocation policy based on weighted workload</td>
<td>85</td>
</tr>
</tbody>
</table>
Chapter 6  Extending SwinDeW for incomplete process support .......... 105
6.1 Causes of incomplete processes .................................................. 105
6.2 Requirements analysis ................................................................. 108
6.3 Scope of incomplete process support ........................................... 109
6.4 Multi-level process modelling and execution ............................... 112
   6.4.1 Hierarchical process modelling .............................................. 113
   6.4.2 Existing difficulties for supporting incomplete composite tasks ... 115
   6.4.3 Introducing the decomposition task ....................................... 116
   6.4.4 Performing the decomposition task ...................................... 121
   6.4.5 Instance execution .............................................................. 125
   6.4.6 Inconsistency handling ........................................................ 128
   6.4.7 Elaboration of incomplete atomic tasks ................................. 130
6.5 Summary .................................................................................... 134

Chapter 7  Case studies ................................................................. 135
7.1 Case study 1: student registration service ..................................... 135
   7.1.1 Introduction .......................................................................... 135
   7.1.2 System settings ................................................................. 138
   7.1.3 System support for build-time functions ............................... 141
7.1.4 System support for run-time functions............................................ 142
7.1.5 Summary of this section................................................................. 145
7.2 Case study 2: management of a research project ............................... 145
  7.2.1 Introduction..................................................................................... 146
  7.2.2 System support for the incomplete project...................................... 148
  7.2.3 Summary of this section................................................................. 151
7.3 Summary ............................................................................................. 151

Chapter 8 Prototype implementation............................................................. 152
  8.1 JXTA peer-to-peer framework............................................................. 152
  8.2 Overall prototype framework............................................................. 154
  8.3 Overview of SwinDeW functionality................................................... 157
  8.4 Implementation of key system components....................................... 158
    8.4.1 Process definition tool................................................................. 158
    8.4.2 Implementation of peer organisation.......................................... 161
    8.4.3 Implementation of Peer Kernel .................................................. 164
    8.4.4 Implementation of incomplete process support......................... 167
    8.4.5 Implementation of local visualisation module ......................... 169
  8.5 Summary ............................................................................................. 172

Chapter 9 Discussions..................................................................................... 173
  9.1 Discussion on the advantages of this research................................... 173
  9.2 Discussion on the possible Tradeoffs of the proposed approach......... 176
  9.3 Discussion on suitable application domains of SwinDeW............... 179
  9.4 Summary ............................................................................................. 179

Chapter 10 Conclusions and future work...................................................... 181
  10.1 Summary of this thesis................................................................. 181
  10.2 Contributions of this thesis............................................................ 183
  10.3 Future work..................................................................................... 187

Bibliography ............................................................................................. 189

The author’s publications........................................................................... 206
List of Figures

Figure 2.1 WfMC’s Workflow Reference Model ..................................................... 13
Figure 2.2 Workflow Continuum ([NH94], all rights reserved) .............................. 24
Figure 3.1 Client-server model vs. p2p model ....................................................... 31
Figure 3.2 Decentralised system architecture of SwinDeW ..................................... 33
Figure 3.3 Structure of a peer in SwinDeW ............................................................ 38
Figure 3.4 Initial virtual communities in SwinDeW ................................................. 42
Figure 3.5 Possible information loss when a peer leaves the system ....................... 48
Figure 4.1 Constructs of process representation ..................................................... 61
Figure 4.2 Visualised representation of a workflow process .................................... 61
Figure 4.3 A task with its context in a process ....................................................... 63
Figure 4.4 Distribution of a task partition ............................................................. 68
Figure 5.1 Example of process instantiation procedure .......................................... 75
Figure 5.2 Task instantiation in a sequential structure .......................................... 78
Figure 5.3 Task instance state transition diagram .................................................. 92
Figure 5.4 “Unavailable peer” exception in a sequential structure ............................ 99
Figure 6.1 Hierarchical process modelling ............................................................ 113
Figure 6.2 A decomposition task and the associated composite task ....................... 117
Figure 6.3 Visualised representation of a composite task ....................................... 120
Figure 6.4 Automated decomposition of controlled composite tasks ..................... 123
Figure 6.5 Conversion of the peer network for the present instance ....................... 125
Figure 6.6 Formation of composite tasks for incomplete atomic tasks ................... 131
Figure 7.1 A typical student registration process................................. 137
Figure 7.2 Virtual communities of peers at the time when the system is initialised
......................................................................................................................... 140
Figure 7.3 Different peer networks for two process instances............... 142
Figure 7.4 Part of CICEC research process........................................... 149
Figure 7.5 Sub-process of workflow research........................................ 150

Figure 8.1 Framework of SwinDeW prototype................................. 154
Figure 8.2 Process definition tool illustrating a complete process............. 159
Figure 8.3 Part of an XML file of a workflow process defined in XPDL ...... 161
Figure 8.4 Capability configuration of a peer ........................................ 162
Figure 8.5 Implementation of peer organisation.................................... 163
Figure 8.6 Description of the Peer Kernel............................................. 165
Figure 8.7 Process definition tool illustrating an incomplete process.......... 167
Figure 8.8 Run-time support for incomplete processes ............................ 168
Figure 8.9 Run-time modelling of the a composite task............................ 169
Figure 8.10 Implementation of local visualisation module...................... 170
Figure 8.11 Visualised representation of a peer’s process definition ......... 171
Figure 8.12 Visualised representation of a peer’s task instances.............. 171
List of Tables

Table 4.1 Summary of control routing ................................................................. 58

Table 5.1 Types of control conditions among tasks for starting task instance....... 93
Table 5.2 Types of data and control flows for completing task instance............... 94

Table 7.1 Summary of workflow participants and their capabilities in a student registration service ........................................................................................................... 139
Table 7.2 Summary of initial peer repositories, process repositories, and task repositories for two individual process instances......................................................... 140

Table 8.1 Summary of functionality in SwinDeW prototype ............................... 157
Chapter 1

Introduction

This thesis addresses the limitations of the conventional workflow management systems based on the dominating client-server distributed system architecture. The novel research reported in this thesis is concerned with the investigation into a new framework and process coordination technologies for decentralised, dynamic workflow systems based on peer-to-peer (p2p), rather than client-server, distributed system architecture, which have been recognised as one of the most strategic future directions for workflow research [Mas02]. An innovative workflow management system based on p2p, known as Swinburne Decentralised Workflow (SwinDeW), is presented in this thesis. The corresponding system mechanisms for both completely-specified (complete for short) and incompletely-specified (incomplete for short) processes are designed. Moreover, a system prototype based on Sun Microsystems JXTA is also developed for demonstration and proof-of-concept purposes.

This chapter introduces the background, motivations and key issues of this research. First, a brief introduction to workflow management is given in Section 1.1. Then, Section 1.2 outlines the key issues of this research. Finally, Section 1.3 presents an overview of the remainder of this thesis.

1.1 Introduction to workflow management

At the heart of any organisations is a more-or-less formalised set of processes, which refer to the unique manner in which organisations coordinate and organise work
activities, information and knowledge to produce a product or service [LL02]. Typical examples of processes include loan application processing, insurance claim processing, recruitment of an employee, and so on. It is widely agreed that the adequate support for processes has become paramount to the agility and success of the organisation as a whole [BUMR99]. Over the past decades of process support research, paradigms have evolved from (hard-wired) office automation systems to workflow systems. In this Internet era, Workflow Management (WfM), as an enabling technology for Business Process Management (BPM), is increasingly being exploited by a variety of organisations and becoming an important part of organisational information systems.

In short, a workflow represents the operational aspects of a process, which include the order of tasks and who perform them, the information flow to support tasks, and the tracking and reporting mechanisms that measure and control them. Formerly, workflow is defined as “the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules to achieve, or contribute to, an overall business goal” [WfMC99]. Whilst workflow may be manually organised, in practice, most workflow is better organised within the context of an Information Technology system to provide computerised support for the process automation. Such computer-based systems, known as Workflow Management Systems (WfMSs), are designed to improve processes by providing the technology enabler for automating the following aspects of the workflow [WfMC99]:

- routing work in the proper sequence,
- providing access to the data and documents required by the individual work performers, and
- tracking all aspects of the process execution

To achieve this, a WfMS “consists of software components to store and interpret process definitions, create and manage workflow instances as they are executed, and
control their interaction with workflow participants and applications” [WfMC99]. At the highest level, all WfMSs may be characterised as providing support in two functional areas:

- the build-time functions, concerned with defining and representing the workflow process and its constituent tasks, and data storage issues, and

- the run-time functions, concerned with creating and managing the workflow instances in an operational environment.

Arguably, workflow management originates in office automation for document-oriented processes such as approval processes in the banking and insurance domains in late 1980s [PS84]. Since then, workflow management systems have quickly evolved from manual routing of folders and image-based document-centric systems, into general-purpose mainstream middleware [CCPP96, Sch99]. Nowadays, workflow management systems have become an essential asset of organisations, as they are capable of integrating various kinds of resources, such as heterogeneous systems, existing applications, human beings, and so on [Moh98a, Sch99]. It is observed that the proper use of workflow management could help to streamline formerly inefficient procedures, reduce costs and flow times, and increase quality of service and productivity, thus yielding competitive advantages. As a result, the power of workflow is one of the keys to significantly improve performance of an organisation.

In both the research and practice areas, workflow management has undoubtedly been one of the most important domains of interests since its advent. A number of conferences, workshops and symposia have been organised for researchers to discuss a wide range of topics in the workflow area, such as architecture, modelling, evaluation, interoperability, etc. Various theoretical approaches and research prototypes have been presented and numerous contributions have been published [e.g., GHS95, JB96, CCPP96, Moh98b, AH02, Fis02]. At the same time, empirical workflow practice is also thriving. A vast variety of commercial workflow

In addition, great efforts have been placed to push workflow towards standardisation. Workflow Management Coalition (WfMC, http://www.wfmc.org), founded in 1993, is a non-profit, international organisation of workflow vendors, users, analysts and research groups whose mission is to promote the use of workflow through the establishment of standards for software terminology, interoperability and connectivity between workflow products. Since 1993, the coalition has quickly become established as the primary standards body for this rapidly expanding software market. A workflow reference model and a series of standards, which have been widely accepted today, have been released to minimise the risk of developing workflow management systems. Another standardisation effort is made by Object Management Group (OMG, http://www.omg.org), who published its workflow management facility specification in 2000. This specification relates to the OMG-compliant interfaces for workflow management systems in order to use CORBA and relevant technologies to implement workflow systems.

1.2 Key issues of this research

Although workflow research and practice have reached a certain degree of maturity, some limitations of the existing approaches have been witnessed. The state-of-the-art in workflow management has so far been determined by the functionality provided in workflow systems [AAAM97]. Issues such as system performance, reliability, scalability, user support, system openness and incomplete process support are hardly ever considered sufficiently in the development of existing workflow systems [GHS95, AS96, GAHM98, Yan02a]. Thus, workflow systems often suffer from deficiencies in these areas, such as bad performance, vulnerability to failures, poor scalability, user restrictions, unsatisfactory system openness, and lack of
incomplete process support. To address some of these problem areas, some significant work has been done [e.g., AMGA+95, GACT+97, HKBT98, JSHH+01].

This thesis aims at addressing the above problems fundamentally from a system architectural point of view. As many researchers have pointed out, most of the above problems are mainly, if not completely, caused by the inadequate adoption of client-server architecture in most of today’s workflow management systems [GAHN98, Coo02]. Hence, to address these problems properly, the centralised framework based on client-server is expected to be replaced by an open, collaborative, and decentralised framework, which can reflect workflow’s increasingly distributed features more naturally. Although some work has been done in this area [e.g., PPC97, FK04], this thesis proposes some new solutions to add to and improve on the existing approaches.

A key element of this thesis is to investigate a new framework and process coordination technologies for p2p, rather than client-server, based decentralised workflow systems. This research work mainly focuses on the first aspect of workflow as indicated in Section 1.1, i.e., routing work in the proper sequence in a decentralised environment through coordination. In particular, this thesis starts with the investigation of process coordination technologies for complete processes in a p2p setting. The proposed approaches mainly address the problems related to performance, reliability, scalability, user support, and system openness. Then this research is extended to investigate technologies in support of incomplete processes under the umbrella of the proposed p2p decentralised framework. The proposed approaches are expected to be helpful for adoption of workflow systems in some new, non-traditional application domains.

The significance of this research is seen in its aim to provide a better solution to the existing problems as described above. This is regarded as a paradigm shift because the p2p technology provides a new platform for more natural process support solutions. The proposed approaches are expected to be widely deployable to real world applications to support processes, which was infeasible before.
1.3 Overview of this thesis

In particular, this thesis is dealing with the design of an innovative, p2p-based, decentralised and dynamic workflow framework and corresponding novel technologies for build-time and run-time workflow functions in support of both complete and incomplete workflow processes.

In Chapter 2, the research problems are analysed and discussed in detail. The major related work is then reviewed, including the WfMC’s workflow reference model, workflow management approaches under aspects of client-server based distribution, decentralised workflow approaches based on other computing technologies, and research regarding incomplete process support. Chapter 2 also analyses the requirements for workflow management based on p2p architecture.

Based on the requirements analysis in Chapter 2, the design of an innovative, decentralised workflow system—SwinDeW—is presented in Chapter 3. This system consists of a set of peers which are on behalf of various workflow participants. The peer structure and the peer organisation are described. The initialisation of the system is also addressed. In addition, Chapter 3 discusses system dynamism due to peer’s dynamic behaviours and how SwinDeW provides enhanced system scalability.

Chapter 4 presents the novel technologies for workflow build-time functions. An innovative, decentralised data storage approach, known as “know what you should know”, is presented. This approach divides a process definition into individual task partitions and then distributes each task partition to relevant peers. To realise this approach, a graph-based workflow representation and a new task notation are given. The former identifies key components of a workflow process and defines the operations to specify workflow processes. The latter represents a task within the context of a process. Then, the mechanisms for data partition, task distribution and data maintenance are presented.

Chapter 5 presents the novel technologies for workflow run-time functions, including process instantiation, instance execution and execution monitoring. All
these functions are carried out collaboratively through direct communication and coordination between relevant peers. The functional operations for task instances and the communication messages are also described.

Chapter 6 further extends SwinDeW to support incomplete processes. Novel technologies for handling incomplete processes incrementally at run-time are presented. The innovative task decomposition mechanisms for multi-level process modelling and execution are described.

Base on the system design in Chapter 3 and system mechanisms in Chapters 4, 5 and 6. Two case studies are presented in Chapter 7 for the purpose of demonstration. The first case discusses a student registration service, which represents a traditional, complete workflow. The second case relates to the management of a research project, which is treated as a non-traditional application domain where incomplete process support is illustrated in this case.

In Chapter 8, the issues of prototype implementation are addressed. The prototype is built on top of the Sun Microsystems JXTA infrastructure for proof-of-concept.

Chapter 9 discusses the pros and cons, and the suitable application domains of the p2p-based decentralised workflow management system presented in this thesis. The advantages of this research and the potential tradeoffs of the proposed approaches are discussed in this chapter. Moreover, suitable application domains of the proposed approach are also analysed.

The last chapter, Chapter 10, summarises the new ideas discussed in this thesis, the major contributions of this research, and the consequent further research goals.
Chapter 2

Literature review and requirements analysis

In this chapter, first, an in-depth analysis of the research problems is given in Section 2.1. Then Section 2.2 introduces major related work with respect to these research problems. In particular, Section 2.2.1 introduces the WfMC’s workflow reference model. Section 2.2.2 introduces conventional workflow approaches under aspects of client-server based distribution. Section 2.2.3 introduces some initial research on decentralised workflow based on other computing technologies. Section 2.2.4 introduces research work related to incomplete process support. Finally, based on the problem analysis and the discussion of the related work, it is pointed out that p2p-based decentralised workflow system is the most fruitful line to follow. The requirements for designing a p2p-based decentralised workflow system in support of both complete and incomplete workflow processes are analysed carefully.

2.1 Research problems analysis

As briefly discussed in Chapter 1, there are some problems which remain unsolved in today’s workflow research. In this thesis, these problems are analysed and addressed in two groups. The problems in the first group are more or less related to centralised system architecture, i.e., bad performance, vulnerability to failures, poor scalability, user restrictions, and unsatisfactory system openness. The only problem in the second group is related to flexibility, i.e., lack of incomplete process support.
It is argued in this thesis that the problems in the first group are caused by the centralised architecture traditionally used in most of today’s workflow systems. To understand this better, it is necessary to identify the gap between the inherent workflow feature and the existing solutions. Workflow processes within organisations often involve a large number of resources, people and tools distributed over a wide geographic area. Workflow management systems are used to automate the coordination of these diverse elements. Therefore, given the nature of the application environment and the technology involved, workflow applications are inherently distributed [AMGA+95, GAHM98, Yan00]. However, almost all the current workflow systems are based on the centralised client-server architecture, in which a dedicated server provides most of the functionality. The popular adoption of the client-server architecture is understandable because it has been a dominating system architecture to support distributed applications. As a proven technology, client-server technology is able to satisfy functional aspects of workflow like process management and coordination. Moreover, client-server technology also offers benefits such as thin clients, centralised monitoring and auditing, simple synchronisation mechanisms, one copy of process state, and ease of design, implementation and deployment for workflow systems. However, because of its inherent feature, i.e., centralised management, client-server technology is not the ideal supporting technology for distributed workflows in some application domains because it encounters some difficulties to satisfy non-functional aspects of workflow like performance and scalability, as detailed as follows.

Obviously, bad performance, vulnerability to failures and poor scalability are some of the severe weaknesses caused by the client-server architecture. This is because of the following reasons:

- A client-server based workflow management system merely relies on a centralised data repository for data storage and a centralised workflow engine for execution coordination while the computing potential at the client sides is barely used. Such a workflow system is heavy-weight. When the system load increases (for example, many workflow instances are executed at the same time), the centralised server (workflow server) may be
overloaded with heavy computation and intense communication, thereby becoming a potential bottleneck. Hence, system performance may be largely degraded under this circumstance.

- A workflow system based on the client-server architecture lacks robustness and is always vulnerable to server failures. In this regards, the centralised server is normally viewed as a single point of failure in the system. The whole system always becomes “dead” when the server is unavailable due to hardware or software failures. Although the primary-secondary server approach can be used to improve the reliability, it inevitably increases the complexity of the already sophisticated workflow systems.

- A workflow system based on the client-server architecture offers limited scalability due to the potential bottleneck caused by the centralised server. As a result, lack of scalability prevents a workflow system from coping with the ever-changing workflow environment. It also raises difficulties in system configuration, as any change to the system, e.g., joining of new participants, requires modifying and updating the centralised workflow server, which is very inconvenient and inefficient.

Another severe limitation caused by the client-server architecture is user restrictions. It has been widely recognised that human beings are centred on the computing environment so that human support is a significant pacing factor in any process-centred environments [Per93, Per94, WR93, Red93]. Therefore, the effectiveness of a workflow management system largely relies on the effectiveness of workflow participants involved. An essential and critical element of any workflow systems is empowering participants to maintain autonomy and control [AL03]. However, most of client-server based workflow management systems coordinate human beings in the same way they coordinate software tools. It is observed that workflow participants are highly restricted, rather than well supported, in such systems. This can be reflected by the fact that:
ordinary participants (everyday people) on the client sides have no control over data as the data is stored centrally on the server side,

ordinary participants on the client sides have no decision-making and negotiation power, as they are under full control of the dedicated workflow engine, and

the prevailing messaging constructs are based on the client-server methodology of request and reply. This mandates bilateral communication between any two workflow participants. The communication cannot flow directly between workflow participants. Therefore, the computer-supported inter-human communication is rather weak.

Because of these restrictions, workflow participants normally participate in the operations passively in a client-server based workflow system. Users’ intentions for autonomy are largely ignored. Consequently, human resources of a workflow system can be greatly underutilised.

Moreover, as Web services and Grid services become the reference model for business resources, workflow can provide a powerful framework for composing individual services into complete solutions. Concepts such as service-oriented workflow [PFW03], Web service workflow [Gan01] and Grid Workflow [DBGK03] are receiving more and more attention. However, the client-server architecture is not suitable for such areas where workflow technology is used in conjunction with services. This is because the client-server architecture is rather closed to facilitate external (Web) services available on the Internet. Thus, it is better to have an open model which allows external services to be used [Mas02].

Besides the above problems caused by a centralised architecture, lack of ability to support incomplete processes is also a major concern. Workflow research is initially based on two assumptions. First, a workflow process is modelled completely at build-time before the execution of workflow instances. Second, the instances of a workflow process remain unchanged during the execution time. These
two assumptions are justifiable initially because workflow technology is traditionally used in domains which are characterised by pre-determined, routine-based processes at that time. These so-called “production” or “prescriptive” processes are functionally predictable and repetitive. Later on, the latter assumption is proven to be unsound, with the observation that workflow processes are subject to both inside and outside changes [AJ00]. As a result, issues of dynamic workflow change, exception handling and workflow adaptation, as some of the today’s major research topics, have been addressed greatly [AJ00, HA00, LGS02]. More recently, the former assumption that workflow processes are always modelled completely at build-time is also at risk in support of processes [Wes99, SSO01]. There is substantial evidence of workflow processes for which trying to define (or prescribe) every step may compromise the process goal. In many cases, the work practices themselves would not fit into a prescriptive framework and introducing a technology which imposes it would result in decreased productivity. Nonetheless, these processes are not totally devoid of coordination and control requirements. In other words, the processes do not exclusively belong to the “ad-hoc” class of processes, which are generally not repetitive, and either represent an administrative level of complexity (e.g. document routing) or a very high level complexity (e.g. strategic planning) [SSO01]. Certain domains such as software development and research project management have increased likelihood of workflow processes that have both ad-hoc and prescriptive process requirements. However, most of today’s workflow management approaches lack the ability to provide adequate support for processes in such domains. A detailed analysis of the reasons for incomplete workflow processes as well as the consequences they cause are given in Section 6.1.

2.2 The WfMC’s workflow reference model

Workflow Management Coalition published its reference model in October 1994 [WfMC95], identifying the functional areas addressed by the workflow management facility and typical usage scenarios. This model defines a workflow management system and the most important system interfaces. Other WfMC standards as well as the OMG standard make reference to this model. Figure 2.1 shows the major
components and interfaces of a workflow management system, which are outlined as follows:

- **Workflow Engine**: A software service that provides the run-time environment in order to create, manage and execute workflow instances.

- **Process Definition**: The representation of a workflow process in a form which supports automated manipulation.

- **Workflow Interoperability**: Interfaces to support interoperability between different workflow systems.

- **Invoked Applications**: Interfaces to support interaction with a variety of IT applications.

- **Workflow Client Applications**: Interfaces to support interaction with the user interface.

**Figure 2.1 WfMC’s Workflow Reference Model**

- Workflow Engine: A software service that provides the run-time environment in order to create, manage and execute workflow instances.

- Process Definition: The representation of a workflow process in a form which supports automated manipulation.

- Workflow Interoperability: Interfaces to support interoperability between different workflow systems.

- Invoked Applications: Interfaces to support interaction with a variety of IT applications.

- Workflow Client Applications: Interfaces to support interaction with the user interface.
• Administration and Monitoring: Interfaces to provide system monitoring and metric functions to facilitate the management of composite workflow application environments

The release of this reference model is regarded as a milestone in workflow management area. Since its advent, the WfMC’s workflow reference model has been widely accepted as the guide to develop workflow systems. Almost all the deployable workflow systems are based on and compatible with this reference model.

2.3 Workflow approaches under aspects of client-server based distribution

Many research efforts have been carried out to address problems with respect to the topic of workflow distribution in the conventional client-server workflow environment. The importance of associating “workflow management” with “distribution” has been emphasised in a lot of literature, e.g., [MWWD+98, EP99, HHJN+99]. At the same time, some conceptual approaches and research prototypes have been proposed, which aim at addressing these problems by adding more distribution to the conventional client-server based workflow management systems, through different means.

2.3.1 Exotica/FMQM

Exotica/FMQM is developed at IBM Almaden Research Centre (http://www.almaden.ibm.com/cs/exotica/). This project aims at incorporating advanced transaction management capabilities in IBM’s products and prototypes. Work in Exotica is done in the context of IBM’s workflow product FlowMark and IBM’s messaging and queuing product MQSeries. For this reason, FMQM stands for FlowMark on Message Queue Manager.

The Exotica/FMQM project focuses on distribution, scalability and fault-tolerance by minimising the need for central control structures [AMGA+95]. This
strategic workflow product follows a layered client-server architecture that supports the concepts of build-time and run-time. In this workflow system, centralised data repository is replaced by persistent messages. A set of autonomous nodes cooperate to complete the execution of a process. Each node functions independently of the rest of the system. The only interaction between nodes is through persistent messages informing that the execution of a step of the process has been completed. Thus the performance bottleneck of having to communicate with the server during the execution of a process is avoided. In each node, a *node manager* builds a *process table* that contains static information related to the execution of process instances. During run-time, the information regarding the process instances will be stored in an *instance table* handled by the activity threads. Workflow data is transferred from *output containers* of finished nodes to *input containers* of next nodes with the help of messages.

Exotica/FMQM is a technique-oriented research prototype which has two concrete goals. The first is to study the effects of complete decentralisation on the design of a workflow product. The second is to analyse the feasibility of replacing a centralised database by persistent messages. However, structures of messages for the transfer of process description and workflow data are not explained in detail in the literature. Aspects of flexibility such as incomplete process support are not discussed in this approach either.

### 2.3.2 ADEPT

ADEPT stands for *Application Development based on Encapsulated pre-modelled Process Templates* ([http://www.informatik.uni-ulm.de/dbis/f&l/forschung/workflow/ftext-addept_e.html](http://www.informatik.uni-ulm.de/dbis/f&l/forschung/workflow/ftext-addept_e.html)). This project was started in 1996 at University of Ulm and aims at building the next generation workflow technology for enterprise-wide and cross-enterprise workflow management [RRD03]. One important facet in the ADEPT project is to investigate distributed workflow control in order to avoid overloading of the workflow servers and of the communication network.
To address these problems, ADEPT reduces the network load by partitioning workflow definitions and by migrating the control of workflow instances from one server to another during run-time, i.e., a workflow instance may no longer be controlled by only one workflow server. When performing such a migration, a description of the instance state is transmitted to the target server. This includes information about activity states as well as workflow relevant data. To avoid unnecessary communication between servers, ADEPT allows control of parallel branches of a workflow instance independently from each other. The communication behaviour can be further improved if variable server assignment expressions are used. These expressions can be determined at build-time, allowing the selection of a suitable workflow server to keep most of the communication local within the same subnet, and require almost no additional effort at run-time. Furthermore, ADEPT supports both static and variable server assignments [BD99]. The former means appropriate workflow servers are chosen for various partitions of a workflow definition. On the contrary, variable server assignment allows for dynamic workflow server assignment at run-time, which may improve the system performance significantly.

### 2.3.3 Endeavors

With the enormous growth of the Internet and World Wide Web (WWW), many authors have started to use this platform to enhance their workflow management systems. Web technology can be used to provide a coordination mechanism for distributed process execution and tool integration on the Internet.

Endeavors [HKBT98] is a workflow management prototype developed at University of California, Irvine. Endeavors is an open, distributed process modelling and execution infrastructure that addresses communication, coordination and control issues. To provide support for transparently distributed people, artefacts, process objects, and execution handlers, Endeavors has customisable distribution and decentralisation policies. In addition, Endeavors’ processes, as well as the means to visualise, edit, and execute them, are easily downloaded using current and evolving Web protocols. This is accomplished similar to how applets are executed on the
Web: sending both the data (process) and the means to execute the data (interpreter), which allows for coordination and communication between process participants who simply have access to the Web. Coordination between distributed sites is accomplished using a *summit* and *treaty* mechanism to negotiate and maintain constraints between sites.

Besides Endeavors, similar WWW-based approaches include WorldFlow [KRGL97] which is developed at University of Massachusetts at Amherst, WorkSpaces [Tol02] which is developed at Technische Universität, Berlin, and so on.

### 2.3.4 DartFlow

DartFlow [CGN96] is a workflow management system on the Web using transportable agents, developed at Dartmouth College. It uses Web-browser embedded Java applets as its front end and transportable agents as the backbone. While Java applets provide a safe and platform independent GUI, the use of transportable agents makes DartFlow highly flexible and scalable.

The emphasis of DartFlow is on ameliorating some of the nagging problems like lack of flexibility, adaptation, specialisation, and intelligent error handling that plague commercial workflow systems. In the DartFlow system, each workflow process is handled by a transportable agent, which is a program that can migrate under its own control from machine to machine. The agent follows the steps in a previously defined process and migrates to each site and gathers information. The transportable agent knows where it needs to travel within the workflow process. It also stores some basic knowledge about its internal states and behaviour. When an agent arrives at the user’s workstation, it acts according to the information it is carrying within itself. The agent stores a process map and generic object information that can be used at various steps in the workflow, consisting of state variables to store the application-specific data and methods for the particular object’s class. This means that each agent stores the business rules for the successful execution of each
step of the process. Therefore, there is no need to consult the central database server at every step.

2.3.5 METUFlow

METUFlow [GACT+97] is a distributed workflow management system developed at Middle East Technical University. This project aims at designing a distributed workflow enactment service which contains several schedulers on different nodes of a network. Each scheduler executes parts of process instances. Hence, such a system could fit naturally to the distributed heterogeneous environments, enhance failure resiliency and increase system performance.

The approach proposed in METUFlow is based on the observation that controlling the occurrence of events provides the coordination of the tasks. This means intertask dependencies are represented by event dependencies. To provide distributed execution of workflow computations, each event in METUFlow is made responsible for controlling its execution to decide on the right time to occur. Required information for this operation is termed as a guard, which is a temporal expression defined on an event. Occurrences of events are permitted only if their guards are true. Thus, each node in the process tree is implemented as a CORBA object with an interface for the guard handler to receive and send messages. Workflow is assigned to these CORBA objects, with computed guards controlling distributed execution.

2.3.6 Discussion

The above approaches do add some distribution to workflow systems and bring benefits such as improved performance, increased failure resiliency and enhanced scalability as they claimed. However, these approaches mainly address distribution instead of decentralisation. A common characteristic of these approaches is that they are still based upon and limited by the client-server architecture. Thus, these approaches either address the problems partly, or require complicated languages and/or complex algorithms. The remaining centralised services like centralised process instantiation and work assignment also make them relatively inflexible in
some application domains. Moreover, the aspects of user support and system openness are hardly ever considered.

To summarise, the problems related to the centralised system architecture have not been and probably cannot be addressed thoroughly when the whole workflow system is still based on the client-server architecture.

2.4 Decentralised workflow approaches based on other computing technologies

While client-server based workflow approaches failed to properly address the problems in the first group as described in Section 2.1, the emergence and boom of other computing technologies such as p2p technology and grid computing have provided new platforms for process support solutions. Some research efforts, although very limited, have been put into investigation of using these collaborative and decentralised frameworks to support workflow management systems.

2.4.1 PeCo

PeCo, which stands for Peer Collaboration, is an ongoing project developed by Proteus Technologies, LLC (http://www.proteus-technologies.com/). Proteus’ mission is to become a premier professional services leader in software and systems engineering, serving the Government Intelligence Community, Transportation, Health Care, and Commercial Industries. PeCo is a p2p-based collaborative workflow management system composed of peers, core services, applications, and portable plug-ins that enable generic system integration [Coo02]. PeCo’s purpose is to decentralise workflow management using collaborative technologies and concepts while providing a pluggable framework for integrating process applications and human participants.

In PeCo, workflow peers are peer agents that are responsible for a particular role in a workflow’s enactment. Core services, i.e., deployment tool, role coordinator, group coordinator factory, data extractor factory and administrator, are provided for
system initialisation and system administration. Moreover, Jini infrastructure and services are used to support crucial characteristics of the PeCo architecture including agent/peer discovery, fault tolerance, and peer availability awareness. In general, workflow peers interact with group coordinators to join enactment groups. Peers coordinate and interact with proprietary applications, user inboxes, and other peers to perform workflow tasks, through portable plug-ins.

2.4.2 An architecture based on WWPD and WWP

Another ongoing p2p-based workflow project is conducted at Manchester Metropolitan University. This project presents a p2p architecture for dynamic workflow management, which is based on concepts such as Web Workflow Peers Directory (WWPD) and Web Workflow Peer (WWP) [FK04]. The WWPD is a centralised feature of the system, which provides a peer registration service and maintains a list of active peers and their profiles. With support of this architecture, peers are allowed to register with the system and offer their services and resources to other peers. During the execution of workflow instances, Workflow process administration is achieved by employing a notification mechanism. It is claimed that such an approach is adaptive, easily scalable and flexible.

2.4.3 RainMan

RainMan [PPC97] is a distributed, object-oriented workflow system developed at IBM T.J. Watson Research Centre. It is developed in Java that lives naturally on the Internet. Therefore, the potential of the Internet can be explored to enable decentralised workflow execution via interoperable workflow components that reside across this global infrastructure. This system is a loosely-coupled collection of independent services that cooperate with each other rather than being a monolithic system.

RainMan is based on RainMaker, a generic workflow framework that defines a core set of abstract interfaces for workflow components. RainMaker identifies four important abstractions within the workflow domain, which are Sources, Activities,
Performers and Tasks. PerformerAgent and SourceAgent are core RainMaker interfaces that help support the decentralised execution model.

In RainMan, workflow management, activity distribution, directory services, and worklist management are all treated as independent services that work together to deliver workflow functionality to Internet users. Moreover, issues of performance, failure handling and compensation, and security are also mentioned in RainMan. By using RainMan, it is expected to form virtual enterprises on the Internet, which involve dispersed individuals, multiple organisations, scattered network resources, and heterogeneous workflow systems.

2.4.4 Serendipity-II

Serendipity-II [GAHM98] is a decentralised process management environment developed at University of Waikato and University of Auckland, New Zealand. It supports collaborative and distributed process modelling and enactment for distributed software development projects.

To provide robust, efficient, distributed process support, the Serendipity-II environment is based on a decentralised architecture which uses multiple point-to-point communication across the Internet. Each user’s environment has its own receiver and sender components to communicate with other users’ environments. By these means, distributed users are allowed to edit process models collaboratively. To enact a process, each user’s environment has a process enactment engine which is able to enact process stages (tasks) locally. Enactment events are generated and propagated to other users’ environments, which subsequently cause other connected stages to be enacted, finished, and so on. Moreover, to manage complex cooperative software development, decentralised work coordination agents are developed to automate tasks, track work history, coordinate work, and integrate tools.

2.4.5 Matrix

The Matrix project ([http://www.npaci.edu/DICE/SRB/matrix](http://www.npaci.edu/DICE/SRB/matrix)) is developed at SDSC (San Diego Supercomputer Centre). The goal of the Matrix is to perform research
and development in order to deliver the grid workflow protocols and workflow language descriptions necessary to build a p2p infrastructure for Grid Workflow Management Systems.

Grid workflow is emerging for computation intensive domains such as earthquake forecast. To date, the Matrix middleware allows applications and services based on standards such as Web Services Description Language (WSDL) and Simple Object Access Protocol (SOAP) to communicate with data and other resources in Grid environments. Notification and standard querying mechanisms at run-time are offered to coordinate a workflow between multiple peers in a grid infrastructure. P2p gridflow brokering is also provided, which breaks and analyses each flow using suitable heuristics. Distributed protocols for coordination and confluence of the gridflow brokers are designed. In the near future, Matrix will completely use the Data Grid language and Sangam Protocols to provide p2p gridflow management.

2.4.6 Discussion

The above approaches abandon traditional client-server architecture and adopt newer, decentralised architecture to support processes. In particular, the few attempts at combining p2p computing technology with workflow technology have opened brand-new ground in workflow, and process support area in general. The distinguished features of p2p technology make it suitable to address the problems related to the client-server architecture ultimately. The potential of p2p-based workflow, which offers significant value to organisations, has been revealed in the existing approaches. Based on the analysis of the existing approaches, it is justifiable to believe that p2p-based workflow may have a bright future and would likely to become popular in the near future.

However, it is evident from the literature that research on implementing workflow in a p2p environment is still at a very initial stage with many problems being addressed insufficiently. Fakas’s work on WWPD and WWP, only reports conceptual ideas about linking workflow with p2p without concrete system analysis,
design and prototyping. Matrix, an ongoing project which is also incomplete, may not be suitable for non-Grid workflow domains. Other approaches like PeCo, RainMan and Serendipity II mainly focus on decentralised enactment of process instances at run-time in order to remove potential performance bottleneck, increase fault tolerance and offer enhanced scalability. However, some issues which are essential to decentralised enactment have not been specified clearly in these approaches. For example, it is not clear in these approaches that how the process definition data are managed so that decentralised peers are able to access task information at run-time. Subsequently, problems of process instantiation have been found missing in these approaches. Issues such as dynamic participants, work allocation and better user support also have not been addressed sufficiently. Moreover, none of these approaches has addressed problems of incomplete process support in a decentralised environment, which makes them unable to support workflows in non-traditional workflow domains such as research project management, as discussed in Chapter 6. Therefore, there is still a long distance to go in exploiting p2p-based workflow and the challenges are significant.

2.5 Workflow research related to incomplete process support

Incomplete process support is one of the key foci in the recent development of workflow systems. However, very limited work has been conducted so far and only few approaches can be found in the literature.

2.5.1 Workflow Continuum

The workflow Continuum was developed in 1993 by Nastansky and Hilpert at University of Paderborn, Germany [NH94]. It identifies four different categories of workflow management on a scale between structure and flexibility, as shown in Figure 2.2.
Workflow Continuum

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>e-mail, store and forward</td>
<td>one-step, next agent, shared DB</td>
<td>combination of pre-determined and open tasks within a single workflow</td>
<td>generally preset next agent, shared DB</td>
</tr>
<tr>
<td>– urgent</td>
<td>– shared access</td>
<td>– completely open as well as standardised tasks</td>
<td>– highly recurrent</td>
</tr>
<tr>
<td>– short-lived</td>
<td>– common task</td>
<td>– intersection of both</td>
<td>– pre-determined</td>
</tr>
<tr>
<td>– exceptional</td>
<td></td>
<td></td>
<td>– easy-to-apply ad hoc modification / re-routing</td>
</tr>
<tr>
<td>– confidential</td>
<td></td>
<td></td>
<td>– pre-defined</td>
</tr>
<tr>
<td>e.g. new type of request</td>
<td>e.g. co-authoring of publication</td>
<td>e.g. co-editing of annual report</td>
<td>e.g. consumer credit application (particular customer request)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>e.g. consumer credit application</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a). open team task within standard WF</th>
<th>b). controlled team task within standard WF</th>
<th>c). ad hoc modification of standard WF</th>
</tr>
</thead>
<tbody>
<tr>
<td>– urgent</td>
<td>– partially unspecified elements within pre-determined workflow group (n of m)</td>
<td>– completely pre-determined workflow and exception</td>
</tr>
<tr>
<td>– short-lived</td>
<td></td>
<td>– highly recurrent</td>
</tr>
<tr>
<td>– exceptional</td>
<td></td>
<td>– pre-determined</td>
</tr>
<tr>
<td>– confidential</td>
<td></td>
<td>– easy-to-apply ad hoc modification / re-routing</td>
</tr>
<tr>
<td>e.g. new type of request</td>
<td>e.g. counter-signature</td>
<td>e.g. consumer credit application (particular customer request)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>e.g. consumer credit application</td>
</tr>
</tbody>
</table>

flexible, changeable, unique determined, structured, recurrent

Figure 2.2 Workflow Continuum ([NH94], all rights reserved)

On the left-hand side of the workflow continuum, flexible group interaction patterns are described. Category one, ad hoc workflow, usually deals with unique and rather short-lived processes. Category two, autonomous workgroup, addresses team work in general and is not related to workflow management exactly. Autonomous workgroup makes use of shared databases for team members to work cooperatively on a team task. On the right-hand side of the workflow continuum, category four—the structured workflow pattern—is described, which is usually highly recurrent and well structured. Finally, category three, the so-called semi-structured workflow, is located in-between these two extremes. It is made up of different combinations of structured and flexible elements.

This model contributes the basic patterns to the research on flexible workflow from a process design point of view. Since its first publication in 1993, the workflow continuum has been discussed and used by many authors [e.g., Mar95].
2.5.2 WASA

WASA workflow [Wes98] is a research project developed at University of Potsdam, Germany. This project aims at investigating the usability of workflow technology in the context of applications in science and engineering. During the WASA project, the formal foundation, conceptual design, and prototypical implementation of flexible and distributed workflow management systems based on object-oriented middleware were developed.

The flexibility aspect is an important research topic in the WASA project. The researchers present a hierarchical workflow execution approach based on a set of states and accompanying state transitions for workflow instances. A complex activity may have a nested structure and activity models are created using a set of activity modelling operations. The researchers also identify that there is a myriad of unforeseen events which cannot be modelled completely before activity instances start. Therefore, incomplete process support is new functionality that a workflow system should provide. A set of operations are then presented to satisfy the flexibility requirements, which include user intervention operations and dynamic change operations.

2.5.3 WORKWARE

Håvard from SINTEF Telecom and Informatics, Norway focuses on human-centred solutions and argues that interactive enactment should be pursued more vigorously as a framework for flexible workflow modelling, allowing incomplete workflow models to emerge [Håv01].

This approach supports the view that workflow model activation should be shifted from automated to interactive enactment, and that interaction is a suitable framework for understanding and designing flexible workflow management systems. Interactive enactment allows intertwined articulation and activation of an evolving online model so that the model needs not be 100% complete and consistent. A general interactive workflow management system architecture is presented in this research, which has three layers: shared workflow models, a number of model
activators and an integrated user interface. Using this architecture, the model activators utilise the shared workflow models to provide articulation and activation services through the user interface. This research also presents the WORKWARE prototype, which attempts to re-interpret the workflow concept to include processes with emerging structure.

2.5.4 Pockets of Flexibility

“Pockets of Flexibility” is a concept presented by researchers at University of Queensland, Australia [SSO01, MS02]. Based on this concept, a process modelling and enactment approach has been proposed, which allows capturing both ad-hoc and prescriptive process requirements within the same framework. Flexibility in this research is defined as the ability of the workflow process to execute on the basis of a loosely, or partially specified model, where the full specification of the model is made at run-time, and may be unique to each instance.

To provide a modelling framework that offers true flexibility, the factors which influence the paths of (unique) instances together with the process definition are considered. An approach that aims at making the process of changing part of the workflow process itself is advocated. In Pocket’s framework, the notion of an open instance that consists of a core process and several pockets of flexibility are introduced. A pocket is a special structure within the workflow model, that consists of workflow fragments, which can represent a single business activity, or a complex sub-process; and a special activity called the Build Activity, which carries the rules and constraints under which these fragments can be composed for a given instance. Thus, the build activities provide the functionality to integrate the process of defining a change into the open workflow instance. This framework does not introduce any additional or non-standard modelling constructs, since the dynamic compositions can be mapped to core (standard) modelling constructs. Thus the impact on the functionality of the underlying scheduler is minimal. However, the issue of dynamic modification of running instances is obviously required, since instance templates are partially composed at run-time.
2.5.5 Discussion

From the literature, it is clear that the research on the incomplete process is also at a preliminary stage. The existing few approaches discussed above address the issues of incomplete process support mainly from the modelling technique rather than the system coordination support point of view. Although some mechanisms for run-time process modelling are presented, aspects of system support for on-the-fly process articulation are rarely addressed. Moreover, these approaches are all based on the conventional client-server architecture. Relevant research in decentralised workflow environments from a system coordination support point of view is hardly ever conducted.

2.6 Requirements analysis

After the in-depth analysis of the unsolved problems in the workflow area (Section 2.1) and the literature review of the state-of-the-art in workflow management (Sections 2.2, 2.3, 2.4 and 2.5), it is believed that the problems related to centralisation and the problem related to incomplete process support have become major obstacles for wide deployment of workflow technology in real-world applications. At the same time, the industry trends such as virtual enterprises and flattening of organisational structures also show that the future image of business will include distributed groups of collaborating teams that combine talents and skill sets to create new methodologies and processes. Therefore, there is a growing trend that the next generation of workflow systems will be built in a truly decentralised manner, providing supports for both complete and incomplete processes [Yan02b].

The growing popularity of p2p technology is making workflow decentralisation more visible. The unique features of p2p reflect workflow’s distributed feature more naturally. The few attempts at replacing the aging, old-fashioned client-server paradigm with p2p collaborative and decentralised framework have revealed great potential. Very recently, p2p-based workflow systems have also been recognised as one of the most strategic future directions for workflow research [Mas02]. Therefore,
p2p-based decentralised workflow might be a highly relevant and practical solution for future process support and deserves intensive investigation.

To have a cost-effective and decentralised workflow system based on p2p, the system needs to be re-designed fundamentally. The centralised server is expected to be ruled out while the services traditionally provided by the centralised data repository and workflow engine should be retained. In particular, to achieve this, a decentralised workflow system should:

1. adopt the p2p loosely-coupled structure and provide a pluggable framework for integrating workflow process applications and human participants which imply the presence of neither a centralised data repository for data storage, nor a centralised workflow engine for coordination,

2. store data, traditionally stored in a centralised data repository, in a decentralised way without losing information so that relevant sites in the system can access the data when necessary,

3. migrate services traditionally performed by a workflow engine to other sites (peers) in the system so that the decentralised run-time environment can provide the functionality of a workflow engine,

4. provide means to locate the service providers in a way that the traffic (requests and responses) could be guided to appropriate sites (peers) automatically,

5. enable a service seeker to communicate with a service provider directly, and vice versa,

6. accommodate service-oriented applications which is the trend for application systems including WfMS from the system openness point of view, and
(7) provide adequate support for incomplete processes, which allows incomplete processes to be specified and stored at build-time, and instantiated, articulated and executed on-the-fly at run-time.

With the above objectives achieved, the problems discussed in Section 2.1 will be expected to be largely addressed. Hence, workflow systems will be expected to be widely deployable to real world applications to support processes, which was infeasible before.

2.7 Summary

In this chapter, the motivations of this research have been evaluated. The major unsolved problems in today’s workflow research, namely, bad performance, vulnerability to failures, poor scalability, user restrictions, unsatisfactory system openness, and lack of incomplete process support, as well as origins of these problems have been analysed profoundly. The literature in relation to these problems has been widely reviewed. Based on the problems analysis and literature review, the p2p-based workflow is suggested in supporting both complete and incomplete processes as the most fruitful line to follow. The detailed requirements for p2p-based workflow have also been analysed.
Chapter 3

System framework and design

As analysed in Section 2.1, the client-server architecture has placed some limits on reflecting workflow’s increasingly distributed nature, which has caused serious problems subsequently. In this chapter, the framework and design of SwinDeW, an innovative, decentralised workflow management approach, are presented. This approach combines workflow technology with p2p technology and aims at providing genuinely decentralised coordination support for workflow management. In particular, this approach is designed to support complete processes first in Chapters 4 and 5 and is further extended in Chapter 6 for supporting incomplete processes.

This chapter first introduces the p2p technology briefly in Section 3.1. Then in Section 3.2, the overall workflow system framework is presented. After that, Section 3.3 discusses the structure as well as the organisation of peer which is the most important entity in the system. Based on the peer structure and organisation, the peer management service is addressed in Section 3.4, which includes the peer discovery service, the peer registration service and the capability configuration service. Finally, Section 3.5 discusses the issues regarding system initialisation briefly.

3.1 Peer-to-peer technology

There is a revolution underway that represents a different computing model for distributed systems. This revolution is being sparked by the technology known as
peer-to-peer (p2p), or peer-to-peer computing [MKLN+02]. P2p can be simply defined as the sharing of computer resources, information and services by direct exchange. The concept of p2p is not new and could be traced back to the original design of the Internet where any two computers could send packets to each other [AH01]. Usenet news, which allows people to exchange information freely, is regarded as an example of old-fashioned p2p. Nowadays, p2p is driving a major shift in the area of genuinely distributed computing and has achieved considerable traction with mainstream computer users and service providers. Success stories such as eBay (http://www.ebay.com), Gnutella (http://gnutella.wego.com/), Freenet (http://freenet.sourceforge.com/) and ICQ (http://www.icq.com/) have shown the practical applicability and public demand for such systems.

As illustrated in Figure 3.1, there are fundamental differences between the client-server and p2p models. In the client-server model, every exchange or communication goes through, and is managed by, a centralised server. By contrast, in a p2p system, every participating node, which is known as a peer, can serve the function of both a client and a server. That means, a peer can initiate requests (like a client), and it can provide access to its resources and respond to requests from other peers (like a server). In reality, p2p is a broad term covering a number of different yet related decentralised architectural styles, such as the homogeneous p2p system, the hybrid p2p system, the hierarchical p2p system, and so on. Generally, speaking,
the distinguishing properties of all these p2p systems, as summarised in Aberer’s conceptual definition [AH01], are as follows:

- no central coordination
- no central database
- no peer has a global view of the system
- global behaviour emerges from local interactions
- peers are autonomous
- all existing data and services should be accessible

The p2p computing model offers a number of compelling advantages [Ste01]. Technically, p2p computing eliminates the risk of single-source bottleneck and failure, shares different computer’s processing and storage capabilities, enables better scalability, and provides load balance in large organisations. In addition, p2p computing offers a very open framework which signifies the effective use of distributed resources. Thus, systems built on top of p2p are expected to achieve better performance, increased resilience to failure, improved scalability, and enhanced system openness. Besides these, much of the wide appeal of p2p is due to social factors [Bar00]. Since every peer in a p2p system is autonomous, ordinary participants can always play more active roles and enjoy the ability to bypass centralised control. It is now believed that p2p is not only an extension and complement to the traditional client-server model, but also a potential technology that could re-engineer distributed architectures.

### 3.2 Decentralised workflow architecture based on p2p

Based on the discussion in Section 3.1, it is believed that p2p computing can be utilised to provide a decentralised workflow environment and address some of the problems discussed in Section 2.1, i.e., bad performance, vulnerability to failures, poor scalability, user restrictions and unsatisfactory system openness. This is because the following reasons:
(1) P2p’s ability to share computer’s computing and storage capabilities can greatly enhance system performance.

(2) Since p2p removes centralised services like the centralised engine and the centralised database, the risk of single point of failure can be largely decreased.

(3) System scalability, which is weak in conventional client-server based workflow systems, is a distinction offered by p2p computing architecture.

(4) P2p allows more autonomy for participating peers (participants), which can loosen users from heavy restrictions.

(5) The p2p collaborative framework provides complete openness for service-oriented workflow. P2p and services (including Web services) are closely related and may converge to become a single category in the future.

![Decentralised system architecture of SwinDeW](image)

**Figure 3.2 Decentralised system architecture of SwinDeW**

Combining concepts from workflow management and p2p computing, SwinDeW is designed as a special p2p-based system which provides workflow support in a truly decentralised manner [YYR02]. SwinDeW adopts a flat, flexible and loosely-coupled structure with an intentional absence of a centralised data
repository for data storage and a centralised workflow engine for coordination. Figure 3.2 illustrates SwinDeW at a very high level. In brief, the system is defined as four layers. The top layer is the application layer, which defines the application-oriented semantics to fulfil workflow functions. Core services of the workflow system are provided at the service layer, which include the peer management service, the process definition service, the process enactment service, and the monitoring and administration service. The data layer consists of data repositories (DRs) which store workflow-related information such as process definition and instance information. At the bottom, the communication layer facilitates direct communication among workflow participants.

To further discuss this system architecture, the terms used in this architecture are formally defined as follows:

- WfPS: Workflow Participant Software is an application which provides interfaces to interact with a workflow participant and other workflow participant software, requesting services and responding to requests. The internal structure of this software is detailed in Section 3.3.1.

- Peer: A peer is the basic working unit in SwinDeW, which consists of the WfPS and a set of data repositories (DRs) as detailed in Section 3.3. In most cases, a peer that resides on a physical machine is associated with and operates on behalf of a workflow participant, to enable direct communication with other peers in order to carry out the workflow.

- Peer management service: The peer management service is a system service which performs functions to configure, manage and administer the peers in the system. The key components of this service include the peer discovery service, the peer registration service, and the capability configuration service, as detailed in Section 3.4.
(1) Peer discovery service: The peer discovery service is an automated service which enables a peer to locate other peers in the workflow system.

(2) Peer registration service: The peer registration service is a system service which provides enhanced scalability for the workflow system. With support of this service, new peers can easily join the system at any time, and existing peers can leave the system when needed.

(3) Capability configuration service: The capability configuration service is an administrative module to manage the capabilities of workflow participants which are objects encapsulating rules associated with roles in workflow processes. For example, the capability of the course advisor in a student registration processing workflow (as discussed in Section 7.1) might require registration documents for approving or rejecting a registration request.

- Process definition service: The process definition service is a system service which provides support for workflow build-time functions, including process representation, process data storage, and process data maintenance in a dynamic environment, as detailed in Chapter 4.

- Process enactment service: The process enactment service is a system service which provides support for workflow run-time functions, including process instantiation and instance execution, as detailed in Chapter 5.

- Monitoring & administration service: The monitoring & administration service is a system service that provides administrative capabilities, status monitoring, performance feedback, historical processing information and human intervention, as detailed in Chapter 5.
• Data layer: The data layer of SwinDeW consists of a set of distributed data repositories which store various data in relation to the workflow system and the workflow application. The details of data repositories will be discussed in Section 3.3.1.

• P2p communication layer: The p2p communication layer is the underlying network infrastructure which enables data transfer between physical machines through point-to-point connections. The management of physical connection and communication is beyond the scope of this thesis. Some details of the communication layer are addressed in Chapter 8 when the prototype implementation is discussed.

In order to focus on workflow management rather than p2p itself, this research assumes that the underlying p2p network is homogeneous and all peers are physically interconnected. In fact, as detailed in Section 3.3.2, peers are gathered logically in SwinDeW to form an overlay network layered on top of the existing p2p network infrastructure.

The design of SwinDeW is compatible with the WfMC’s workflow reference model [WfMC95]. In particular, the WfPS plays the role of a workflow client application. The process definition service, the process enactment service and the monitoring & administration service correspond to the process definition module, the workflow engine, and the administration and monitoring tools, respectively. However, the interoperability interface and the invoked application interface are beyond the scope of this thesis. Moreover, because SwinDeW is based on the p2p decentralised computing architecture, all the core services are not provided by a single site (server). Instead, the services are provided through the collaboration of many peers involved. The mechanisms for these services will be discussed in the rest of this thesis.
3.3 Peer structure and peer organisation

As indicated in Section 3.2, the peer is the basic functional unit in SwinDeW. Each peer represents a self-managing, independent entity which is normally associated with a workflow participant. In this Section, the structure of a peer and the way peers are organised are described.

3.3.1 Peer structure

From the functional perspective, each peer contains the workflow participant software, which consists of three software components: user component, task component and flow component. These three software components interact with one another internally to facilitate the associated workflow participant to act independently in the system. At the same time, these software components interact with software components of other peers, allowing the associated workflow participant to act collaboratively in the system. To facilitate the functions of software components, each peer also contains four data repositories: peer repository, resource and tool repository, task repository, and process repository.

Figure 3.3 depicts the structure of a peer in SwinDeW and the interactions among the software components and data repositories. These software components and data repositories represent a number of workflow features and facets in a precise and comprehensive way:

- Software components:

  - A user component of a peer is a “bridge” between the associated workflow participant and the workflow environment. On one hand, a user component provides procedures and operations to the associated participant by delivering essential information of the workflow. On the other hand, a user component represents the roles of the associated participant in the system in terms of capabilities (e.g., enrolment officer and course advisor in a student registration workflow as discussed in Section 7.1). In addition, a user
component also manages the peer’s organisational model which is stored in the peer repository, and is involved in providing services such as the peer discovery service.

![Diagram of SwinDeW peer structure]

**Figure 3.3 Structure of a peer in SwinDeW**

- **A task component of a peer** is in charge of the execution of tasks undertaken by the associated participant. In detail, a task component determines the time when a particular task starts, monitors the execution, administrates resources, invokes tools when necessary and terminates the thread after the work is done.

- **A flow component of a peer** helps to fit an individual task into the workflow. The main purpose of a flow component is to deal with data and control dependencies amongst tasks by handling incoming and outgoing messages. The data dependency means a task may require some artefacts as inputs, which are normally created by preceding tasks, and generates some artefacts as outputs, which are normally consumed by succeeding tasks. The control dependency means a task can start only after all or some of its preceding
tasks are complete, and the completion of a task may trigger the start of its succeeding tasks. In this way, the workflow execution can be coordinated step-by-step as pre-defined. A flow component manages the process repository which stores the process definition distributed to this peer.

- Data repositories:

  ➢ A peer repository stores an organisational model which represents organisational entities and their relationships. This repository represents a participant’s view of the system’s overall organisational model, which means different peers in the system may have different peer repositories. Since a peer in the system only has local knowledge about the organisation as discussed in Section 3.1, its view of the overall organisational model is normally incomplete. Moreover, as the overall system’s organisational model is subject to changes and a peer may learn and acquire knowledge about the organisation during the system operation, the peer repository is also dynamic. In SwinDeW, a peer repository is realised in a directory. Each record in a peer repository contains complete information about an individual peer and can be described in XML format as:

  ```xml
  <Peer>
      <Peer_name />
      <Peer_location />
      <Peer_capability />
  </Peer>
  ```

To a peer, all the peers in its peer repository are known as its neighbour peers. A peer’s knowledge about another peer is known as peer awareness. In SwinDeW, peer awareness is bi-directional. This means if peer A knows about peer B (i.e., B is in A’s peer repository), then peer B also knows about peer A (i.e., A is in B’s peer repository). This feature eases the peer registration service which is discussed in Section 3.4.2.
A resource repository stores data related to the specific workflow application, including tool resources which represent non-human resources such as machines, external hardware, applications, and data resources such as documents. This repository is managed by the task component, as a task instance may require resources to support its execution. As indicated earlier, the interfaces to invoke applications are not emphasised in this thesis.

A task repository stores a set of task instances, which represent the work allocated to the associated workflow participant in the context of process instances. Complete information about each task instance, such as deadlines, instance-related data, audit and monitoring information about the past instances are stored in this repository. The task component and the flow component co-govern this repository to ensure the correct execution of each task instance, as detailed in Chapter 5.

A process repository, which stores a partial process definition distributed to this peer, is managed by the flow component. The process definition stored in a peer’s process repository is actually composed of definitions of a set of tasks, which provide the peer with the templates for the work to be done. The mechanism for process definition distribution is addressed in Chapter 4.

Given essential information and authority, each peer is an autonomous entity. Compared with the client-server architecture, it is clear that a peer in SwinDeW plays not only the role as a client because the peer may send requests and notifications to other peers, but also part of the role as a server in two areas. First, a peer helps to manage part of data such as organisational information and process definition, which is conventionally performed by a centralised data repository. Second, the three components interact with one another internally and externally to provide run-time coordination support for the execution of workflow instances, which is conventionally performed by a centralised workflow engine.
3.3.2 Peer organisation

Although SwinDeW is based on a p2p infrastructure, a purely flat peer organisation is not suitable. This is because of the following reasons. First, since every peer is associated with a workflow participant, the peers should be organised in a manner that reflects the system’s organisational model, e.g., the relationship between one participant and another should be well reflected from the relationship between the corresponding peers. Second, since a workflow system is a collaborative system, the peers should be organised properly so that a peer discovery service can be realised easily for peers to locate other cooperative peers.

Considering a real world organisation, staff members are always grouped into various units to form a pyramid-like hierarchical organisational model, according to the skills they have or the roles they play. Staff in the same unit normally have capabilities in common and share the same interests. Similarly, peers in SwinDeW are grouped logically into various virtual communities, according to the capabilities of their associated workflow participants. Each virtual community is characterised by a particular capability and formed by a cluster of peers which are associated with the workflow participants who have this capability. In each community, there is a peer that plays the role of a coordinator, performing some administrative functions. This peer is known as a coordinator peer. Conceptually, the administrative work is very simple and can be done automatically by a peer without human intervention. So each peer in the community is capable of exhibiting the same functionality. In practice, the role of the coordinator can be delegated to a particular peer when the system is started up. An alternative is to allow all the peers in the community to be the coordinator peer in turn. Therefore, in case that a coordinator peer becomes unavailable, another peer can be selected to act as a coordinator, as discussed in Section 3.4.2.
As shown in Figure 3.4, each peer in SwinDeW is involved in at least one virtual community according to the capability attribute of the associated participant. For each peer, information about other peers in the same community is captured and stored in its peer repository when this peer is started up. In other words, a peer is aware of all the other peers in the same community. As a result, all peers in the same community are interconnected (represented by the edges connecting peers within a community in Figure 3.4). If a participant has more than one capability, the associated peer is involved in multiple virtual communities accordingly, e.g., peer P4 in Figure 3.4 is involved in both communities A and B. In this case, all the peers in these multiple communities are interconnected through this common member. In other words, all the peers in these multiple communities are able to be aware of one another. For example, in Figure 3.4, the peers in communities A and B are interconnected via peer P4. In addition, some external means such as organisational management and organisational configuration can be used to ensure that no community in the system is isolated although this situation is normally rare. For example, the line in bold which connects peers P5 and P8 in Figure 3.4 is a configuration connection which connects community C with the rest of the system.

With this logical gathering of peers, any two peers in the same community are able to locate each other. Two peers in two different communities can locate each other if there is a peer involved in both communities. Even if there is an absence of a particular peer involved in two communities, peers in these two different...
communities can still find a path towards each other via third party peers. For example, peers P2 and P9 in Figure 3.4 can find a path to each other via peers P4, P5 and P8. Please note that Figure 3.4 only describes the static communities at an initial stage. In fact, virtual communities in SwinDeW are formed and changed dynamically with the discovery of unknown peers, join of new peers and departure of existing peers. This will be discussed in Section 3.4.2.

3.4 Peer management service

Based on the peer structure and peer organisation discussed in Section 3.3, design of the peer management service is discussed in this section.

3.4.1 Peer discovery service

The peer discovery service is one of the key components of the peer management service. As discussed in Section 3.2, SwinDeW carries out workflow functions through the direct communication and coordination amongst peers. However, since each peer only has local knowledge about the system, the peer discovery service helps a peer to locate unknown collaborative peers when needed.

Generally speaking, the peer discovery service is based on capability-based addressing. Given a certain capability, a peer can always locate at least one peer with this capability. Each discovery request, issued by a peer known as a request peer, contains a value indicating the capability information about the destination peers. The procedure of resolving a discovery request is very similar to the ideas of DNS on the Internet. This service converts a capability (domain name) to the location(s) (IP address) of one (many) peer(s). The steps of the peer discovery are described as follows:

1. If the request can be satisfied locally at this peer, it is done. This means at least one peer with the requested capability is in the peer repository of the request peer.
If the request cannot be satisfied locally, the request is routed by the request peer to its neighbour peers, i.e., peers in its peer repository. If the request can be resolved at this point, it is done. Otherwise the request may be further routed by the neighbour peers to their own neighbour peers.

Step (2) is repeated until the destination information is obtained.

It may happen that a peer receives the same request more than once, which is both time- and resource-consuming. For example, a neighbour peer of the request peer receives a discovery request from the request peer but cannot resolve this request. Later on, this peer, as a neighbour peer of another peer that routes this request, may receive the same request once again. To avoid this situation, those peers which have attempted (but failed) to satisfy the request will be attached to the request itself. The peers that receive and cannot resolve such a request will attach themselves to the request and route the new request only to those neighbour peers which are not enclosed in the request.

Using the peers in Figure 3.4 as an example, suppose that peer P₁ is looking for a peer with capability C, i.e., one of peers P₇, P₈ and P₉. First, peer P₁ executes Step (1) but failed to satisfy the request locally because none of peers in its peer repository (P₂, P₃ and P₄) has capability C. Then P₁ moves to Step (2) and propagates the request to P₂, P₃ and P₄. Again, none of them can satisfy this request based on their own peer repositories. Hence, the request is further routed to P₅ and P₆, which are peers in the peer repository of P₄, to execute Step (3). Finally, P₅ can resolve this request because P₆ is in its peer repository which has capability C. The information about P₆ is returned from P₅ to P₁, and the service invocation is terminated. Note that once a peer has detected a new peer and received information about the new one, it would insert a new record into its peer repository. Later on, this peer can locate the detected peer in its peer repository without requesting other peers again. This is a self-learning procedure and thus a peer is capable of discovering the system more
and more. With this service, a peer in the system is able to locate other peers with certain capabilities. Then the request and response messages can be transferred directly between peers.

3.4.2 Peer registration service

Workflow systems are always considered dynamic and scalable with peers joining and departing sometimes, in correspondence with the dynamic behaviours of their associated workflow participants. For example, new participants will go online and an existing participant may leave the system and rejoin collaboration from a different computer later on. When SwinDeW is well-established, it is assumed that no peer has global influence to the system. Therefore, an individual peer’s behaviour only has influence in a local scope and will not bring the whole system down. For example, when a single peer goes offline, the system should remain operational. This issue will be further discussed in Section 5.4.3.

Unlike the client-server based workflow systems which use the server configuration to manage client dynamism, SwinDeW supports peer dynamism through the use of a decentralised peer registration service. Generally speaking, this service manages and configures a peer’s dynamic behaviour locally and only modifies and updates the relevant peers which are affected by a peer’s behaviour. As discussed in Section 3.3.1, the information about participating peers is stored in the distributed peer repositories. Apparently, alteration of participating peers will affect the peer repositories of relevant peers. In other words, the adjustment of the peer repositories of relevant peers reflects the alteration of the participating peers.

When a new workflow participant joins the workflow management system, the associated peer actually joins some virtual communities according to the capabilities of this participant. A new peer notifies the system of its joining through issuing an announcement which carries the information about itself including its capability information, in order to let the system know about its intention to join. The announcement could be sent to and interpreted by any existing peer which is
detected by the new peer using the underlying p2p communication service (e.g., the JXTA discovery service discussed in Chapter 8). This existing peer is known as an access peer for the new peer because it serves as an access point for the new one to join the system. To achieve this, the access peer retrieves the capability information of the new peer and then invokes the peer discovery service to locate another peer with the same capability. After that, the announcement is re-routed to peer(s) in the virtual community which is labelled by this capability. These peers of the community accept the new peer as a member of the community by inserting a new record containing the information about the new peer into their peer repositories. At the same time, each peer in the community responds with a confirmation message which contains its own peer information. This information forms a partial organisational model for the new peer and allows the new peer to establish its initial peer repository. If a new peer has more than one capability, multiple announcements will be issued to join the virtual communities one-by-one. Moreover, a new peer also needs to get some essential workflow-related data such as process definition data. This operation will be further addressed in Chapter 4.

An existing peer can leave the system intentionally according to some pre-defined rules. Normally, some constraints exist for a peer’s departure. For example, a peer cannot leave a system intentionally when the associated participant is carrying out a task instance. For an existing peer, its existence is known by some other peers because its information such as capability and location are stored in the peer repositories of some other peers as part of their organisational models. This information allows them to provide services or make decisions. When a peer leaves the system, those peers which store its information in their peer repositories might be affected because the departure of a peer makes its information in others’ peer repositories obsolete. Based on this outdated information, the peers may thereby provide wrong services or make wrong decisions. For example, a peer may provide wrong information to others in the peer discovery service due to the obsolete information in its peer repository. Therefore, a peer should keep those relevant peers posted about its departure and the relevant peers should update their peer repositories properly.
To notify the relevant peers about its departure, a peer needs to determine the set of peers which store its information in their peer repositories. In SwinDeW, the records in a peer’s peer repository can be regarded as links to various peers. As discussed in Section 3.3.1, these links are bi-directional. If peer $A$ has the information about peer $B$ in its peer repository, then the information about peer $A$ is also in the peer repository of peer $B$. As a result, for each peer in SwinDeW, only its neighbour peers, i.e., those peers that are in this peer’s peer repository, store the information about this peer. Therefore, in the situation that a peer leaves the system, only its neighbour peers need to be informed. This design of the peer repository simplifies the problem. A peer does not need to find those peers who store its information when it leaves. Alternatively, a peer notifies its neighbour peers to let the system know of its departure. Upon receiving a departure notification, the relevant peers update their peer repositories and delete the information about the leaving peer. Furthermore, if a coordinator peer leaves the system, the leaving peer will designate an available peer as the new coordinator peer before it leaves and let all the other peers in the same community know. For example, the leaving coordinator peer can randomly select an available peer from the community to be the new coordinator peer.

Meanwhile, it is possible that some information may be lost if a peer leaves the system. For example, as shown in Figure 3.5, the departure of peer $P$ which is involved in two virtual communities may isolate the peers in one community from those in the other. To avoid information loss, the leaving peer also transfers the information about coordinator peers of various communities it is involved in to one another. These coordinator peers then update their repositories appropriately, inserting the new peers into their peer repositories if necessary. For example, peer $P$ in Figure 3.5 (a) transfers the information about coordinator peer $C_1$ to coordinator peer $C_2$ and also transfers information about $C_2$ to $C_1$ before it leaves. As a result, $C_1$ and $C_2$ can establish awareness with each other, which is represented by the bold, dashed edge in Figure 3.5 (b). Therefore, peers in two communities are still interconnected after $P$’s departure.
The above mechanism is only applicable to the situation that a peer leaves the system intentionally. However, in the case of exceptions such as computer crash, a peer may leave the workflow system quietly without notifying other peers. To deal with this problem and keep the system informed of a peer’s exceptional departure, each peer sends a status message to its neighbour peers periodically, indicating that the sender is active. If a peer has not been heard from for a period of time, it is assumed that this peer has encountered some exceptions and left the system accidentally. Thus, proper actions will be taken by the neighbour peers to remove the “dead” peer automatically. At the same time, an exception message would be generated by the coordinator peer of each community in which the leaving peer is involved, to notify system administrators and system engineers to handle problems such as information loss. In the situation that a coordinator peer leaves the system accidentally, the rest of the peers in the same community would negotiate together to determine a new coordinator peer. For example, the rest of the peers can select one of them randomly to be the new coordinator peer. Finally, when the “dead” peer recovers, it is treated as a new peer to join the system.

There could be some occasions that an existing peer changes the capabilities it has without leaving the system. For example, a workflow participant is trained with gaining a further capability or a workflow participant is found lacking skills with
regard to an original capability. This update results in an alteration of the organisational models of some other peers. The peers being affected need to modify their peer repositories accordingly to reflect the up-to-date organisational settings.

To update relevant peers’ organisational models, the changing peer sends out an updating message to all its neighbour peers, carrying information about the change. The neighbour peers then update their peer repositories accordingly by adding new capability information to the existing record, or deleting existing capability information from the existing record. When necessary, a peer may leave or join some specific communities according to the alteration. At the time when the changing peer needs to join a new community, it advertises itself and obtains the organisational information of the new community using the same mechanism as a new peer joins the system.

In summary, the peer registration service supports the peer’s dynamism through local adjustment and update. This service is also expected to enhance system scalability greatly. With the support of the peer registration service, new peers can easily join the system at any time and through any existing peer with no need to change the settings of a particular site.

3.4.3 Capability configuration service

The capability configuration service is a system service that constructs the capabilities, which are referred in workflow process models. As defined in Section 3.2, capabilities in SwinDeW are objects encapsulating a set of rules associated with functions of a workflow process. Normally, system administrators and system engineers are responsible for constructing these objects from the organisational model of the workflow application, and distributing them to the appropriate peers.

The construction of capability objects is normally carried out manually. System administrators and engineers interact with management of the organisation to obtain
information about the various roles in the organisation, including the rules for a particular role. This information is then used to construct the deployable capability objects. The details of capability construction are beyond the scope of this thesis.

When peers are initialised, they ultimately have to obtain their capabilities for system operation (i.e. the WfPS is not operable without a set of rules by which to work). The peers query system administrators and engineers for capability objects, according to the roles their associated participants play. Another usage of the capability configuration service is to change the capability settings of the initialised peers. In this case, system administrators and engineers interact with existing peers to indicate the new settings and pass the capability objects when necessary.

3.5 System initialisation

Workflow system initialisation is normally carried out centrally in client-server based workflow systems. Ordinarily, starting a workflow management system might involve system engineers and administrators who manually configure the system and start up the services as needed [Coo02]. A similar practice needs to be carried out centrally in SwinDeW as well, although the workflow system architecture is decentralised.

As mentioned in Section 3.4.3, before starting up the individual peers, system engineers and administrators must configure the capabilities of each peer in the system. The initialisation also assigns the resources and tools and/or the authorisation to use these resources and tools to various peers, in accordance with their capabilities. Moreover, during system initialisation, the initial peer repositories should be established appropriately for individual peers, in order to:

1. allow individual peers to have proper views of the organisational model,

2. form virtual communities initially,
(3) ensure connectivity amongst virtual communities, and

(4) enable the peer discovery service.

At the same time, some plug-in services such as the GUI-based process definition tool which is detailed in Chapter 8 need to be started up for particular peers.

To summarise, the system initialisation sets up the system initially to enable workflow support. This work is carried out centrally through the manual configuration or the use of deployment tools.

### 3.6 Summary

In this chapter, the novel design of the SwinDeW decentralised workflow system which is based on p2p computing technology has been presented. This design intentionally eliminates the centralised data repository and the centralised workflow engine which are used in conventional workflow approaches. Based on the p2p architecture, the SwinDeW system is built on top of a set of autonomous and collaborative peers. The unique design of the peer structure and the peer organisation enables peer autonomy and collaboration. Based on this design, the novel, decentralised peer management service and the essential, centralised system initialisation have also been discussed.

This system design provides a decentralised platform for workflow. Based on this design, the mechanisms for build-time and run-time workflow functions corresponding to the process definition service, the process enactment service and the monitoring & administration service are proposed in the following chapters.
Chapter 4

Build-time functions

Build-time functions of a workflow management system mainly regard workflow modelling, workflow representation, and storage of process definition data. In this chapter, the mechanisms supporting build-time workflow functions in SwinDeW, i.e., the process definition service in Figure 3.2, are presented.

First, key issues of process representation are discussed in Section 4.1. Then Section 4.2 describes a graph-based process representation which uses different symbols such as nodes and edges to represent key workflow components. In Section 4.3, a conceptual task notation is presented to uniformly represent individual tasks within the context of a process. Based on the graph-based process representation and the conceptual task notation, an innovative “know what you should know” data storage policy is presented in Section 4.4. This policy divides a process definition into task partitions (as discussed in Section 4.4.1), distributes each partition to the relevant peers for decentralised data storage (as discussed in Section 4.4.2), and maintains process definition data properly to support dynamic and scalable workflow applications (as discussed in Section 4.4.3).

4.1 Issues of process representation

A workflow process consists of a number of partially-ordered tasks to reach a goal. Each task comprises a logical, self-contained unit of work within the process definition [FH93]. A task forms one logical step within a process and represents
work which will be processed by a workflow participant. The particular human resource, i.e., the workflow participant whom should be assigned to perform a particular task is specified by an attribute of the task. Moreover, workflow tasks are interrelated with one another. For example, the start of a task may require that some (or all) preceding tasks have been completed, and the required data from other tasks are available. This relationship is represented by transition information [WfMC99].

There are two types of transitions: data transitions and control transitions. A data transition defines the application-related data dependencies between tasks. Because tasks may have typed in- and out-parameters, a data transition is used to pass intermediate results between tasks. On the contrary, a control transition defines the control dependencies between tasks and is used for evaluation in conditional expressions. Each transition has three elementary properties: the from-task, the to-task and the condition under which the transition is made. The information related to the transition condition is defined within the appropriate tasks.

In short, a process representation logically defines a network of tasks and their relationships, start and termination of a process, and information about the individual tasks [WfMC99]. This representation is also known as a workflow model which is used by workflow management systems to control the execution of workflow instances. The models to be used to define workflow models are known as workflow meta-models which define the concepts of workflow components and their relationships. Nowadays, there is an absence of a universal workflow meta-model which is generally agreed upon. A variety of meta-models are used today in different workflow management systems [LS97].

The two popularly used workflow meta-models are process graphs and the Petri nets-based meta-model. Process graphs specify a workflow process as a directed graph, whose nodes represent tasks and whose edge set is partitioned into a set of control flow edges and a set of data flow edges [AMGA+95, Mou98, Wes98, Yan00]. Control flow edges represent potential control flow, defined by control transitions. Data flow edges connect parameters of different tasks, defined by data transitions. Petri nets-based workflow meta-models also have a graphical nature [Aal96]. Both classical Petri nets and enhanced Petri nets (e.g., colour, time, and
hierarchy) use the formal semantics of Petri nets to model workflows [EG91, AHH94, EKR95, Aal02]. By using Petri nets, various workflow concepts are mapped onto Petri nets. For example, tasks are modelled by transitions, and conditions which are the states between tasks are modelled by places. Furthermore, the state of a process instance can also be represented explicitly in Petri nets.

On the other hand, workflow processes are defined formally in process definition languages for interoperability purposes. Again, there is a variety of process definition languages available at present, which use different data types, schemas and syntax to define processes formally [AHW03]:

- The XML Process Definition Language (XPDL) [WfMC02] is a language for describing workflow process definitions and also a grammar for the interchange of process definitions. The final draft of version 1.0 was released in October 2002 by WfMC.

- The Web Services Flow Language (WSFL) [IBM01] was proposed by IBM in May 2001. WSFL is an XML language for defining workflow processes within the framework of the Web services architecture composition.

- XLANG [Microsoft01] is a notation developed by Microsoft for the specification of message exchange behaviour among participating Web services. XLANG supports especially the automation of business processes.

- Business Process Execution Language for Web Services (BPEL4WS) [IBM03] is collectively proposed by researchers from BEA, IBM, Microsoft, SAP AG and Siebel Systems. BPEL4WS represents a convergence of the ideas in IBM WSFL and Microsoft XLANG specifications and supersedes them by combining and extending the functions of these previous foundation technologies. At the core of the BPEL4WS process model is the notion of p2p interaction between services described in Web Service Definition
Language (WSDL). Both the process and its partners are modelled as WSDL services.

- Process Specification Language (PSL) [NIST00] is developed by National Institute of Standards and Technology. PSL defines a neutral representation for manufacturing processes.

- Little-JIL [Wis98] is presented by the University of Massachusetts Amherst. Little-JIL is a graphical language for defining processes that coordinate the activities of autonomous agents and their use of resources during the performance of a task.

Moreover, from a data repository viewpoint, the representation of a process can be stored in different physical format, such as a file system, a special-purpose in-house (built-in) database system, an object-oriented database system, or a (commercial) relational database system [YZ97]. It is no doubt that how to represent a process, both logically and physically, is very important to the success of a workflow management system. Since the physical data repository is always determined at system implementation stage, this chapter only emphasises the logical process representation.

### 4.2 A graph-based process representation

To model application processes as workflows with the aim of controlling their execution, a suitable formalism has to be provided. The process representation in SwinDeW is based on process graphs with enhancement to represent more primitives identified by WfMC, mainly from a coordination viewpoint. This is because of the following two reasons. First, it is not the purpose for SwinDeW to invent yet another new process modelling approach. Second, graph-based representations of workflow models are simple and intuitive.
In general, graph-based process representation in SwinDeW uses nodes and edges to represent workflow elements such as tasks, control transitions, data transitions, and resource requirements of tasks, explicitly. As described below, there are three types of nodes, i.e., task nodes, control nodes and resource nodes, and three types of edges, i.e., data edges, control edges and resource edges, in this representation. Task nodes represent the set of tasks. Control conditions are separated from the tasks and form a set of control nodes. The human resources of the organisation are modelled as a set of resource nodes. Three types of edges represent the data transitions between tasks, the control transitions between tasks and the resource demands of tasks, respectively. Formally, a workflow process can be represented by a six-tuple, \( P = \langle TN, CN, RN, D, C, R \rangle \), conceptually, where:

- \( TN = \{t_1, t_2, t_3, \ldots \} \) is a set of task nodes. Each node in this set represents a piece of work that forms one logical step within a process. It is assumed in this chapter that all the tasks in a process definition are atomic. Atomic tasks represent single, executable and the smallest units of work which cannot be further decomposed. Modelling and representation of composite tasks will be discussed in Chapter 6.

- \( CN = \{c_1, c_2, c_3, \ldots \} \) is a set of control nodes which represent various logical relationships among tasks. Each control node has a control operator, which can be \( and, or \) or \( straight \), denoting different logics as discussed in Section 4.3.

- \( RN = \{r_1, r_2, r_3, \ldots \} \) is a set of human resource nodes. Each element in this set represents a unique resource type such as a capability.

- \( C \subseteq TN \times CN \cup CN \times TN \) is a set of arcs connecting elements in \( TN \) and \( CN \), indicating the relationships between elements in \( TN \) and \( CN \). The combination of \( CN \) and \( C \) represents the control dependencies between tasks.
• \( D \subseteq TN \times TN \) is a set of arcs connecting elements in \( TN \), indicating the data dependencies between tasks.

• \( R \subseteq RN \times TN \) is a set of arcs connecting elements in \( RN \) and \( TN \), indicating the human resource requirement of a task.

There are some constraints applied to this process representation as follows:

• \( \forall tn_j \in TN, \exists cn_k \in CN, \text{where } (tn_j, cn_k) \in C \lor (cn_k, tn_j) \in C \). Each task node should be connected to at least one control node.

• \( \exists tn_s \in TN, \forall cn_k \in CN, (cn_k, tn_s) \notin C \), \( tn_s \) is known as the starting task. It is assumed that there exists and exists only one task node which has no incoming arc from any control node. This assumption is justifiable because in the case that there is more than one entry point, an additional starting task can be generated as the preceding task of all the entry points with an And-Split structure.

• \( \exists tn_t \in TN, \forall cn_k \in CN, (tn_t, cn_k) \notin C \), \( tn_t \) is known as the termination task. Again, it is assumed that there exists and exists only one task node which has no outgoing arc to any control node. In the case that there is more than one exit point, an additional termination task can be generated as the succeeding task of all the exit points with an Or-Join structure.

• \( \forall tn_j \in TN, \exists rn_k \in RN, (rn_k, tn_j) \in R \). It is assumed that each (atomic) task can be executed by an agent, typically a person, or a software system or a person using a software system [RS95].

Each control node is a place of decision-making to determine how to route the process enactment threads. Given the different combinations of the control operator
and the number of incoming and outgoing arcs, a control node indicates different routing types, as summarised in Table 4.1.

Table 4.1 Summary of control routing

<table>
<thead>
<tr>
<th>Control routing type</th>
<th>Control operator</th>
<th>Number of incoming arcs</th>
<th>Number of outgoing arcs</th>
<th>Conditional routing?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential routing</td>
<td>straight</td>
<td>1</td>
<td>1</td>
<td>No</td>
<td>A segment where two tasks are executed in sequence under a single thread of execution.</td>
</tr>
<tr>
<td>And-Join routing</td>
<td>and</td>
<td>&gt;1</td>
<td>1</td>
<td>No</td>
<td>A segment where two or more parallel, executing activities converge into a single common thread of execution.</td>
</tr>
<tr>
<td>And-Split routing</td>
<td>and</td>
<td>1</td>
<td>&gt;1</td>
<td>No</td>
<td>A segment where a single thread of control splits into two or more threads which are executed in parallel within the workflow.</td>
</tr>
<tr>
<td>Control routing type</td>
<td>Control operator</td>
<td>Number of incoming arcs</td>
<td>Number of outgoing arcs</td>
<td>Conditional routing?</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------------</td>
<td>-------------------------</td>
<td>-------------------------</td>
<td>----------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Or-Join routing</td>
<td>or</td>
<td>&gt;1</td>
<td>1</td>
<td>Yes</td>
<td>A segment where two or more alternative activities re-converge to a single common activity as the next step within the workflow.</td>
</tr>
<tr>
<td>Or-Split routing</td>
<td>or</td>
<td>1</td>
<td>&gt;1</td>
<td>Yes</td>
<td>A segment where a single thread of control makes a decision upon which branch to take when encountered with multiple alternative workflow branches.</td>
</tr>
</tbody>
</table>

Based on this conceptual process tuple, process models are created using the following set of modelling operations:

- *CreateAtomicTask(i)*: Create an atomic task, \( i \), including the information about how to execute this task such as the required applications and tools, and a set of input- and output-parameters with their respective types.
- **DeleteAtomicTask(i):** Delete an atomic task, \(i\), from the process, which involves the deletion of edges adjacent to \(i\).

- **CreateControl(j,o):** Create a control node, \(j\), with a particular control operator \(o\).

- **DeleteControl(j):** Delete a control node, \(j\), which involves the deletion of edges adjacent to \(j\).

- **CreateResource(k):** Create a resource type, \(k\), including the definition of the unique capability associated with this type.

- **DeleteResource(k):** Delete a resource type, \(k\), which involves the deletion of edges adjacent to \(k\).

- **AddControlEdge(i,j,m):** Add an edge in the process, which connects two tasks, \(i\) and \(m\), via a control node, \(j\).

- **DeleteControlEdge(i,j,m):** In the process, delete an edge which connects two tasks, \(i\) and \(m\), via a control node, \(j\). This involves the deletion of \(j\) if \(j\) has no other adjacent edges.

- **AddDataEdge(i,m):** Add an edge, \(i \rightarrow m\), in the process, which connects an output-parameter of task \(i\) to an input-parameter of task \(m\).

- **DeleteDataEdge(i,m):** Delete an edge, \(i \rightarrow m\), in the process.

- **AddResourceEdge(k,i):** Add an edge, \(k \rightarrow i\), in the process, which assigns the required human resource \(k\) to task \(i\).

- **DeleteResourceEdge(k,i):** Delete an edge, \(k \rightarrow i\), in the process.
To make this process representation easy to understand, the representation of a process is viewed as an acyclic graph [YZ97]. Various symbols are used to represent different elements in a workflow process, as described in Figure 4.1.

Figure 4.1 Constructs of process representation

Figure 4.2 Visualised representation of a workflow process

Figure 4.2 denotes a visualised representation of a workflow process. This process consists of six individual tasks, each requiring a particular human resource. $tn_1$ is the starting task because $tn_1$ does not have any preceding control node. $tn_6$ is the termination task because $tn_6$ does not have any succeeding control node. It is also clear from this figure that tasks should be enacted in a certain order. Task $tn_2$ and $tn_4$ should be executed after task $tn_1$ with an And-Split routing structure. Then task $tn_4$ follows $tn_2$ and $tn_3$ with an Or-Join routing structure. Finally, $tn_5$ follows $tn_4$ and $tn_6$ follows $tn_5$ sequentially. At the same time, all the adjacent tasks have data dependencies. For example, the output-parameter of $tn_4$ is the input-parameter
of \( m_5 \). Moreover, a resource type is assigned to each task node in order to fulfil its resource demand. For instance, resource type \( r_{n_i} \) is assigned to task nodes \( m_{n_2} \) and \( m_{n_6} \).

Formally, the process in Figure 4.2 can be represented using the six-tuple \( P = \langle TN, CN, RN, C, D, R \rangle \), where:

\[
TN = \{m_1, m_{n_2}, m_{n_3}, m_{n_4}, m_{n_5}, m_{n_6}\}
\]

\[
CN = \{c_{n_1}, c_{n_2}, c_{n_3}, c_{n_4}\}
\]

\[
RN = \{r_{n_1}, r_{n_2}, r_{n_3}\}
\]

\[
C = \{(m_1, c_{n_1}), (c_{n_1}, m_{n_2}), (c_{n_1}, m_{n_3}), (m_{n_2}, c_{n_2}), (m_{n_2}, m_{n_3}), (m_{n_3}, c_{n_2}), (m_{n_3}, m_{n_4}), (c_{n_3}, c_{n_4}), (c_{n_3}, m_{n_5}), (c_{n_4}, m_{n_5})\}
\]

\[
D = \{(m_1, m_{n_2}), (m_{n_1}, m_{n_3}), (m_{n_2}, m_{n_4}), (m_{n_3}, m_{n_5}), (m_{n_4}, m_{n_5})\}
\]

\[
R = \{(r_{n_2}, m_{n_1}), (r_{n_1}, m_{n_2}), (r_{n_2}, m_{n_3}), (r_{n_3}, m_{n_4}), (r_{n_3}, m_{n_5}), (r_{n_1}, m_{n_6})\}
\]

### 4.3 Task notation

Although all the tasks in a process are somewhat interrelated, it is observed that in an ideal process, each task is concerned only with its context, i.e., its direct predecessors and successors. Before its enactment, an individual task only interacts with tasks whose outcome directly affects the start condition of this task, e.g., those tasks which produce the required data for this task and those tasks which logically need to be completed before the start of this task. Similarly, after enactment, an individual task only interacts with those tasks which depend on the outcome of this task, e.g., those tasks which require this task to provide data to them and those tasks which logically should start after the completion of this task. Therefore, an individual task, together with its context, can be extracted from the process definition to form an independent fragment.
As shown in the dashed rectangle in Figure 4.3, an individual task, together with its context, forms an independent fragment, which can be separated from the whole process. With the information about this fragment, the related task can be executed properly. From a coordination viewpoint, each task fragment can be specified with a six-tuple:

\[ T(\text{Process-id}; \text{Task-id}; C_{pre}; C_{post}; \text{Capability}; \text{Specifics}) \]

- **Process-id**: process identifier which identifies a workflow uniquely.
- **Task-id**: task identifier which identifies a task uniquely in the context of a workflow.
- **C_{pre}**: precondition (taskset1, logic). The relationship between the task and its predecessors where:
  - taskset1: ((taskid1, capability1, input data1), (taskid2, capability2, input data2) | C_{pre} | null)). Taskset1 represents a set of preceding tasks with corresponding input data.
  - logic: (and, or, straight). By combining either an incoming or an outgoing flow with logic, a variety of types of logic operators are produced. The operators produced are and-in, and-out, or-in, or-out,
straight-in and straight-out. Of these, and-in and or-in are types of convergence, and-out and or-out are types of branching, and straight-in and straight-out indicate sequential structure. On one hand, those logic operators related to the incoming flow will determine the start condition of this task. On the other hand, those logic operators related to the outgoing flow sometimes invoke a decision-making process.

- \( C_{\text{post}} \): post condition \((taskset2, \text{logic})\). The relationship between the task and its successors where:

  \[
  \text{taskset2: } ((\text{taskid1, capability1, output data1}), ((\text{taskid2, capability2, output data2}) \mid C_{\text{post}}, \mid \text{null})).
  \]
  Taskset2 represents a set of successive tasks with corresponding output data.

- \textbf{Capability}: the participant’s capability that is necessary to fulfil the task.

- \textbf{Specifics}: the specific about the task, which may include information about how to execute this task.

Clearly, this notation defines the details of a task, its control and data dependencies with other tasks, as well as the human resource requirement. This notation is independent of any definition languages and data repository format. That is, a task specification using this notation can be obtained from a process defined in any language and the real data can be stored in any data format. By these means, a process specification can be converted into a list of individual task partitions. An approach for this conversion is addressed in the next section. Later, with the unique process identifier, task partitions can be reunited to form a complete process.

### 4.4 Process definition service

According to Figure 3.2, the process definition service in SwinDeW provides support for build-time functions. A process in SwinDeW is defined by a peer known
as a *definition peer*. Normally, a definition peer is associated with an authorised participant with process modelling capability such as process-engineering. A process is defined through the use of a process definition tool. The key elements identified in Section 4.2 are described in a process definition clearly, which provides a template for workflow instances. In SwinDeW, the process definition data should be accessible to workflow participants because they are in charge of passing the artefacts and scheduling the various task instances for execution. Therefore, where to store the process definition data in a decentralised workflow system is a critical issue which needs to be addressed.

In general, three prevalent mechanisms are used to support process data storage in distributed and/or decentralised workflow systems.

1. The workflow system still retains a centralised data repository. Workflow participants need to access a single site to retrieve the data. For example, most of the Web-based distributed workflow systems use a single HTTP server to store data. This kind of approach is easy to implement, as the client-server architecture is considered a mature model to suit the needs. However, as client nodes have no process knowledge at all, they are restricted and can only participate in workflow systems passively. In addition, since the server is retained in the system, the problem of single-source failure remains.

2. The workflow system replicates all the process definition data at each node, on the assumption that definition data is rarely updated but intensively accessed. For example, the system proposed in [HY98] uses this storage approach. However, it is evident that replicating the complete workflow definition on each peer is resource-consuming and error-prone for consistency maintenance. As all the nodes have full data, problems like lack of security are also incurred. Moreover, due to the frequently changing
environment and the increasing size of the system, this kind of approach is clearly impractical and unnecessary.

(3) Another way to store data is migrating data from one place to another either by agents or multiple workflow servers. Normal nodes do not have to keep data at all. Process definition data are transferred to the right places when necessary. For instance, DartFlow [CGN96] uses transportable agents to carry essential process data. However, this kind of approach always requires complicated communication languages and complex algorithms to control data migration which are normally expensive and inflexible.

Since there is an intentional absence of a centralised data repository in SwinDeW, as mentioned in Chapter 3, the first mechanism is inapplicable. The other two mechanisms also have limitations as indicated above. More importantly, an individual peer in SwinDeW normally needs to be concerned with only part of the process, in correspondence with the capabilities of the associated participant. With respect to this philosophy, SwinDeW presents an innovative “know what you should know” policy, which requires a peer to obtain only essential knowledge of a process [YYR03a]. Based on this policy, a workflow is partitioned into individual tasks after it is modelled, and then individual tasks are distributed to relevant peers properly. The peers are able to manage and maintain the received data to perform workflow functions, even in a dynamic environment. Therefore, a key part of the process definition service is to control process partition, data distribution and data maintenance.

4.4.1 Partitions of a process

After a process is completely defined by a definition peer, the first concern of the presented data storage approach is to divide a process into individual task partitions, using the task notation presented in Section 4.3. Each task partition contains the complete information about a task so that the peers that receive this partition are able to enact the derived task instances accordingly.
Given a process which is specified with the conceptual process representation \( P = \{TN, CN, RN, C, D, R\} \), each task node in \( TN \) can be denoted with the task notation \( T (\text{Process-id}; \text{Task-id}; C_{\text{pre}}; C_{\text{post}}; \text{Capability}; \text{Specifics}) \).

Of the six elements in the task notation, \( \text{Process-id}, \text{Task-id} \) and task \( \text{Specifics} \) are attributes and information about the task itself. These three elements can be derived from the process definition directly, although they are not described in the conceptual process representation explicitly. For the remaining three elements, the pre-condition, \( C_{\text{pre}} \) and the post-condition, \( C_{\text{post}} \) of a task are generated from the data and control dependencies among the task nodes, and the \( \text{Capability} \) attribute is obtained from the human resource requirement of the task. The following conversion algorithm is developed.

\[
\text{For each } tn_i \text{ in } TN
\]

\[
\text{a new } T_i \text{ is generated as } T_i(\text{p-id}_i; \text{t-id}_i; C_{\text{pre}}^i; C_{\text{post}}^i; \text{capability}_i; \text{specifics}_i)
\]

where

\[
C_{\text{pre}}^i = \begin{cases} 
\{(t_{j}(r_{n_j}, d_j))(m_{j}, c_{n_j}) \in C \land d_{j} = (m_{j}, t_{n_j}) \in D \land (r_{n_j}, t_{n_j}) \in R\} & \text{if } \exists (c_{n_j}, t_{n_j}) \in C \\
\text{null, otherwise} & 
\end{cases}
\]

\[
C_{\text{post}}^i = \begin{cases} 
\{(t_{k}(r_{n_k}, d_k))(c_{n_k}, t_{n_k}) \in C \land d_{k} = (m_{k}, t_{n_k}) \in D \land (r_{n_k}, t_{n_k}) \in R\} & \text{if } \exists (t_{n_k}, c_{n_k}) \in C \\
\text{null, otherwise} & 
\end{cases}
\]

\[
\text{capability}_i = r_{n_k} \text{ where } (r_{n_k}, t_{n_k}) \in R
\]

Next

This conversion algorithm is processed centrally by the definition peer. By executing this algorithm, a process specification is divided into a list of task partitions properly without losing information.
4.4.2 Distribution of data partition

Normally, the definition peer initialises the distribution immediately after a process is defined and partitioned. As each partition represents a template of future work rather than work itself, the process distribution is not the procedure of work assignment. A peer does not commit to any work by receiving a task partition. Alternatively, a task partition contains information about the work that the recipients can do and possibly will do in the future. Therefore, a task partition normally is not distributed to a single peer, but is distributed to all the relevant peers that are associated with the participants who are capable of performing the work derived from this task partition.

On one hand, as indicated in Section 4.3, each task partition has a capability attribute indicating the human resource required to carry out future instances which are derived from this task template. Only the participants with the required capability will possibly undertake the instances of this task. On the other hand, as discussed in Section 3.3.2, peers are clustered to form virtual communities labelled by their capabilities indicating what types of work they can do. Obviously, the partition distribution could use a capability-match mechanism. Each task partition is expected to be delivered to the peers in a virtual community whose capability character corresponds to the capability attribute of the task partition.

![Figure 4.4 Distribution of a task partition](image)

Figure 4.4 Distribution of a task partition
As shown in Figure 4.4, the following steps illustrate the distribution procedure of task partition $T_i$ with capability $\text{Capability}_i$ by definition peer $P_i$.

1. $P_i$ invokes the peer discovery service automatically to find a peer with capability $\text{Capability}_i$,

2. Suppose that $P_j$ is returned as the result of this service invocation, i.e., $P_j$ is a peer with capability $\text{Capability}_i$,

3. $P_i$ issues a synchronisation request to $P_j$ for confirmation,

4. If $P_j$ is an active peer, $P_j$ acknowledges this synchronisation request, otherwise go back to step (1),

5. Task partition $T_i$ is distributed from $P_i$ to $P_j$,

6. $P_j$ propagates $T_i$ within the virtual community to those other peers with capability $\text{Capability}_i$,

7. End.

This data distribution mechanism is simple and functional. The definition peer only distributes a task partition to one relevant peer and can start distributing the next partition thereafter. The data distribution within the virtual community is taken over from the definition peer, which reduces the cost of data transfer. By these means, the task partitions are distributed appropriately one-by-one. As a result, normally none of the peers has global knowledge about the process\(^1\). Each peer acquires only the relevant knowledge about the process in accordance with its capabilities and the contribution to this process. This is the reason why this storage policy is named “know what you should know”.

---

\(^1\) The definition peer and the peers associated with system administrators and engineers may have global knowledge about the process as a backup.
4.4.3 Data maintenance

All previous discussion is based on an assumption that the workflow management system is static with all the workflow participants (represented by the associated peers) remaining unchanged after the system is started up. However, as discussed in Section 3.4.2, the workflow system is dynamic and scalable with peers joining and departing at times. The impact of a peer’s dynamic behaviour on the system’s organisational model has been discussed in Section 3.4.2. Moreover, as workflow participants need to access the right part of process definition to carry out workflow functions, a peer’s behaviour may also influence the process definition data that this peer should have. For example, a new peer should obtain essential process definition data when it joins the system in order to be able to perform corresponding functions, and an existing peer needs to retrieve new process definition data when it gains a new capability. As a result, the process repository of a peer should be maintained accordingly when this peer behaves dynamically. Regarding different peer behaviour, i.e., peer joining, peer departing and peer updating, various actions need to be taken to maintain the process definition data properly.

When a peer joins the system through joining the corresponding communities one-by-one, it obtains the corresponding process definition data one-by-one from the existing peers to generate its process repository. Once accepting a new peer, the coordinator peer of a community transfers the process definition data which are the task partitions requiring the capability of this community to the new member. Thus, the new peer has all essential information about the potential work it will take. With this information, the peer can take part in the workflow system and perform some basic functions such as accepting a task instantiation request immediately.

When an existing peer leaves the system, the process definition data are no longer used. There is no need for a leaving peer to retain the process definition for future use because the definition may be modified during its absence even if this peer rejoins the system later. Therefore, a peer simply deletes the process definition data stored in its process repository when it leaves the system. In the future, the data
can be re-obtained when the peer rejoins the system. In summary, a peer’s departure behaviour will not cause process data maintenance problems.

When a peer changes the capabilities it has, the process definition data which are stored in this peer’s process repository need to be changed accordingly. On one hand, if a peer abandons a capability, the process data related to this capability become useless and should be deleted. On the other hand, if a peer acquires a new capability, the process definition data in relation to this new capability should be obtained. In other words, departure from a particular community leads to the deletion of unnecessary process definition data. In addition, at the time when the changing peer needs to join a new community, the peer requires the new process definition data using the same mechanism as a new peer joins the system.

4.5 Summary

In this chapter, the mechanisms supporting the build-time functions of SwinDeW have been presented. A graph-based process representation, which can be visualised as an acyclic graph, has been given to represent the key elements of a process and the relationships among them. A task notation has also been developed to describe tasks uniformly within a process. Based on this task notation, a unique process storage policy called “know what you should know” has been developed to divide process definition data into task partitions and distribute task partitions to relevant peers. Moreover, the corresponding mechanisms for maintaining process definition in the dynamic environment have been discussed.

The “know what you should know” data storage policy contributed in this chapter naturally reflects a fact that each participant involved in a workflow system knows neither everything nor nothing at all. This policy also reflects one characteristic of p2p — no peer has global knowledge (except some management-related peers). In addition, the advantages of the mechanisms proposed in this chapter also include simplicity, low cost and flexibility. The process representation is natural and effective. The communication protocols are simple and functional.
Data partition and distribution are independent of any data repositories and can be realised without using any external tool. Moreover, virtual communities are utilised to reduce the cost of data transfer and simplify data maintenance, as the distribution within a community can be taken over from the definition peer. Finally, since the centralised data repository is removed from the system and since the data is shared locally, this approach also increases resilience to single site failures.
Chapter 5

Run-time functions

Run-time functions of a workflow management system mainly deal with creation and execution of workflow instances. In this chapter, the mechanisms supporting run-time workflow functions, i.e., the process enactment service and the monitoring & administration service in Figure 3.2 are presented.

First, the procedure of decentralised process instantiation in SwinDeW is described in Section 5.1. Then in Section 5.2, the mechanisms for instantiating tasks of different structures are discussed. Based on these mechanisms, Section 5.3 proposes three dynamic work allocation policies. After that, the coordination mechanism for instance execution is described in Section 5.4. Finally, the monitoring and administration of workflow execution is presented in Section 5.5.

5.1 Process instantiation procedure

The relationship between a process definition and a process instance in workflow is the same as the relationship between a class and an object in object oriented computing. A process definition represents a workflow process and its constituent tasks. While a process instance is derived from a process definition and represents a single enactment of the process, using its own process instance data. For example, the processing of one insurance claim is an instance of the insurance claim workflow. For each invocation of the process, a process instance is created, managed and eventually terminated by the workflow management system [WfMC95]. Because a
process definition consists of a network of tasks, a process instance also consists of a network of task instances. Similarly, each task instance represents a single invocation of a task, relates to exactly one process instance and uses the process instance data associated with the process instance. Therefore, the procedure of process instantiation creates the relevant task instances and binds individual data to the task instances. Moreover, as mentioned in Section 4.2, each task in the process definition requires a human resource to fulfil the work represented by this task, which is represented by the resource edge in the graph-based process representation. This human resource requirement in a task instance is realised by allocating the task instance to a particular workflow participant with the required resource (capability). Hence, to enact a process instance, various task instances need to be allocated to workflow participants properly.

In client-server based workflow management systems, various task instances are normally created centrally on the server side because only the server knows about the process definition. Then various task instances are presented to the workflow participants via the work lists. As a result, assignment of task instances, i.e., work allocation, is pre-defined and static, which lacks flexibility [Mas02]. However, in a decentralised workflow environment like SwinDeW, a process instance cannot and should not be created at a single site because each peer only has local knowledge about the process. More importantly, “who does what” is not necessarily pre-defined and may be more dynamic. In SwinDeW, the required human capability for a task is described explicitly in a process definition. Therefore, a task instance and the associated data could be packaged into a virtual “envelope” indicating the capability required. Such an envelope is then assigned to a peer (on behalf of a workflow participant with the required capability) dynamically, based on the capability match.

The mechanism discussed in this section proposes a collaborative instantiation approach and supports dynamic work allocation. In short, this mechanism dispatches task instantiation requests to a variety of virtual communities, requesting capable peers to accept various task envelopes dynamically [YYR03b]. There is no centralised peer that dispatches all the instantiation requests. Instead, the requests to instantiate various tasks are issued by different peers. That means the instantiation of
tasks is decentralised. The whole process instance is eventually created through the instantiation of relevant tasks one-by-one by various peers. Figure 5.1 shows the procedure of instantiating a sample process which consists of five tasks. Task T₁ which is the starting task is instantiated first by peer P₁. Then P₁ seeks another peer, e.g., P₂, to instantiate task T₂ which is the direct succeeding task of T₁. P₂ instantiates T₂ and repeats the same work to seek peers to perform the direct succeeding tasks of T₂, i.e., tasks T₃ and T₄. Eventually, the whole process instance is created. Various task instances are created and will be performed by P₁, P₂, P₃, P₄, P₅, respectively.

(1) P₁ instantiates T₁ and issues a request to instantiate T₂

(2) P₂ instantiates T₂ and issues requests to instantiate T₃ and T₄

(3) P₃ and P₄ instantiates T₃ and T₄ respectively and issue requests to instantiate T₅

(4) P₅ instantiates T₅, and the process instantiation is completed

Figure 5.1 Example of process instantiation procedure
In general, the steps to create a process instance is summarised as follows:

(1) A starting task instance is created by a peer under the guidance of management (e.g., being instructed by a manager) or as a response to a coming event (e.g., receipt of a claim form in an insurance claim workflow). Associated instance data are also bound to this instance. For example, when a workflow instance is created for a new insurance claim, the information about this claim is treated as the instance data. The peer that creates the instance of the starting task also commits to performing the work represented by this task instance. The instantiated task is known as the current instantiation task and the peer which creates this instance is known as the current instantiation peer.

(2) If the current instantiation task has one or more direct succeeding tasks\(^2\), these succeeding tasks are then required to be instantiated by various peers. The work represented by these task instances are also accepted by the peers which instantiate them. The details of task instantiation and work allocation are presented in Sections 5.2 and 5.3. The peers which instantiate these succeeding tasks are known as the successor peers of the current instantiation peer. On the contrary, the current instantiation peer is known as the predecessor peer of the selected peers which instantiate the succeeding tasks.

(3) The newly-instantiated tasks act as the current instantiation task and repeat step (2), either in parallel or one-by-one.

(4) If there is no current instantiation peer, process instantiation is completed.

From the above description, it is clear that process instantiation is carried out in a decentralised manner through the direct communication and collaboration amongst

---

\(^2\) The succeeding tasks of the current instantiation task can be obtained from the process repository of the current instantiation peer.
peers. Tasks are instantiated one-by-one from the starting task to the termination task. At the same time, the procedure of process instantiation is also the procedure of work allocation. The work represented by a task instance is automatically assigned to the workflow participant associated with the peer that creates this task instance. Eventually, the outcome of process instantiation is a network of relevant peers on behalf of workflow participants performing various task instances. Again, instance data are stored in a decentralised manner. A task instance, after its creation, is inserted into the peer’s task repository, waiting to be scheduled for execution.

In summary, what has been discussed so far is a mechanism for fulfilling process instantiation in a decentralised way, which is conventionally done centrally. This mechanism allows different tasks to be instantiated by various peers at dispersed locations. Tasks are instantiated from the starting task to the termination task through the direct interactions between relevant peers.

5.2 Task instantiation

The key step of the process instantiation mechanism discussed in Section 5.1 is Step (2), which concerns instantiation and allocation of the tasks in order. Before Step (2), the current instantiation task is instantiated by the current instantiation peer. It is the duty of the current instantiation peer to request the instantiation of the direct succeeding tasks of the current instantiation task. Each instantiation request is issued by the current instantiation peer and dispatched to capable peers, seeking one of them to accept this request. The capable peers of a task instantiation request are those peers associated with workflow participants who have the required capability to carry out the task, i.e., those peers in the virtual community labelled with the required capability. For the current instantiation peer, because the capability requirements of the succeeding tasks are also stored in its process repository, the discovery of the capable peers can be achieved through the invocation of the peer discovery service. The acceptance of a task instantiation request by a peer means this peer will create an instance of the requested task and undertake the work. For a current instantiation task, there are three types of structures to control the order in
which the succeeding tasks are executed, i.e., sequential, branching and convergent structures, as described in Section 4.2. Correspondingly, there are different procedures for peers to instantiate tasks:

![Diagram of task instantiation in a sequential structure](image)

Figure 5.2 Task instantiation in a sequential structure

- The sequential structure denotes the procedure where tasks are executed in order. In a sequential relationship, a particular task only has one direct succeeding task, which is executed in order after the completion of this task. Assume an instance of task \( T_i \) with definition \( T_i (\text{Process-id}; \text{Task-id}; C_{\text{pre}}; ((T_j, \text{Capability}_j, output_i), \text{straight}); \text{Capability}_i; \text{Specific}_i) \) is created by peer \( P_i \). From this definition, it is clear that \( T_j \) is the only succeeding task of \( T_i \), and \( T_i \)'s output-parameter \( output_i \) is passed on from \( T_i \) to \( T_j \) as \( T_j \)'s input-parameter. To create a task instance of \( T_j \), \( P_i \) needs to coordinate with other relevant peers. First, the peer discovery service is activated by \( P_i \) to seek a peer with the required capability to perform task \( T_j \), i.e., \( \text{Capability}_j \). As a result of this service invocation, a peer, e.g., \( P_k \), is returned to \( P_i \). \( P_i \) then sends an instantiation request (IR) to \( P_k \), which carries instance-related information such as when the task needs to be executed. However, \( P_k \)'s receipt of this request does not mean that this task instance is allocated to \( P_k \). The request needs to be broadcast in the corresponding virtual community (labelled with \( \text{Capability}_j \)) to let other capable peers know about this work. Then peers in this virtual community negotiate automatically to determine who will carry out this task instance eventually, normally based on dynamic work allocation policies which will be described in Section 5.3. Finally, \( P_j \),
the peer selected to accept $T_j$, sends a confirmation to $P_i$, creates an instance of $T_j$, and restarts the instantiation procedure to find its own successor peers. Figure 5.2 describes task instantiation procedure in a sequential structure.

- The branching structure denotes the procedure of splitting a task thread into multiple task threads, which are executed in parallel, or the procedure of making a decision about which branches to take when encountering multiple choices. In this case, task $T_i$ ($Process-id_i; Task-id_i; C_{pre}; C_{post}; Capability_i; Specifics_i$) has more than one succeeding task in its $C_{post}$. During the instantiation stage, each succeeding task should be instantiated. Suppose task $T_i$ has been instantiated by peer $P_i$. Each task in the $C_{post}$ of $T_i$ will then be instantiated with the mechanism for sequential structure, as described above. After that, the peers that are selected to instantiate those succeeding tasks of $T_i$ restart the instantiation procedure to seek their own successor peers.

- The convergent structure denotes the procedure of moving some parallel task threads towards one point and triggering the execution of a single task thereafter, either synchronously or asynchronously. In this case, more than one peer seeks a single peer to create and perform the instance of their common succeeding task $T_i$ ($Process-id_i; Task-id_i; C_{pre}; C_{post}; Capability_i; Specifics_i$). As a result, the peers in the virtual community which is labelled with $Capability_i$ may receive multiple requests to instantiate the same task. Each request is issued independently by a peer performing a preceding task instance. To deal with this situation, the peers in the virtual community store the received requests in their task repositories temporarily and postpone their negotiation to the time when all the requests from the preceding peers are received. Then all the requests are considered and balanced to reach an agreement upon which peer will accept the requests to create an instance of $T_i$. Again, the selected peer restarts the instantiation procedure to continue the instantiation of the succeeding task(s).
In summary, with respect to different routing structures in a process, SwinDeW uses different means to create task instances. This procedure is realised through the automated negotiation among relevant peers.

5.3 Dynamic work allocation

As described in the preceding section, work allocation is dynamic in SwinDeW. It is up to the peers in a virtual community to negotiate automatically about who will actually accept an instantiation request in order to create and perform an instance of a task. By these means, the selection computation can be distributed within the community rather than be congregated at the initiator. In addition, negotiation rather than designation can push the workflow participants into a more active working style. Ordinary workflow participants are able to adjust some configurations such as preference indexes which will be discussed in Section 5.3.3 to gain some control over task allocation. In order to select a peer to create and perform a task instance, the competent candidates for a particular task instance need to be identified. These candidates are peers that are capable of executing the requested task instance, i.e., those peers that have the matched capability to undertake this task instance and meet all the other special requirements of this task instance like execution time. In terms of the availability of the competent candidates, there are three possibilities as follows:

1. absence of any candidate, which means no peer can accept this task instance request at the time,

2. only one candidate, and

3. more than one candidate.

The first possibility is an erroneous situation and needs to be handled through the monitoring and administration service, which is always related to human intervention. For example, existing human resources can be adjusted or new workflow participants can be recruited to make the competent peers available. This
situation is detected by the coordinator peer of the virtual community and reported to the peers associated with system administrators and engineers, known as the administrator peers. The second possibility is easy to handle. Since there is only one competent peer, the task instance will no doubt be assigned to this peer. The rest of this section discusses the third possibility on how to select an instance creator and performer from a set of competent candidates to which the allocation policy needs to be applied. A good allocation policy always enables a workflow system to utilise its human resources effectively. It is also expected that a good allocation policy helps to balance load, optimise performance and satisfy workflow participants. To achieve these goals, SwinDeW supports three allocation policies which are applicable under different circumstances.

5.3.1 Allocation policy based on round-robin scheduling

The first policy is based on round-robin scheduling [FSP00], which is one of the oldest, simplest, fairest, and most widely used scheduling algorithms. In a virtual community, all the peers are placed in a queue. When a new instance request comes, the first competent peer in the queue is picked to take it, and then this peer is added to the tail of the queue. The peers after the selected one in the queue are pushed one position forwards in the order automatically for the next request. Therefore, the task instances will be equally assigned to the peers in a virtual community in turn. This policy is proven to be very effective in the circumstance that all the peers in the same community have equivalent capabilities and all the task instances have equal workload [SYWS+03].

5.3.2 Allocation policy based on workload

However, instances of different tasks always have different workload. In some cases, even various instances of the same task have different workload. For example, in a home loan processing system, the workload of checking the applicant’s financial status varies case-by-case. The round-robin based allocation policy cannot reflect the difference in the workload very well. Peers taking the same number of task instances may display significant differences in terms of real workload. Thus,
this second allocation policy takes the workload into account and is based on the idea that balancing the workload may achieve better system performance. In short, this policy calculates the workload of each peer and assigns a task instance to the peer with the least load for the time being. To calculate the load mathematically, the workload of a task instance is measured as the amount of time to accomplish this instance, which is either estimated or recorded.

To optimise the system, all the competent peers are identified first. Then $w_i$, the workload of each competent peer $i$ in a time period such as one day is calculated using the formula:

$$w_i = \sum_{k=1}^{N(i)} t^k_i$$

where $t^k_i$ is the workload of task instance $k$ that has been assigned to peer $i$ and $N(i)$ is the total number of task instances that have been assigned to peer $i$ in this time period. For a task instance which has been fulfilled, its workload is the recorded actual time to carry out this instance. For a task instance which has not been fulfilled, its workload is the estimated time required to carry out this instance. The estimated value is normally defined at build-time when the workflow process is modelled, using historical experiences for reference. Eventually, the task instance is assigned to peer $k$ with minimum $w$.

The objective of this algorithm is to bring the workload into proportion among the competent peers on the basis of the current condition. By always assigning work to the peer with the least load, the coexistence of idle peers with overwrought peers could be avoided on a local scale, i.e., within a virtual community. Thus, system performance can be optimised locally.

Moreover, peers in a virtual community always have different capabilities and play different roles in the overall system. It often happens that some key tasks require some capabilities associated with high-level skills that only belong to a small
number of participants such as managers. These more capable people usually can perform some low-level tasks as well. If the allocation policy is based on balancing individual workload only, these people may be busy with performing low-level tasks at a time because they may have the least workload when a low-level task instance comes. In this case, when a key task arises, it could be the case that no peer associated with these participants can accept it because all of them are engaged. Thus, possibility one where candidate is absent occurs. The whole process instantiation is blocked and the global performance may be degraded. Therefore, it is required that workload of some other peers may also be considered when a peer calculates its workload.

To optimise system performance globally, the performance bottleneck of the whole system should be identified and relieved. Given a workflow system with \( n \) virtual communities, it can be viewed that the task instances are assigned to different communities according to the capability attributes, and are taken by various participants involved in these communities. Thus, the overall system performance is determined by the community with the heaviest workload. To a particular community \( i \) with \( M(i) \) members, various members may have different workload. The mean workload of this community, \( \bar{w}_i \), in the current time period can be calculated as:

\[
\bar{w}_i = \frac{\sum_{k=1}^{M(i)} w_k}{M(i)}
\]

where \( w_k \) is the workload of member \( k \) in the community. Therefore, the community with the heaviest mean workload becomes the bottleneck of system performance, i.e., in a workflow system with \( n \) communities, the global performance is determined by:

\[
\max_{i \in (1, n)} (\bar{w}_i)
\]
To each peer $i$ in the system, its maximum workload $w'_i$ is defined as the maximum value of the mean workload of the communities it is involved in:

$$w'_i = \max_{j \in \text{Comm}(i)} \left( \overline{w}_j \right)$$

where $\text{Comm}(i)$ is the set of communities in which peer $i$ is involved.

Based on the above analysis, the following algorithm is designed to select a peer to accept a coming task instance with workload $w$. This algorithm considers the philosophy that for a peer involved in more than one community, the assignment of a task instance increases the mean workload to all the communities in which the peer is involved.

```
for each competent peer $j$, re-calculate its maximum workload if it accepts
the coming task instance

\[ w'_j = \max_{i \in \text{Comm}(j)} \left( \overline{w}_i \times \frac{M(i)}{M(i)} \right) + w \]

next
assign the task instance to $k$ with minimum $w'$
```

Again, in this algorithm, $M(i)$ is the number of peers in community $i$, $\overline{w}_i$ is the current mean workload of community $i$, and $w$ is the workload of the coming task instance. Obviously, every time when a new task instance needs to be created, this algorithm seeks a peer to accept the instance and pursues the lowest $\max(\overline{w}_i, i \in [1, n])$ after allocation, i.e., the best performance of the whole system. For a capable peer which has both high- and low-level capabilities, the coming work requiring a low-level capability may not be assigned to it even if it has the least workload.
In the case that more than one peer has equal workload at a time, some extra policies can be used to ensure that work is dispatched in an average manner. For example, for each peer, the last time when the work was assigned can be recorded. If two or more peers have equal workload, the present work is allocated to the peer that has not been assigned a task instance for the longest time.

5.3.3 Allocation policy based on weighted workload

In selecting a participant to perform work, both round-robin and workload balance based allocation largely ignore human factors such as user working habits, styles and preferences. In the real world, a workflow participant who is capable of performing multiple types of tasks may have preference for one task over another. When multiple instance requests come, a workflow participant prefers to take them in the order of preferences if possible. A task instance with a particular load is relatively less effort for a participant who prefers to do it, while the task instance could be relatively heavier for a participant who is forced to take it. Therefore, allowing workflow participants to possibly undertake tasks which suit them the most would contribute to human satisfaction and inspire human enthusiasm.

To respect the above preference for tasks, for a workflow participant represented by peer $i$, a preference index $p_i^k$ is given to each type of task $k$ that the participant can perform. Let $n$ be the number of types of tasks a workflow participant can carry out. Workflow participants are able to pre-define and update their own preference indexes which are bounded by the following two constraints:

Constraint 1: $0 < p_i^k \leq 1 \quad k \in (1, n)$

Constraint 2: $\sum_{k=1}^{n} p_i^k = 1$

The first constraint indicates that for a certain type of task, the bigger the preference index, the more preferable is this type of task to the participant. At the same time, the second constraint is used to avoid unreasonable preferences, that is, workflow participants cannot just decrease the preference indexes for all the tasks.
they can perform. If the preference index for one type of task is decreased, the preference index for some other types of tasks must be increased.

When a request asking for creating and performing the task instance $j$ comes, in addition to calculating the real workload of each competent peer, the preference index of this task is also taken into consideration. Let $N(i)$ be the total number of task instances assigned to peer $i$ in the time period, $t^i_k$ be the workload of task instance $k$ that has been assigned to peer $i$, and $t^j_j$ be the workload of task instance $j$. Thus, the preferable workload, or the weighted workload of peer $i$ is:

$$w^p_i = \sum_{k=1}^{N(i)} t^i_k \times (1 - p^k_i) + t^j_j \times (1 - p^j_j)$$

Obviously, the first factor in this formula calculates a peer’s preferable workload of past work and the second factor is the preferable workload of coming work. Then the present task instance is assigned to a peer with the least preferable workload.

5.3.4 Summary and discussion

To summarise, this section has presented three allocation policies. Round-robin based allocation simply assigns the work in the order. The allocation policy based on the workload assigns the work dynamically to balance the “absolute” load amongst the peers. Both local and global performance optimisations for this policy are discussed. Moreover, the allocation policy based on weighted workload even takes human preference for various types of work into consideration, and assigns work dynamically to balance the “relative” load for better user satisfaction.

To support real-world applications, system administrators can choose a most suited policy. Moreover, the dynamic allocation policies can be further extended to offer adaptive workload scheduling. Historical information such as performance of peers over some period of instantiating the workflow can be considered in allocating work.
5.4 Instance execution

After a process instance is created completely, the instance should be executed under the management of the workflow system. Various task instances should be scheduled to enact step-by-step, as defined in the process definition. This instance enactment provides operational functions to support the execution of a process instance [WfMC95], such as:

- interpretation of the process definition,
- management of instance execution, e.g., start/stop/suspend/resume, etc.,
- navigation between tasks and routing decisions, and
- supervisory and management functions

In traditional client-server based workflow systems, these functions are completely fulfilled by a centralised workflow engine. However, in SwinDeW, because there is an intentional absence of a centralised workflow engine, these functions need to be fulfilled through the direct communication and coordination among peers. In short, the responsible peers interpret the process definition data and manage the execution and the state transition of corresponding task instances independently. Upon termination of a task instance, the peer makes its own decision on the routing of the execution thread, according to the process definition and the conditions of the terminated task instance. The adjacent peers exchange control information directly to enable task instances to be executed in the proper order. At the same time, application-related data are also transferred forwards from one peer to another as required. Moreover, information about the instance status, performance information and historical data, is also exchanged, normally between working peers and administrator peers, to provide supervisory and administrative services.

5.4.1 Types of messages

To schedule the execution of various task instances in a proper order without the assistance of a centralised workflow engine, relevant peers should collaborate with one another in enacting process instances. This collaboration is realised through
direct message exchange amongst peers. The message flow amongst peers for the process enactment service can be categorised as two types, i.e., *data messages* and *control messages*.

A data message transfers the application data related to a process instance to coordinate data dependency between tasks. Application data, also known as application case data, are application specific and not accessible (indeed, are not required for accessing) by the workflow management system. Instead, application data are strictly managed by the applications supporting the process instance. A data message normally transfers application data to match the output-parameter of a task instance with the input-parameter of another task instance.

In contrast, a control message delivers workflow control data which is a notification from one peer to another. Workflow control data, also known as workflow system data, are the data managed by the workflow management system. Such data are internal to the workflow management system and are not normally accessible to applications. In most cases, control messages facilitate peers to perform functional operations. For example, a peer can decide whether the execution of a task instance can be started upon receipt of a control message from its predecessor peer(s).

All messages used in SwinDeW are structured in XML format. Basically, each message consists of two segments: a message header and a message body. A message header consists of a message type indicator which represents the type of the message (either data or control), and a standard message tag which encapsulates information such as the source and destination of this message. The header of a message helps peers to interpret this message appropriately. The details of this message used in peer interaction are contained in the body of the message. Hence, a data message is formatted as follows:

\[
\text{<Message header (Message type, Message tag)> <Message body>}
\]
The message tag attached to a message contains transactional information used during peer interaction. Normally, a message tag should contain at least the following fields:

<Instance ID>…</Instance ID>
<Source task>…</Source task>
<Source peer>…</Source peer>
<Destination task>…</Destination task>
<Destination peer>…</Destination peer>

The instance ID indicates the process instance to which this message is related. The source task and the source peer indicate the issuer of the message. The destination task and the destination peer indicate the recipient of the message. This tag is extensible so that other information such as date and time when this message is issued can be inserted into this tag easily.

Different types of messages have different message bodies. The message body of a data message is very simple. Normally, documents which contain application-related data are attached to data messages as body sections. For example, in an insurance claim processing workflow, a received claim form is attached to a data message and passed on from one peer to another for processing. In contrast, the message body of a control message is a notification regarding a task instance. SwinDeW supports the following types of notification:

- *Completion of a task instance*: A peer that has just completed a task instance notifies its successor peers that the work has been completed successfully.

- *Cancellation of a task instance*: A peer that is in charge of a task instance notifies its successor peers that the task instance has been cancelled.
• **Reallocation of a task instance**: A peer notifies other relevant peers that a task instance is reallocated to a different peer.

• **Detection of an “unavailable peer” exception**: A peer notifies relevant peers of a detected “unavailable peer” exception.

Upon receipt of a control message, a peer may respond accordingly to manage its own task instance. As detailed in Section 5.4.2.1, when a peer receives a completion notification, it evaluates the start condition of its task instance and determines whether the execution can be started. When a peer receives a cancellation notification, it evaluates the start condition of its task instance and determines whether the instance needs to be cancelled, as detailed in Section 5.4.2.3. In addition, when a peer receives a reallocation notification which normally occurs in exceptional situations as detailed in Section 5.4.3, it updates its successor or predecessor peers, and is ready to interact with new peers. Finally, the handling of exception detection messages will also be discussed in Section 5.4.3.

### 5.4.2 Functional operations and state transition

The execution of a process instance can be reflected by the state transitions of various task instances. In general, the execution state of a task instance can be one of *unacted*, *enacting* and *enacted*. Unactment of a task instance means this instance has not been started. If a task instance is enacting, it means this instance has been started while not completed yet. If a task instance is enacted, it means this instance has been completed. In particular, a number of common states which a task instance may take are defined, each of which represents an internal condition defining the status of a task instance at a particular point of time. SwinDeW maintains such status information as part of its workflow control data. The following statuses are supported, which conform to WfMC standards.

• **Initiated** – the task instance has been created, but may not yet be running

• **Running** – the task instance has started executing
- **Suspended** – the task instance is quiescent, no work is performed until it is resumed

- **Complete** – the task instance has achieved its completion conditions and the succeeding task instances are in progress

- **Terminated** – the execution of the process has been stopped, due to error or user request

- **Skipped** – the task instance has been skipped before its execution starts

Correspondingly, state transitions of task instances are triggered by task instance operations. The following task instance operations are available:

- **StartTask(i)**: Starting a task instance involves setting up the input parameters and the application program to execute it. The task instance is executed under the control of the peer which creates it.

- **SuspendTask(i)**: Suspending a task instance involves the log of all the internal states of the task instance.

- **ResumeTask(i)**: A task instance continues execution from where the instance is suspended.

- **CompleteTask(i)**: When the work represented by a task instance is completed, the task instance is completed.

- **TerminateTask(i)**: A running task instance is stopped abnormally due to external forces, such as errors and human intervention. No internal states will be recorded.
- **RestartTask(i):** Restarting a task instance to perform the work required.

- **SkipTask(i):** Skipping a task before its execution starts. Reserved resources are released as well.

![Task instance state transition diagram](image)

**Figure 5.3 Task instance state transition diagram**

Figure 5.3 depicts the state transition diagram of a task instance and the operations which trigger the transitions. In particular, a task instance after process instantiation is in state `initiated`. The `StartTask` and the `SkipTask` operations trigger the state transition to `running` and `skipped`, respectively. A running task instance enters `suspended`, `completed` and `terminated` as a result of the operations `SuspendTask`, `CompleteTask` and `TerminateTask`, respectively. When a suspended instance is resumed using `ResumeTask`, or a terminated task is restarted using `RestartTask`, the transition state goes back to `running`.

**5.4.2.1 Starting a task instance**

In general, the start of a task instance in workflow depends on the satisfaction of two conditions: the *data condition* and the *control condition*. The data condition of a task defines the start condition of this task from the application data dependency perspective. In most cases, a workflow task requires some input application data, which are normally the output of its preceding task(s), and generates some output application data, which are transferred to its succeeding task(s) to be its(their) input.
A task can be executed only after essential application data are available. For example, in an insurance claim processing workflow, a task to examine the validity of a claim can start only after the claim form is presented. On the contrary, the control condition of a task indicates the start condition of this task from the control data dependency perspective. As tasks in a workflow are normally interrelated, the start of a task may depend on the completion of a(some) relevant task(s). The start condition of a task instance is defined as transactional rules in the workflow definition.

For different types of execution structure, there are different execution conditions, as displayed in Table 5.1.

**Table 5.1 Types of control conditions among tasks for starting task instance**

<table>
<thead>
<tr>
<th>Execution Structure</th>
<th>Control conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>![ single ]</td>
<td>The control condition of task instance $i$ is satisfied only after its single preceding task instance is completed</td>
</tr>
<tr>
<td>![ and ]</td>
<td>The control condition of task instance $i$ is satisfied only after all its preceding task instances are completed</td>
</tr>
<tr>
<td>![ or ]</td>
<td>The control condition of task instance $i$ is satisfied only after one of its preceding task instances is completed</td>
</tr>
</tbody>
</table>

The combination of the data condition and the control condition is the start condition of a task instance. For example, in an Or-Join execution structure, the start condition could be satisfied when one of its preceding task instances is completed and the required data from the completed instance are received. A peer evaluates the start condition of a task instance and starts the task execution when the condition is satisfied, using the StartTask operation.
5.4.2.2 Completing a task instance

The completion of a task instance means the completion criteria for the task instance are met. From the coordination point of view, the completion of a task instance not only triggers the state transition to complete, but also normally results in the data and control messages flowing to the succeeding task instance(s), according to the rules defined in the process definition. In particular, the control message carries a task completion notification.

Again, given the different types of execution structure, there are different types of data and control flows, as displayed in Table 5.2.

**Table 5.2 Types of data and control flows for completing task instance**

<table>
<thead>
<tr>
<th>Execution Structure</th>
<th>Data and control flows</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image.png" alt="i" /></td>
<td>The completion of task instance $i$ results in the data and control flowing to its single succeeding task.</td>
</tr>
<tr>
<td><img src="image.png" alt="i and" /></td>
<td>The completion of task instance $i$ can results in the data and control flowing to all of its succeeding task instances.</td>
</tr>
<tr>
<td><img src="image.png" alt="i or" /></td>
<td>The completion of task instance $i$ can results in the data and control flowing to one of its succeeding task instances.</td>
</tr>
</tbody>
</table>

The peer in charge of a task instance completes the execution of this task instance through the *CompleteTask* operation. After that, this peer determines to which peer(s) the intermediate data and control flows should be routed. Consequently, the transfer of data and control information may change the start conditions of the succeeding task instance(s) and trigger the relevant peer(s) to start succeeding task instance(s).
5.4.2.3 Skipping a task instance

Task instances which are in state *initiated* can be skipped using the *SkipTask* operation which triggers a state transition to *skipped*. Hence, a task instance can be skipped only before its execution starts. Skipping task instances allows participants to save time and effort for tasks which are not needed in a particular process instance. A typical case of task skipping is in the situation of *Or-Split* routing. In this situation, the execution thread only takes one branch when encountered with multiple workflow branches. The task instances at those branches which have not been selected thus need to be skipped.

If the skipped task instance has its own succeeding tasks with a sequential or branching structure, these tasks need to be skipped as well because their start conditions will never be satisfied. Therefore, to deal with this situation, the peer in charge of a skipped task instance sends a control message to its successor peers, carrying a task cancellation notification. The succeeding task instances then may be skipped. In this situation, the skip operation is realised through the transfer of cancellation notifications.

5.4.2.4 Suspending and resuming a task instance

Suspending and resuming a task instance are control operations performed by the working peer. A running task instance can be suspended temporarily using the *SuspendTask* operation. The reasons that cause the suspension of a task instance may vary. For example, the execution of a task instance is blocked because the required resource is unavailable, or a task instance is waiting for synchronisation with the execution of another suspended task instance. Correspondingly, a suspended task instance can be resumed using the *ResumeTask* operation when the reason for this suspension is no longer tenable.

Under certain circumstances, a running task instance can also be suspended and resumed by an administrator peer. This should be always related to exception handling. For example, when the working peer in charge of a running task instance is required to undertake more critical work urgently at a certain time, the peer will
be instructed to suspend the current running instance temporarily. After the urgent work is completed, this working peer then resumes this task instance.

5.4.2.5 Terminating and restarting a task instance

When errors or exceptions occur, a running task instance can also be terminated using the \textit{TerminateTask} operation. This operation stops the execution of the task instance without keeping the execution states and conditions. In most cases, termination of a task instance will cause severe problems to the process instance. Dealing with exceptional and erroneous situations is very complicated. Normally, human intervention is required and methods like rollback and compensation may be used. If possible, a task instance in state \textit{terminated} can be restarted using the \textit{RestartTask} operation when pre-existent errors/exceptions are fixed. Support for task termination, general exception handling and task restart are beyond the scope of this thesis.

5.4.3 Automatic exception handling in the case of peer unavailability

A process is intended to achieve process goals completely and with maximum efficiency. Any departure from this achievement is considered an \textit{exception}. In general, exceptions can arise from changes in resources, organisational structure, company policy, task requirements or task priority, and hardware/software failures [HSB98, KD00]. If not detected and handled effectively, exceptions can result in severe impacts on the cost and schedule performance of workflow systems. Therefore, a critical challenge for workflow systems is their ability to respond effectively when exceptions occur [BW95, CCP99, HA00].

The SwinDeW system offers limited automatic exceptions handling ability so far. Most of the exceptions require human intervention through the monitoring and administration service. However, the handling of exceptions caused by unavailable peers during instance enactment is addressed in SwinDeW to enhance reliability. This is because of the following reasons:
(1) Peer (agent) unavailable is one of the major causes of exceptions in workflow systems. Therefore, this exception may occur more frequently.

(2) The exceptions caused by unavailable peers during instance enactment may cause more severe problems in SwinDeW than in conventional client-server based workflow systems. This is because each peer in SwinDeW plays a partial role of workflow engine functions. Thus, an unavailable peer may disable the system to schedule the execution of a process instance. Moreover, the ordinary peers in SwinDeW are likely to be more vulnerable to failures than servers in client-server based workflow systems. Although SwinDeW removes the single-source failure (server), there are multiple potential places for problems to arise if the “unavailable peer” exception cannot be handled properly.

Due to the above two reasons, it is important to design an automatic facility to handle such exceptions. The first step of this facility is for peers in SwinDeW to detect these exceptions. The “unavailable peer” exception can be detected using the peer registration service. As described in Section 3.4.2, each peer notifies its neighbour peers about its intention to leave. Moreover, each peer also sends a status message to its neighbour peers periodically, indicating that this peer is alive. Thus, the unavailable peer can be detected easily by its neighbour peers, including its predecessor peers and successor peers in a process instance. Normally, there are the following three types of “peer unavailable” exceptions:

(1) A peer leaves the system intentionally because of some special reasons (e.g., the participant being sick) and thus becomes unavailable to perform the task instances initially assigned to it. This exception is detected proactively, in fact, normally reported by the leaving peer.

(2) A peer becomes unavailable accidentally due to software/hardware failures before it starts executing the task instance. In this case, the start condition of the task instance which is assigned to the “dead” peer has not been satisfied. In other words, at least one predecessor peer of the “dead” peer
has not completed a preceding task instance. Such a predecessor peer can
detect this exception because the execution thread cannot go forward to the
“dead” peer later.

(3) A peer becomes unavailable accidentally during the execution of a task
instance, hence, it, as the “dead” peer, cannot pass the work onto the
successor peer(s). This exception can be detected by the successor peer(s)
of the “dead” peer, which is waiting for notifications from the “dead” peer.

Once an exception has been detected and reported through detection
notifications, it needs to be resolved so that the workflow can be returned to a viable
state. This can be achieved in SwinDeW by automatic peer collaboration.
Particularly, three general strategies are available in dealing with “peer unavailable”
exceptions, with the following preconditions and actions [KD00]:

(1) Wait till peer available: if the unavailable peer will become available in
time to complete the task on the current schedule then wait for this peer to
start (or resume) the task.

(2) Change the task so that an available peer can perform it: if the task can be
performed in a different way by available peers with other capabilities then
modify the task and re-assign it accordingly.

(3) Find a new peer with the same capability: if another peer with the same
capability is available, then assign the task to that peer.

Obviously, it is very difficult to evaluate the preconditions of the first two
strategies in a decentralised environment. Therefore, SwinDeW only supports the
third strategy to re-assign the task instance to an available peer with the same
capability, i.e., an available peer in the same community with the unavailable peer.
To resolve an “unavailable peer” exception, the relevant peers should collaborate. Figure 5.4 describes an “unavailable peer” exception; it illustrates the case of a sequential routing structure for the purpose of simplicity. Initially, task instances $T_p$, $T_i$ and $T_s$ are assigned to peers $P_p$, $P_i$ and $P_s$, respectively. $P_j$ and $P_k$ are peers in the same community as $P_i$. During instance execution, an exception may arise when $P_i$ becomes unavailable because the peer network is disconnected due to the exclusion of $P_i$.

In the case that $P_i$ leaves the system intentionally, exception type (1) may occur, as discussed earlier in this section. This exception is reported and handled by $P_i$ itself. That is, when $P_i$ sends out a departure notification, it also forwards the unaccomplished task instance ($T_i$) to peer members in the same community, i.e., $P_j$ and $P_k$. $P_j$ and $P_k$ then negotiate automatically to determine the reallocation of $T_i$. After that, the selected peer, e.g., $P_k$, notifies the predecessor and successor peers ($P_p$ and $P_s$) about the reallocation through reallocation notifications.

In the case that $P_i$ becomes unavailable accidentally, the exception is detected by either $P_s$ or both $P_s$ and $P_p$. If $P_i$ has not started executing $T_i$ when it becomes unavailable, this exception will be detected by both $P_p$ and $P_s$ because both $P_p$ and $P_s$ are aware that $P_i$ cannot carry out $T_i$ successfully. Otherwise, if $P_i$ is executing $T_i$ when it becomes unavailable, this exception will be detected by $P_s$ only because $P_p$ might not know whether $P_i$ has carried out $T_i$ successfully. The detection of the exception is then reported to other peers with the same capability, i.e., $P_j$ and $P_k$ in this figure, through the exception detection message(s), as discussed in Section 5.4.1.
Obviously, for the rest of the peers in the same community as the unavailable peer, they know which type of exception has occurred upon receipt of exception detection message(s). If the message(s) from the predecessor peer(s) arrives, it is clear that an exception of type (2) occurs. If the message(s) from the successor peer(s) arrives first, the reported exception could be of either type (2) or type (3). In order to determine the type of the reported exception, the coordinator peer of the community will check the routing structure of $T_i$ and interact with the predecessor peer(s) of the “dead” peer to see whether the start condition of $T_i$ has been satisfied. If the start condition of $T_i$ has been satisfied, a type (3) exception has occurred. Otherwise the reported exception is a type (2) exception.

To deal with exceptions of type (2) and type (3), there are different peer interactions and collaboration after the detection of the exceptions. Using the exception in Figure 5.4 as an example, the following two automatic exception handling mechanisms explain what happen in those two different cases:

- When a preceding peer ($P_p$) reports an “unavailable peer” exception (exception type (2)), it also starts a new task instantiation to re-allocate the task instance which is initially assigned to the “dead” peer ($P_i$) to another peer. The instantiation request is sent to the rest of the peers in the same community as the “dead” peer, i.e., $P_j$ and $P_k$. Then $P_j$ and $P_k$ negotiate automatically to determine the reallocation of this task instance, using the mechanisms presented in Sections 5.2 and 5.3. After that, the selected peer, e.g., $P_k$, notifies the successor peer ($P_s$) of the reallocation of the task instance through delivering a reallocation message. Eventually, the peer network of this process instance is converted from $P_p \rightarrow P_i \rightarrow P_s$ into $P_p \rightarrow P_k \rightarrow P_s$. The “unavailable peer” exception caused by $P_i$ is resolved properly.

- When an “unavailable peer” exception is only detected by a successor peer ($P_s$) (exception type (3)), this “dead” peer has already started the execution of the task instance. SwinDeW supports rollback for redo-able tasks. When a task is redo-able, it means a task can be re-executed to achieve the same
results. For example, the approval of an insurance claim is redo-able. In this example, $P_i$ and $P_k$ negotiate automatically to determine the reallocation of this task instance. Subsequently, the selected peer, e.g., $P_k$, notifies the predecessor peer ($P_p$) of the reallocation of the task instance through transferring a reallocation message, asking $P_p$ to offer the required intermediate results again. $P_k$ can restart task execution after it receives the required application data. At the same time, $P_k$ notifies the successor peer ($P_s$) of the reallocation of the task instance through delivering a reallocation message. For tasks that are not redo-able tasks, human intervention may be required.

In the case that the exceptions occur in other control structures such as branching and convergence, similar actions can be taken by the predecessor peers, the successor peers and the available peers in the same community as the “dead” peer.

5.4.4 Summary of instance execution

Put simply, instance execution is coordinated by direct communication among the peers performing relevant task instances. Such execution is light-weight and at a relatively low cost. A peer performing a task receives messages directly from its predecessor peer(s) before it starts working. Then the peer makes a decision on whether the task instance can be executed and starts execution when it can. After the task is completed, this peer also notifies its successor peer(s) directly and properly, according to the process definition, and passes the corresponding information as predefined. The successor peers repeat the same procedure until the completion of the whole process. Moreover, in the case that “unavailable peer” exceptions occur, automatic exception handling is designed to reallocate task instances, which also enhances system robustness.

5.5 Monitoring and administration service

WFMSs require an administrative component to allow system engineers and/or system administrators to control system components and query system’s status,
performance data, and processing information. In SwinDeW, this functionality is provided as an integrated system service: the monitoring and administration service as depicted in Figure 3.2. Like the process enactment service, the monitoring and administration service is also provided in an autonomous, collaborative and flexible fashion. To achieve this, various administration messages are exchanged between administrator peers and ordinary working peers.

As mentioned before, there exists in SwinDeW a set of administrator peers which are associated with system administrators and/or system engineers. These peers are authorised to monitor operation of the workflow system, collect performance data, provide administration facilities, etc. The most important administration message is the request and report message which asks for and reports on task instance state, performance data, historical information, and so on.

During run-time, administrator peers are able to send requests to ordinary working peers at any time, querying the statuses of components. The peers servicing these requests then report the states of task instances, and performance and historical information to the requesting peers directly. Status information from a working peer might include information about the current task instance being processed, the progress or stage of processing, and the elapsed processing time for the current task instance. By these means, administrator peers can obtain a global view of the workflow management system and running process instances. The global view of a running process instance can be offered in a visualised fashion as a “mental map” which is very useful to create a better teamwork atmosphere [YLSH+04].

During system operation, ordinary peers also interact with the administrator peers to report on their performance. The administrator peers store this information in a data repository for later retrieval and analysis. Performance information might include process latency and throughput, a cumulative request count, and an error count and description.

System administrators and system engineers may want to access information about completed task instances and/or instance origins and times. When reporting
statistical performance data, each peer also reports request information such as instance origins, instance times, instance parameters, and instance completion times. This information is stored in a data repository for later retrieval as well.

System administrators and system engineers are also allowed to purge historical information from the data repository. Based on the information gathered, system administrators and system engineers can analyse the performance of existing system settings and the efficiency of existing process models. Diagnostic data can be collected to reconfigure the workflow system in a long-run.

In addition, SwinDeW also provides the following types of administration messages:

- *Work overload alert and reallocation of task instance*: A working peer may notify the administrator peers that the workload might cause it to miss the assigned deadline. Subsequently, an administrator peer is able to reallocate work from this peer to another one that is less busy.

- *Deadline alerts*: Administrator peers notify working peers when deadlines are approaching or have expired or some other conditions are met.

- *Exception report and handling*: Ordinary working peers are able to report exceptions and erroneous situations to administrator peers. Handling of exceptions normally requires human intervention. For example, if an exception occurs when a required tool is unavailable, administrator peers can handle it by either finding a new tool with same functionality or changing tasks to apply available tools. The details of general exception handling are beyond the scope of this thesis.
5.6 Summary

In this chapter, the mechanisms supporting the run-time functions of SwinDeW, i.e., the process enactment service and the monitoring and administration service, are discussed. The provision of these two services is based on SwinDeW facilities for data storage. Process instantiation, instance execution, and monitoring and administration are accomplished through direct communication and coordination amongst peers which have essential data to support them. From the user support point of view, this service allows the vast number of the ordinary workflow participants to be involved in administrative workflow functions such as process instantiation from which they are traditionally excluded. With the assistance of peers, workflow participants are capable of negotiating automatically and making decisions such as determining who accepts a task instance. This in turn enables workflow participants to have more control over their work and inspires the participant to work more actively. From the system performance point of view, the dynamic work assignment is expected to improve system efficiency and the decentralised coordination helps to achieve light-weight coordination at a relatively low cost. Moreover, the automatic handling of “unavailable peer” exceptions is likely to enhance the system reliability.
Chapter 6

Extending SwinDeW for incomplete process support

The SwinDeW system mechanisms discussed in Chapters 4 and 5 are designed exclusively for supporting completely-specified (complete for short) workflow processes. In this chapter, the SwinDeW decentralised workflow management system is extended to support incompletely-specified (incomplete for short) processes as well.

First, Section 6.1 identifies the major causes of incomplete processes and discusses the difficulties that incomplete processes bring to workflow management systems. Based on the discussion, the new requirements for SwinDeW for supporting incomplete processes are analysed in Section 6.2. After that, Section 6.3 discusses the scope of incomplete process support addressed in this thesis. Finally, to support incomplete processes, a hierarchical, multi-level process modelling and execution approach is presented in Section 6.4.

6.1 Causes of incomplete processes

In traditional workflow application domains such as office work and the banking industry, the processes supported by a workflow management system are fairly mature and fixed. The goals of these processes, the steps towards each goal and the details of each step are normally pre-determined. Hence, process modellers and
engineers can completely foresee and formally specify all the situations of a process before-hand. The support of workflow processes in such application domains is naturally based on the philosophy that a process is defined completely before the execution of process instances. As a result, traditional workflow management systems abide by the principle of defining first and executing thereafter. The execution mechanism adopted traditionally is known as lazy execution where the workflow system starts the execution of the instances of a workflow process only after the process is modelled and specified completely. The build-time functions which deal with process modelling, representation and storage issues and the runtime functions which control the execution of process instances are performed separately.

However, in some newer, non-traditional application domains such as software process support, scientific computing and research project management, the process specification obtained after build-time is often a sketch which describes a workflow process roughly and incompletely. Major tasks, artefacts, roles and the structure of the process model can normally be elicited. However, further elaboration and articulation of the process are required to be completed during process enactment based on instance data [JST97, Sie99]. Such processes are known as incomplete processes. Normally, incomplete processes emerge because the processes in some new application domains have exhibited some distinguishing features, as observed as follows:

1. Processes in some new application domains are much more complex than those in traditional ones. The increased complexity makes a process very difficult to portray completely before its execution. At build-time, only the major structure and the first few steps can be determined, while some other steps remain uncertain and need to be completed during process execution. For example, in the software process area where the entire exercise of software development is modelled as a complex process, the steps to implement complex software may not be finalised until the requirements analysis and system design have been done [VKK01]. Normally, this part of the process is modelled roughly as some big blocks of steps (i.e., sub-
processes) during the process modelling period. These blocks need to be further articulated to executable steps at run-time when process execution reaches a certain stage, taking the outcome of the completed steps into account.

(2) Processes in some new application domains are also more flexible than those in traditional ones. Various cases may not follow procedural rules exactly but have minor deviations. In particular, the ways to fulfil a particular task in various cases may vary dramatically. Foreseeing and modelling of all these deviations is either impossible or at least cumbersome. Thus, it is inapplicable to define a single, complete process model which describes all the situations that the instances derived from it may encounter. A typical example of this case is healthcare, where in-patient treatments are prescribed uniquely for each case [SSO01].

(3) In some cases, there is a lack of essential information to model processes completely, especially in application domains like scientific research, invention and the laboratory environment. Modelling of such exploratory work is a tough mission because very limited information can be used for reference [WWVB96, SV96]. Especially, some modelling information can be obtained only after the process instance is executed to a certain stage. For example, in most research proposals, a research project is normally modelled as a process where the research goal is achieved through the fulfilment of a set of partially-ordered research tasks. The outcomes and achievements of the research tasks at an early stage will always influence the way to carry out the research tasks at a later stage. Therefore, the complete specification of tasks at a later stage cannot be made because the outcomes of tasks at an early stage remain unsure. Consequently, a complete blueprint of the whole research procedure cannot be obtained at the build-time.

These three features are somewhat interrelated in reality. For example, a workflow process can be so complex (feature (1)) that there lacks sufficient
information to model it (feature (3)). The main commonality of these three features is that they make traditional workflow systems unable to support processes in such new domains. Obviously, the principle that a workflow should be defined completely before process execution is violated. Workflow processes sometimes cannot or should not be defined completely at build-time. Therefore, a desirable feature of workflow systems is to provide appropriate support for incomplete processes, which intertwines build-time and run-time functions and postpones some build-time functions to run-time. Unfortunately, adequate facilities in today’s workflow systems supporting incomplete processes are rather weak. As workflow technology is being increasingly used in non-traditional application domains, this limitation is seen as a major obstacle for the wide deployment of workflow systems in the real world.

6.2 Requirements analysis

Based on the discussion in Section 6.1, incomplete process support is the ability of workflow systems to execute a workflow process on the basis of a loosely or partially specified model, where the full specification of the process is made at run-time incrementally, and may be unique to each instance. Therefore some new requirements have been developed for workflow management systems as follows, which are not fulfilled satisfactorily in today’s workflow research [YYR04].

1. Incomplete parts of workflow processes should be allowed to be specified explicitly at build-time using a process modelling tool.

2. A workflow management system needs to adopt a different execution mechanism known as *eager execution*. Eager execution is where execution of the process instances can be started even if the process is not specified completely.

3. An automated run-time facility should be introduced, which enables further elaboration of the workflow processes on-the-fly without suspending the
whole system. Support for both process- and instance-level elaboration should be provided.

In a decentralised workflow management environment like SwinDeW, there are some extra requirements:

(4) An incomplete process definition can be divided into task partitions and task partitions can be distributed to and stored by relevant peers properly. An incomplete part of a process should be allowed to instantiate before execution. The workflow represented by an incomplete part of a process should be allowed to allocate.

(5) Run-time incomplete process support can be carried out in a decentralised environment so that on-the-fly process elaboration can be performed at the right time and the right place by the right participant.

Obviously, the original design of SwinDeW does not offer such an ability to support incomplete processes. To satisfy the above requirements, extra system mechanisms should be designed, which should reuse the existing system mechanisms largely and can be incorporated into the overall SwinDeW system framework seamlessly. This has been achieved in extending SwinDeW for incomplete process support in this thesis. The mechanisms are discussed in the rest of this chapter.

6.3 Scope of incomplete process support

The graph-based process representation discussed in Section 4.2 assumes that a process definition after build-time is complete and only contains atomic tasks. However, the definition of an incomplete process obtained after build-time modelling, known as a process draft, has two characteristics. First, a process draft is a “near-complete” definition. The major tasks and the structure of the process are documented clearly while the details of some tasks are missing. Second, the first few
steps of a process draft are complete so that the execution of process instances can start.

Since a process consists of partially-ordered tasks, support for incomplete processes can be treated as support for incomplete tasks within processes. The incomplete tasks which are addressed in this thesis could be classified as two types:

1. Incomplete composite tasks missing internal specifications. Of the six elements of a task, $T$ (Process-id; Task-id; $C_{pre}$; $C_{post}$; Capability; Specifics), Specifics is specified partially while Process-id, Task-id, $C_{pre}$ and $C_{post}$ are specified completely. The complete Specifics are expected to contain a set of partially-ordered sub-tasks which form a sub-process. As a result, Capability is always uncertain either.

2. Incomplete atomic tasks missing important attributes. Of the six elements of a task, $T$ (Process-id; Task-id; $C_{pre}$; $C_{post}$; Capability; Specifics), Specifics is specified completely while at least one element of $C_{pre}$, $C_{post}$ and Capability is specified partially.

An incomplete composite task is a “black-box” in the core process. Each composite task represents a unit of work which is fulfilled by executing a set of sub-tasks. These sub-tasks, each of which can be either atomic or composite, are also partially ordered, forming a sub-process. A composite task has a determined contribution to the overall process. In other words, the objective of a composite task, its position within the process, and its input- and output-parameters are all known. However, the details about how to fulfil a composite task, i.e., how to convert input parameters into output parameters remain uncertain for the time being. The full specification of the task needs to be made at run-time. The elaboration of a composite task, i.e., the conversion from the composite task to a complete, equivalent sub-process, will not affect those tasks which supply it with inputs or consume its outputs.
Given the workflow continuum introduced in Section 2.5.1, incomplete composite tasks are related to the first two subcategories of semi-structured workflow. They are open team (composite) tasks within standard workflow and controlled team (composite) tasks within standard workflow. The details of these two types of incomplete composite tasks are outlined as follows:

(1) An open composite task within standard workflow combines pre-determined and open tasks within a single workflow [NH94]. In this case, a pre-determined workflow is used as the major process structure while within a particular step of the process several participants will be engaged in the completion of a single composite task. This composite task is unknown at build-time and needs to be decomposed at run-time.

(2) A controlled composite task within standard workflow is characterised by integrating such types of tasks into predefined workflows that are somewhat more pre-determined than completely open process elements [NH94]. In particular, a controlled composite task is associated with a set of fragments, where a workflow fragment may consist of either an atomic task or a composite task. Normally, the fulfilment of a controlled composite task requires the execution of some of the fragments in a certain sequence. However, this sequence remains uncertain after build-time and needs to be articulated at run-time.

On the other hand, an incomplete atomic task is an atomic task with some uncertainties. Typical examples are that the human resource requirement and/or the pre- and post-condition of an atomic task are uncertain for the time being. These attributes need to be articulated before the execution of task instances starts.

There could be some other types of incomplete tasks which are more complex and are difficult to address. For example, a task may have both incomplete specifications and uncertain relationships with other tasks. However, a process which contains such incomplete tasks normally cannot be regarded as “near complete”. It is practically rare and could be very risky to start executing such
incomplete processes with expectation to articulate them appropriately at run-time. Therefore, adequate support for the above two types of incomplete processes is believed sufficient for real-world applications.

6.4 Multi-level process modelling and execution

As discussed in Section 6.1, due to the new features of processes, it is very difficult to completely foresee and define all the situations of a workflow process through a single attempt. Therefore, process modelling work is no longer a one-off occurrence but may experience several rounds. Regarding modelling of incomplete processes, some significant work has been done over the past a few years [AD00, KBTH+00]. Different modelling approaches such as run-time dynamism, configurable/partial execution, typed modelling, layered modelling, reflexivity, integrated support for participant communication have been proposed. Of these approaches, the classical approach is best represented in “Hierarchical Task Analysis (HTA)” [KA92], where a process is described as a hierarchical structure of tasks and sub-tasks. Since it is not the intention of this thesis to develop a new modelling approach, this section simply discusses a multi-level process modelling and execution approach which is based on HTA, within the context of SwinDeW. This approach defines a process incrementally in a stepwise manner and supports on-the-fly process articulation at run-time. Investigation into incomplete process modelling itself is beyond the scope of this thesis and may be conducted in the future.

First, Section 6.4.1 discusses the hierarchical process modelling approach in general which supports two types of incomplete processes as discussed in Section 6.3. Then the existing difficulties in SwinDeW for supporting composite tasks are analysed in Section 6.4.2. To address these difficulties, a special decomposition task is introduced. This decomposition task’s definition and fulfilment are presented in Sections 6.4.3 and 6.4.4, respectively. After that, the mechanisms to execute instances of incomplete processes according to run-time decomposition are discussed in Section 6.4.5. Moreover, Section 6.4.6 further discusses handling of inconsistency caused by process-level decomposition. Finally, based on what has
been discussed in these previous sections, elaboration of incomplete atomic tasks is covered in Section 6.4.7.

6.4.1 Hierarchical process modelling

In general, in hierarchical task analysis, the designer breaks down a task from top to bottom, thereby showing a hierarchical relationship amongst the tasks. To allow uncertain parts of a process to be unveiled gradually, a process modelling approach is supported by SwinDeW, which focuses on hierarchical analysis and decomposition of process models.

![Hierarchical process modelling diagram]

**Figure 6.1 Hierarchical process modelling**

As shown in Figure 6.1, this modelling approach adopts a top-down mechanism and defines a workflow process at different levels of abstraction. The process definition at a higher level of abstraction contains incomplete tasks (either composite or atomic). The remodelling work specifies the uncertainties of this process definition and generates a new (version of the) process definition, which is at the next level. This procedure is known as a *modelling round*. Again, the new definition obtained can contain incomplete tasks. Further modelling round(s) then
need to be conducted to generate process definition(s) at the lower level(s) of abstractions. In this way, process modelling is carried out in a stepwise manner and some rounds might even be carried out during run-time.

The modelling of the sample process in Figure 6.1 experiences three rounds, two at build-time and one at run-time. The first round of modelling creates a process definition which consists of three sequential composite tasks: T_1, T_2 and T_3. In the second round, T_1, T_2 and T_3 are decomposed into three sub-processes respectively and a new process definition is created. T_1 is decomposed into a sub-process which consists of T_{1,1}, a complete atomic task, and T_{1,2}, a composite task. T_2 is decomposed into a sub-process which consists of four complete atomic tasks, T_{2,1}, T_{2,2}, T_{2,3} and T_{2,4}. T_3 is decomposed into a sub-process which consists of T_{3,1}, an incomplete atomic task, and T_{3,2}, a complete atomic task. At run-time, T_{1,2} is further decomposed into a sub-process which consists of T_{1,2,1} and T_{1,2,2}, and uncertainties of T_{3,1} is articulated. The final process definition which only contains complete atomic tasks is obtained.

To support the representation of composite tasks, some new process modelling operations are designed to extend to modelling operations discussed in Section 4.2, as summarised as follows:

- \textit{CreateCompositeTask}(i): Create a composite task, i, including the definition of input- and output-parameters.

- \textit{AddComponentTask}(i,j): To a given composite task, j, add an (atomic or composite) task, i, as a component task which is a sub-task within the sub-process represented by composite task j. It is assumed that both \(i\) and \(j\) already exist.

- \textit{DeleteComponentTask}(i,j): Delete a component task, i, from the composite task, j. This involves the deletion of edges adjacent to \(i\) in \(j\).
• \textit{DeleteCompositeTask}(i): Delete a composite task, $i$, from the process. This involves the deletion of component tasks within $i$ and the edges adjacent to $i$.

At the same time, the operations related to edges, e.g., \textit{AddControlEdge}(i,j,m), \textit{AddDataEdge}(i,m), etc., as described in Section 4.2, support both atomic and composite tasks. An edge can connect two atomic tasks, two composite tasks, or an atomic task and a composite task. Thus, both the external transition (dependencies between a composite task and other tasks) and the internal transition (dependencies between sub-tasks within a composite task) can be constructed using those operations.

Obviously, this hierarchical process modelling approach rationalises the modelling work and reflects human’s behaviour more naturally. Besides this, this approach also provides the potential of multiple-level process reusability because multiple rounds of modelling generate multiple reusable model fragment drafts. A process model at a higher level of abstraction can be reused to generate various process templates at a lower level of abstraction. This feature makes the workflow system capable of dealing with flexible workflow processes. Process instances with variations can be derived from the same process model at a certain level of abstraction and inherit some common features of the model.

\subsection*{6.4.2 Existing difficulties for supporting incomplete composite tasks}

Based on the hierarchical process modelling approach, decomposition of composite tasks on-the-fly at run-time is one of the key functions provided by SwinDeW. The main focus of this research is to address the run-time decomposition from both the process modelling and the system coordination support viewpoints. As such, the approach presented in this section is within the SwinDeW context but is not dependent on the workflow modelling language.

Apparently, the build-time and run-time mechanisms discussed in Chapters 4 and 5 cannot provide the required coordination support appropriately because of the following reasons:
• First, storage of an incomplete composite task is an issue that needs to be addressed. In SwinDeW, a process definition is divided into task partitions and then individual partitions are distributed to relevant peers according to capability match. However, for an incomplete composite task, it is not able to indicate clearly the capability required to fulfil this composite task. Therefore, the system cannot determine the places at where the definition of an incomplete composite task (input, output, predefined component tasks, etc.) should be stored.

• Second, instantiation of an incomplete composite task is a challenge. In SwinDeW, the process instantiation procedure involves multiple distributed peers which accept various task instances actively on behalf of corresponding workflow participants, and the work of a task instance is assigned to the responsible peer that creates it. Again, because an incomplete composite task does not have a complete specification of the sub-process, the system cannot determine the appropriate peers to create and execute the instance of the sub-process.

• Finally, regarding on-the-fly operations, since the run-time function is carried out in a decentralised way and since no peer except the definition peer and the administrator peers in the system has global knowledge about the process, the system cannot answer the following three questions:
   
   (1) Who will decompose a composite task?
   (2) When will a composite task be decomposed?
   (3) Where will the decomposition of a composite task occur?

6.4.3 Introducing the decomposition task

To deal with the difficulties discussed in the previous section, a special managerial task, known as a decomposition task, is designed in SwinDeW [YYR03c]. Figure 6.2 shows a decomposition task and its position in the process. The decomposition
task is associated with an incomplete composite task, $T_n$, and finally decomposes $T_n$ into a sub-process which consists of sub-tasks $t_{n,1}$, $t_{n,2}$, $t_{n,3}$ and $t_{n,4}$. The textual description of a decomposition task is as follows:

- **Description**: A decomposition task decomposes the associated composite task at run-time into a sub-process consisting of a set of partially-ordered sub-tasks, each of which can be either atomic or composite. The decomposition is based on instance data, application rules and application constraints.

![Diagram of a decomposition task and the associated composite task]

- **Responsibility**: A decomposition task is carried out by an authorised person with special skills to model processes, such as a process engineer or a project manager.

- **Inputs**: The inputs of a decomposition task are very flexible. Normally, a decomposition task may take the outputs of the preceding tasks of the associated composite task, as its inputs. In addition, a decomposition task
may have other inputs such as available resources, historical experience, incomplete specifics of the composite task, and so on. Input information is always dependent of particular application requirements and is beyond the scope of this thesis. In Figure 6.2, the input information is omitted.

- **Output**: The single output of a decomposition task is a complete specification of the associated composite task, which is a sub-process template describing how the input parameters of the composite task are converted into output parameters. This specification is taken by the associated composite task as input.

- **Incoming control**: A decomposition task receives notifications from the preceding tasks of the associated composite task upon their completion, i.e., the preceding tasks of the associated composite task are also preceding tasks of this decomposition task.

- **Outgoing control**: A decomposition task notifies the associated composite task upon its completion, i.e., this decomposition task is a preceding task of the associated composite task.

- **Constraints**: Normally, a decomposition task can begin at any time during process enactment. The latest start time is when the peer performing the decomposition task has received all incoming notifications from its predecessor peers.

Based on the above description, a decomposition task and the associated composite task can be specified uniformly using the task notation given in Section 4.3 as follows:

\[
T_{\text{decomposition}} \left( \text{Process-id}; \text{Task-id}'; \left( (C_{\text{pre}}, C'_{\text{pre}}), \text{and}) \right); (T_{\text{composite}}, \text{uncertain, sub-process specification}, \text{straight}); \text{process-engineering}, \text{Specifics'}) \right]
\]
$T_{\text{composite}}$(Process-id; Task-id; $(C_{\text{pre}}, (T_{\text{decomposition}}, \text{process-engineering}, \text{sub-process specification}), \text{and}); C_{\text{post}}; \text{uncertain}, \text{Specifics}^3$)

Clearly, from these two partitions, a decomposition task forms an And-Join structure with the preceding tasks of the associated composite task—$C_{\text{pre}}$—because the decomposition should be completed before the execution of the composite task. Similarly, to a decomposition task, its own preceding tasks—$C_{\text{pre}}$—form an And-Join structure with the preceding tasks of the associated composite task—$C_{\text{pre}}$. Whilst the capability of a composite task is uncertain, the execution of a decomposition task requires a special capability like process-engineering.

To support the modelling of decomposition tasks, two more process modelling operations are developed in SwinDeW.

- $\text{CreateDecompositionTask}(i)$: Create a decomposition task, $i$, including the definition of input and output parameters, and the application rules or constraints which help to decompose the composite task. The association between a decomposition task and a composite task is established through the use of the modelling operations related to edges as defined in Section 4.2.

- $\text{DeleteDecompositionTask}(i)$: Delete a decomposition task, $i$, from the process. This involves the deletion of the edges adjacent to $i$.

When constructing the dependencies among a composite task, its preceding task(s), and its associated decomposition task, the standard operations related to edges, e.g., $\text{AddControlEdge}(i,j,m)$, $\text{AddDataEdge}(i,m)$, etc., as defined in Section 4.2, cannot represent these complex relationships clearly. Therefore, the concept of the dummy task is used in this scenario in order that the representation of an incomplete process is consistent with the graph-based process representation discussed in Section 4.2. A dummy task represents a virtual task which synchronises workflow execution without carrying out any real work. Consequently, a dummy

---

3 The Specifics of a composite task is incomplete.
task does not require any resource and will not be distributed, instantiated and executed. At design stage, the operations related to a dummy task are:

- **CreateDummyTask(i)**: Create a dummy task, \(i\).

- **DeleteDummyTask(i)**: Delete a dummy task, \(i\). This involves the deletion of edges adjacent to \(i\).

![Figure 6.3 Visualised representation of a composite task](image)

**Figure 6.3 Visualised representation of a composite task**

Figure 6.3 describes the visualised representation of a composite task with a dummy task, using the process representation approach discussed in Section 4.2. The preceding task(s) of a composite task forms an *And-Split* routing structure with the decomposition task for this composite task and a dummy task. The decomposition task and the dummy task form an *And-Join* routing structure with this composite task. The succeeding task(s) of the composite task will not be affected. For the purpose of simplicity, data edges and the preceding tasks of the decomposition task are omitted.

With the support of the decomposition task, the first two difficulties discussed in Section 6.4.2, i.e., data storage and task instantiation, are resolved easily. The definition of a composite task is temporarily distributed together with the definition of the associated decomposition task, to peers associated with authorised participants who are capable of decomposing the composite task. As a result, the instantiation of a composite task can be done with the same instantiation mechanism for atomic tasks. A composite task is temporarily instantiated by a peer associated with an authorised person. This peer is in charge of the execution of the composite task.
task instance and performs some functions at run-time, e.g., assigning sub-tasks to other peers, to ensure the composite task instance (instance of a sub-process) is enacted properly. The details of these functions address the final difficulty, i.e., on-the-fly decomposition in a decentralised fashion and will be detailed in Section 6.4.4.

### 6.4.4 Performing the decomposition task

Normally, a decomposition task should be performed before the preceding tasks of the composite task are finished. This kind of decomposition is regarded as a *pull* mode where a composite task is decomposed in advance. Hence, once the work is passed to the composite task, the sub-process resulting from the decomposition can be enacted without real delay. However, in the worst case, the decomposition depends on the up-to-date data of instance enactment and the peer associated with a decomposition task can only start when the preceding work has been done. This is the reason why a decomposition task receives notifications from the preceding tasks of the associated composite task, as shown in Figure 6.2. This kind of decomposition is regarded as a *push* mode where the enactment of a decomposition task is triggered passively and may block the whole process for a certain period of time.

Decomposition of a composite task can be performed at both the instance- and process-levels. Instance-level decomposition represents a one-off change and only takes effect in the present instance. Allowing instance-level decomposition reflects the fact that a flexible process may have multiple, variant instances. Each of the instances fulfils a composite task in a different, sometimes even unique, way. On the contrary, process-level decomposition represents a permanent change to the workflow model and will be applied to all the instances created in the future. Permanent change is always related to process evolution and process reengineering. The consequence of such decomposition is to finalise that a composite task should be fulfilled in a determined way in the future.

To enact a decomposition task in an organised manner, the associated peer may monitor the progress of process enactment through communication with other relevant peers and take various inputs into account. Additionally, analysis and
modelling tools might be used to facilitate the peer to perform decomposition. In general, a decomposition task can be performed with one of the following:

- Partially automated or manual support: The decomposition task requires a workflow participant to define new component tasks or adapt some parameters and then build the sub-process from the new as well as existing component tasks.

- Fully automated support: The decomposition task automatically makes a complete specification of the composite task by composing the component tasks based on the instance data and given constraints.

Partially automated or manual support is mostly used for both open composite tasks and controlled composite tasks. The execution of decomposition tasks with this support always invokes a graph-based modelling tool automatically, helping to draw various nodes and edges for the sub-process. The authorised workflow participant who commits to performing the decomposition will consider the factors such as input- and output-parameters of the composite task, existing workflow fragments, application rules and constraints, instance data, etc. when specifying the sub-process. Also, the participant performing decomposition will indicate it is either instance-level or process-level decomposition. The details of modelling techniques are beyond the scope of this thesis.

On the contrary, fully automated support is normally suited to some kinds of controlled composite tasks where all the component tasks are predefined. The decomposition actually makes a valid composition of the given component tasks according to some application rules for a particular process instance. Typical examples of such controlled composite tasks are group decision scenarios such as votes or countersignature. In the process of task definition the number of potential participants participating in task completion, say \( m \), is defined. Each participant performs a workflow fragment (voting or signing). Later, during the actual task completion at run-time the individual participants out of the predefined superset \( (n \)
of \( m \) are determined automatically, performing the correspondent workflow fragments in a certain order. By default, fully automatic support in SwinDeW decomposes a composite task at the instance level.

However, modelling any non-trivial sub-processes is a difficult task and normally requires qualified persons [AJ00]. For this reason, SwinDeW provides fully automated decomposition support only for limited applications where the application constraints are rather simple so that automatic construction of sub-process can be made. In particular, suppose a controlled composite task has \( m \) workflow fragments which consist of \( m \) component tasks, SwinDeW supports automated sub-process generation for the following four types of application rules:

```
Figure 6.4 Automated decomposition of controlled composite tasks
```

(a)  
(b)  
(c)  
(d)
(1) *Do all of m, but one at a time.* This rule requires all the workflow fragments to be executed in sequence regardless of the sequence of their participation, as shown in Figure 6.4 (a). In this case, the system generates an execution order for these workflow fragments automatically. Sub-processes generated by this rule may have different execution orders in different process instances.

(2) *Do all of m in parallel.* This rule requires all the workflow fragments to be executed in parallel, as shown in Figure 6.4 (b). In this case, the system creates an *And-Split* structure and an *And-Join* structure for all the fragments automatically. In different process instances, sub-processes generated by this rule are the same.

(3) *Do n of m, but one at a time.* This rule requires any n workflow fragments to be executed in sequence regardless of the sequence of their participation, as shown in Figure 6.4 (c). In this case, the system selects n workflow fragments for the present process instance and generates an execution order for the selected fragments, automatically. In different process instance, sub-processes generated by this rule may select different fragments and/or generate different execution orders.

(4) *Do n of m in parallel.* This rule requires any n workflow fragments to be executed in parallel, as shown in 6.3 (d). In this case, the system selects n workflow fragments for the present process instance and creates an *And-Split* structure and an *And-Join* structure for the selected fragments, automatically. For sub-processes in different process instances, the selected fragments may be different.

The mechanisms for on-the-fly decomposition resolve the last difficulty discussed in Section 6.4.2. Both the instance- and process-level decomposition is carried out through the automated coordination amongst peers. The right person (authorised participant performing the decomposition task) performs the decomposition work at the right time (either push or pull mode) and at the right
place (the peer associated with the authorised participant), without suspending the whole system.

6.4.5 Instance execution

After the completion of a decomposition task, the associated peer notifies its single successor peer, i.e., the peer in charge of the composite task instance, and transmits a complete task specification to advise the execution detail of the composite task. Eventually, the instance of this composite task is enacted according to the new specification.

Once a composite task is decomposed into a set of sub-tasks, each of the sub-tasks might be executed by a capable peer. Thus, the fulfilment of a composite task, i.e., the execution of the sub-process, is achieved through the coordination amongst relevant peers, including the peer creating the instance of this composite task and the peers carrying out the sub-tasks. In particular, the instances of the sub-tasks should be created and the work represented by these instances should be assigned to capable peers. As a result, the peer network for the present instance should be converted properly. To describe the conversion of the peer network clearly, the process instance depicted in Figure 6.2 is used for illustration, as shown in Figure 6.5. The meanings of the symbols used in this diagram are as follows:

![Figure 6.5 Conversion of the peer network for the present instance](image)
\(P_{\text{pre}}\): a set of peers that execute the preceding tasks of composite task \(T_n\), i.e., \(Pre\) in Figure 6.2, in the present instance.

\(P_{\text{succ}}\): a set of peers that execute the succeeding tasks of composite task \(T_n\), i.e., \(Succ\) in Figure 6.2, in the present instance.

\(P_{\text{decom}}\): the peer that executes the decomposition task

\(P_n, P_{n,1}, P_{n,2}, P_{n,3}\) and \(P_{n,4}\): peers that execute \(T_n, t_n,1, t_n,2, t_n,3\) and \(t_n,4\), respectively.

Based on Figure 6.5, it is clear that the peers carrying out \(T_n\)’s preceding tasks, i.e., \(P_{\text{pre}}\), and the peers carrying out \(T_n\)’s succeeding tasks, i.e., \(P_{\text{succ}}\), are connected to \(P_n\) initially. After the decomposition is completed, the predecessor peers are connected to \(P_{n,1}\) and the successor peers are connected to \(P_{n,4}\). \(P_{n,1}\) will start performing work after \(P_{\text{pre}}\) complete its work, and \(P_{\text{succ}}\) will start performing work after \(P_{n,4}\) completes its work.

Of instance- and process-level decomposition, the former is relatively simpler to handle, as the specification of the sub-process is valid for one time only. In this case, peers \(P_n, P_{n,1}, P_{n,2}, P_{n,3}\) and \(P_{n,4}\) simply use the locally centralised architecture temporarily to satisfy the coordination requirements. \(P_n\) acts as the server while \(P_{n,1}, P_{n,2}, P_{n,3}\) and \(P_{n,4}\) act as the clients. All the instances of the sub-tasks are created by peer \(P_n\), and presented to \(P_{n,1}, P_{n,2}, P_{n,3}\) and \(P_{n,4}\) via a work list. Again, the selection of each of \(P_{n,1}, P_{n,2}, P_{n,3}\) and \(P_{n,4}\) is dynamic and negotiation-based. However, peers performing sub-tasks do not have direct interactions with one another. The execution scheduling of the sub-tasks is done on the server side, i.e., \(P_n\), because the logical structure of the sub-process only resides on \(P_n\). At the same time, peers performing sub-tasks do not have direct interaction with peers in both \(P_{\text{pre}}\) and \(P_{\text{succ}}\). Peers performing preceding and succeeding tasks of \(T_n\) still interact with \(P_n\) to coordinate instance execution. Instance-level decomposition does not affect other running process instances either. The composite task in other running process instances can be fulfilled in different ways. This approach eases the implementation and management of the sub-process because the locally centralised architecture reflects
the logically hierarchical relationship among the peers. At the same time, this local centralisation does not have side-effects because centralised coordination only occurs in a small scope temporarily.

Process-level decomposition is more complicated compared with instance-level decomposition. In order to reuse the sub-process definition for future process instances, workflow data should be refreshed at the process level. The definition of the sub-tasks should be distributed properly and the existing process repositories of relevant peers may need to be updated. In addition, to execute the present instance properly, the network of peers should be reconstructed properly to reflect the changes. Using the composite task in Figure 6.2 as an example, the execution of the decomposition task results in a process-level model conversion from $T_n$ to a sub-process which consists of $t_{n,1}$, $t_{n,2}$, $t_{n,3}$ and $t_{n,4}$. Again, using the peer network depicted in Figure 6.5 for illustration, the consequent operations of this process-level decomposition can be described as follows:

1. The specification of the sub-process, created by $P_{decom}$, is passed to $P_n$. It is also indicated that this sub-process is a result of process-level decomposition.

2. $P_n$ acts as the definition peer to partition and distribute the definition of the sub-process with the mechanism discussed in Section 4.4.

3. $P_n$ instructs the relevant peers to start creating an instance of the sub-process for the present process instance with the mechanism discussed in Sections 5.1, 5.2 and 5.3. The network of peers which consists of $P_{n,1}$, $P_{n,2}$, $P_{n,3}$ and $P_{n,4}$ is the result of this instantiation.

4. $P_n$ advises $P_{pre}$ to interact with $P_{n,i}$ directly instead of contacting $P_n$, using reallocation messages. $P_n$ also advises $P_{n,i}$ that $P_{pre}$ is the set of predecessor peers.
Similarly, \(P_n\) advises \(P_{\text{succ}}\) that \(P_{n,4}\), instead of \(P_n\), will interact with them directly, using reallocation messages. \(P_n\) also advises \(P_{n,4}\) that \(P_{\text{succ}}\) is the set of successor peers.

As depicted in Figure 6.5, the above steps convert the peer network of the present instance. The following steps will update the process repositories of relevant peers.

6.4.6 Inconsistency handling

Process-level decomposition of a composite task not only affects future process instances, but also may affect other running process instances. At the point that process-level decomposition occurs, other running process instances may have different statuses as follows:
(1) In some running process instances, the instances of this composite task have been completed at that moment, according to the outcomes of different instance-level decomposition.

(2) In some running process instances, the execution threads have not reached the composite task, i.e., the instances of this composite task have not been started at the time.

(3) In other running process instances, the instances of this composite task are being enacted. Relevant peers are executing this composite task according to individual instance-level decomposition at the time.

For running process instances having different running statuses, SwinDeW has different policies to handle the inconsistency between the process definition and the existing process instances. For running process instances in group (1), the outcome of the process-level decomposition normally does not affect the execution of the rest of the process instances. The remaining part of these running process instances will be executed as predefined. For running process instances in group (2), the execution of composite task instances can either continue with sticking to instance-level decomposition for a transition period, or transform to the new specification of the composite task immediately, depending on the particular requirements of applications and may require human intervention sometimes. For running process instances in group (3), there are two options for peers in charge of the instances of this composite task:

- Keep executing this composite task according to individual instance-level decomposition. This option is normally used in the situation that redoing this composite task is either impossible or at least inefficient.
- Terminate the current execution of this composite task and restart the execution according to the updated process definition. This may involve additional operations such as compensation and rollback.
Again, the selection normally depends on the application requirements and sometimes requires human intervention.

6.4.7 Elaboration of incomplete atomic tasks

After discussing the support for incomplete composite tasks, incomplete atomic tasks can be handled in a similar way. The assumption is that an incomplete atomic task can be treated as an incomplete composite task in SwinDeW. This is justifiable because an incomplete atomic task can always be grouped with other relevant tasks to form an incomplete, controlled composite task naturally. These tasks are component tasks of the formed composite task. Another reason to use incomplete composite task support for incomplete atomic tasks is that incomplete atomic tasks are normally interrelated and need to be addressed as a whole. For example, the articulation of the post-condition of an atomic task may affect the pre-condition of the succeeding task(s).

To build an appropriate composite task for an incomplete atomic task, a key issue is to identify the minimal set of tasks which will be affected by the articulation of this atomic task. Failure to identify this minimal set of tasks will result in severe problems. On one hand, if the set of component tasks contains irrelevant tasks, the execution of these tasks may be delayed unnecessarily at run-time, waiting for the outcomes of decomposition, especially in the push mode. On the other hand, if a relevant task is not identified appropriately, the run-time decomposition facility cannot further specify this task even if it is affected by the decomposition.

In general, SwinDeW supports three types of incomplete atomic tasks which have an uncertain capability attribute, uncertain pre-condition and uncertain post-condition, respectively, as discussed as follows:

(1) An atomic task with an uncertain capability attribute means the responsibility of this task is unclear at build-time. The issue of “who should do it” needs to be addressed later on.
(2) An atomic task with an uncertain pre-condition means the start condition of this task is unclear at build-time. The issue of “when it should be done” needs to be addressed later on.

(3) An atomic task with an uncertain post-condition means the transition after the completion of this task is unclear at build-time. The issue of “who should be informed next” needs to be addressed later on.

Regarding different types of incomplete atomic tasks, they form composite tasks differently. Figure 6.6 shows the formation of composite task for different types of incomplete atomic tasks. In this figure, the articulation of an incomplete atomic task, \( i \), eventually becomes the articulation of a composite task \( j \), which can be supported with the mechanisms discussed earlier in this section.

**Figure 6.6 Formation of composite tasks for incomplete atomic tasks**
An atomic task with an uncertain capability attribute forms a composite task by itself because the elaboration of the task responsibility will only affect the execution of this task. In this case, a decomposition task is created and associated with the formed composite task, which will specify the capability attribute of the incomplete atomic task. The algorithm to address this situation is designed as follows:

For each task $i$ of situation (1)

- $\text{Task Pre} = \text{Pre}(i)$; //preceding task of $i$
- $\text{CreateCompositeTask}(j)$; //create composite task
- $\text{AddComponentTask}(i,j)$; //add $i$ as a component task
- $\text{CreateDecompositionTask}(k)$; //create decomposition task
- $\text{CreateDummyTask}(l)$; //create dummy task
- $\text{CreateControl}(c_1,\text{and})$; //create control node
- $\text{AddControlEdge}(\text{Pre},c_1,k)$; //add control edge
- $\text{AddControlEdge}(\text{Pre},c_1,l)$; //add control edge
- $\text{CreateControl}(c_2,\text{and})$; //create control node
- $\text{AddControlEdge}(k,c_2,j)$; //add control edge
- $\text{AddControlEdge}(l,c_2,j)$; //add control edge
- $\text{AddDataEdge}(k,j)$; //add data edge
- ...... //other operations omitted

On the contrary, atomic tasks with an uncertain pre-condition and/or an uncertain post-condition are always interrelated. If there is an atomic task with an uncertain pre-condition, there must be one or more tasks with uncertain post-conditions. Also, if there is a task with an uncertain post-condition, there must be one or more tasks with uncertain pre-conditions. Hence, these tasks should be addressed as a whole. For this reason, the situations (2) and (3) are addressed as follows:

- An atomic task with an uncertain pre-condition forms a composite task with all its preceding tasks which may contain atomic task(s) with an uncertain post-condition. The algorithm for this situation is as follows:
An atomic task with an uncertain post-condition forms a composite task with all its succeeding tasks which may contain atomic task(s) with an uncertain pre-condition. The algorithm for this situation is:

For each task $i$ of situation (2)

\[
\begin{align*}
&\text{CreateCompositeTask}(j); \quad \text{//create composite task} \\
&\text{AddComponent task}(i,j); \quad \text{//add $i$ as a component task} \\
&\text{For each its preceding task $p$} \\
&\quad \text{AddComponentTask}(p,j); \quad \text{//add preceding task(s) of $i$ as component tasks} \\
&\text{Next} \\
&\text{CreateDecompositionTask}(k); \quad \text{//create decomposition task} \\
&\text{Task $Pre'$ = Pre(Pre)}; \quad \text{//preceding task(s) of the preceding task(s) of $i$} \\
&\text{......} \quad \text{//other operations omitted} \\
&\text{Next}
\end{align*}
\]

For each task $i$ of situation (3)

\[
\begin{align*}
&\text{CreateCompositeTask}(j); \quad \text{//create composite task} \\
&\text{AddComponent task}(i,j); \quad \text{//add $i$ as a component task} \\
&\text{For each its succeeding task $s$} \\
&\quad \text{AddComponentTask}(s,j); \quad \text{//add succeeding task(s) as component tasks} \\
&\text{Next} \\
&\text{CreateDecompositionTask}(k); \quad \text{//create decomposition task} \\
&\text{Task $Succ'$ = Succ(Succ)}; \quad \text{//succeeding task(s) of the succeeding task(s) of $i$} \\
&\text{......} \quad \text{//other operations omitted} \\
&\text{Next}
\end{align*}
\]

After composite tasks are formed for incomplete atomic tasks, the mechanisms for data storage, instantiation and run-time elaboration can be used to articulate the uncertain attributes of incomplete atomic tasks.
6.5 Summary

In this chapter, SwinDeW has been extended to support incomplete processes smoothly. The causes of incomplete processes have been identified and the conventional workflow system’s inability to support incomplete processes has been analysed. Finally, a multi-level process modelling and execution approach has been designed in SwinDeW for incomplete process support. The proposed mechanisms can address both incomplete composite tasks and incomplete atomic tasks.

In summary, by introducing a managerial decomposition task, on-the-fly decomposition of a composite task is modelled as an essential step in the process and integrated into the decentralised architecture of SwinDeW seamlessly. From an engineering point of view, this approach is strongly justifiable because the ordinary workflow participants may and should not be given an interface to specify a composite task using a complex workflow modelling language. From a system coordination viewpoint, a decomposition task is distributed, instantiated and scheduled to be executed as an ordinary task. Thus, process modelling at run-time can be performed with the support of the mechanisms for complete processes, at either instance- or process-level. Furthermore, incomplete atomic tasks are composed with other relevant tasks naturally to form composite tasks so that they can be addressed using the same mechanisms.
Chapter 7

Case studies

Based on the SwinDeW system design and the corresponding mechanisms discussed in Chapters 3, 4, 5 and 6, two real-world workflow applications are used in this chapter to illustrate how SwinDeW supports workflow processes in a decentralised way. The first case, discussed in Section 7.1, is a process of student registration service, which can be regarded as a traditional, complete workflow process. In Section 7.2, the second case discusses how SwinDeW supports the management of a research project in CICEC⁴ (Centre for Internet Computing and E-Commerce), which is normally viewed as a non-traditional, incomplete workflow process.

7.1 Case study 1: student registration service

7.1.1 Introduction

In a typical university, the student registration service deals with the registration issues of students, which include registration in courses, change of schedule, withdrawal of courses, payment of bills, and so on. This service is normally well-defined and can be viewed as a fixed flow of tasks that accomplish an objective. Thus, workflow solutions are well-suited to this scenario. A simplified usage scenario of this student registration service would be the following:

⁴ CICEC is a research group at Swinburne University of Technology.
A student submits a registration service request (RSR) form either online or in person.

The RSR form is pre-examined by an enrolment officer to ensure that the form is filled completely and correctly.

The form is then sent to the student’s course advisor for approval. The course advisor views the registration request and could either approve it or reject it. If the request is approved, the registration information will be sent to an enrolment officer for recording. On the contrary, if the request is rejected, the registration form will be sent to an enrolment officer to close this request.

The enrolment officer updates the student’s course information if the request is approved and issues a payment form to the financial section for billing. At the same time, the registration information is also sent to technical staff, requesting setup of the student’s computer account accordingly.

The treasurer and the technical staff handle the payment and computer account, respectively, and inform an enrolment officer afterwards.

The enrolment officer collects notifications from both the treasurer and the technical staff to complete the present registration service request. A registration confirmation is then sent to an enrolment officer who will correspond with the student.

Finally, an enrolment officer advises the student the outcome of the request and closes this request.

This scenario can be modelled as a workflow process, as shown in Figure 7.1. This process consists of a set of tasks which need to be executed in a certain order. Moreover, this process involves participants such as enrolment officers, course
advisors, technical staff and treasurers. A workflow participant may have multiple capabilities and play multiple roles. For example, an enrolment officer can have the capability of a course advisor. After an initial configuration, the peers associated with the workflow participants join relevant virtual communities according to the capabilities they represent.

Enrolment officer:
- Pre-examination, Recording,
  Collection, File-closing

Course advisor:
- Approval

Treasurer:
- Payment-handling

Technical staff:
- Account-management

Figure 7.1 A typical student registration process

---

5 These titles are also used for capabilities.
Two characteristics of the student registration service which need to be addressed properly are illustrated in this thesis:

- First, the student registration service is fairly distributed. To provide the requested services, various staff from different departments of the university are engaged. For example, course advisors approve the registration requests, technical staff configure student computer accounts, treasurers handle payment, and enrolment officers carry out paperwork. These staff who are distributed in terms of physical location and administration need to be coordinated efficiently to provide the registration service.

- Second, the student registration service may experience a heavy load from time to time. Thousands of students, new or current, may lodge their registration requests just before the deadline. Therefore, the performance of the registration processing system clearly is a major concern. The system should be capable of handling a large amount of requests in a short period of time.

With respect to these two characteristics, SwinDeW is evidently an applicable system for this application, as distribution and performance are two of the many features emphasised by SwinDeW.

### 7.1.2 System settings

For the purpose of simplicity, assume this student registration process involves only eight workflow participants (WfPs) with various capabilities for the time being, as summarised in Table 7.1. These capabilities are configured manually by system administrators and system engineers at the time when the system is started up. Each participant is associated with a SwinDeW peer (SP). When a peer is started up, the capability information of the associated participant is retrieved from the local initialisation file. In fact, the workflow participants involved can be dynamic. New staff may be recruited to work on this registration service, the existing staff may be on leave, and the existing staff may be trained to take over the work of the staff on
leave. All these can be supported by the registration service of SwinDeW, as discussed in Section 3.4.2.

Table 7.1 Summary of workflow participants and their capabilities in a student registration service

<table>
<thead>
<tr>
<th>Capability</th>
<th>Participant</th>
<th>Enrolment officer</th>
<th>Course advisor</th>
<th>Technical staff</th>
<th>Treasurer</th>
</tr>
</thead>
<tbody>
<tr>
<td>WfP1(SP1)</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WfP2(SP2)</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WfP3(SP3)</td>
<td></td>
<td>√</td>
<td></td>
<td></td>
<td>√</td>
</tr>
<tr>
<td>WfP4(SP4)</td>
<td></td>
<td></td>
<td>√</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WfP5(SP5)</td>
<td>√</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WfP6(SP6)</td>
<td>√</td>
<td></td>
<td></td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>WfP7(SP7)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>√</td>
</tr>
<tr>
<td>WfP8(SP8)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>√</td>
</tr>
</tbody>
</table>

Obviously, assisted by peers, these participants form various virtual communities according to their capabilities. Initially, the peer organisation at the time when the system is started up is depicted in Figure 7.2. This peer organisation also enables the peer discovery service as discussed in Section 3.4.1. The peers with the same capability naturally know each other. Peers in different communities can discover each other via a “gateway” peer. For example, SP4 connects to SP7 via SP5 and then SP6. Moreover, the connection between SP3 and SP6 is the result of an external management configuration so that SP3 and SP8 are not isolated from the other peers in the system. Based on this system setting, each peer forms its peer repository initially, as summarised in Table 7.2. The summary of the process repository and the task repository in this table will be detailed next.
Figure 7.2 Virtual communities of peers at the time when the system is initialised

Table 7.2 Summary of initial peer repositories, process repositories, and task repositories for two individual process instances

<table>
<thead>
<tr>
<th>WfP_i(SP_j)</th>
<th>Peer repository (see Figure 7.2)</th>
<th>Process repository (see Figure 7.1)</th>
<th>Task repository (see Figure 7.3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>WfP_1(SP_1)</td>
<td>SP_2, SP_5, SP_6</td>
<td>T_1, T_3, T_6, T_7</td>
<td>t_1^1, t_3^2</td>
</tr>
<tr>
<td>WfP_2(SP_2)</td>
<td>SP_1, SP_5, SP_6</td>
<td>T_1, T_3, T_6, T_7</td>
<td>t_6^1, t_1^2, t_7^2</td>
</tr>
<tr>
<td>WfP_3(SP_3)</td>
<td>SP_6, SP_8</td>
<td>T_5</td>
<td>t_5^2</td>
</tr>
<tr>
<td>WfP_4(SP_4)</td>
<td>SP_5</td>
<td>T_2</td>
<td>t_2^2</td>
</tr>
<tr>
<td>WfP_5(SP_5)</td>
<td>SP_1, SP_2, SP_4, SP_6</td>
<td>T_1, T_2, T_3, T_6, T_7</td>
<td>t_3^1, t_2^2</td>
</tr>
<tr>
<td>WfP_6(SP_6)</td>
<td>SP_1, SP_2, SP_5, SP_7</td>
<td>T_1, T_3, T_4, T_6, T_7</td>
<td>t_7^1, t_4^2, t_6^2</td>
</tr>
<tr>
<td>WfP_7(SP_7)</td>
<td>SP_6</td>
<td>T_4</td>
<td>t_4^1</td>
</tr>
<tr>
<td>WfP_8(SP_8)</td>
<td>SP_3</td>
<td>T_5</td>
<td>t_5^1</td>
</tr>
</tbody>
</table>
7.1.3 System support for build-time functions

By using SwinDeW’s build-time mechanisms, this process can be formally defined as a six-tuple, \( P : \langle TN, CN, RN, C, D, R \rangle \), which can be represented as a graph using the graph-based process representation discussed in Section 4.2. The visualised process representation using the process definition tool will be presented in Section 8.4.1 when the prototype implementation is discussed. During build-time, this process definition is also divided into seven task partitions with the partition algorithm presented in Section 4.4.1. Each task partition specifies a task as well as its position within the process of the student registration service and can be represented uniformly using the task notation presented in Section 4.3. For example, the task for \textit{Approval}, is represented as:

\[
T (00001; \\
002; \\
((001, enrolment officer, completed RSR form), straight); \\
((003, approved-RSR-form, enrolment officer), (007, enrolment officer, rejected-RSR-form), or)); \\
course advisor; \\
Specifics)
\]

All the task partitions are then distributed to the relevant peers using the mechanism described in Section 4.4.2. For example, definition of the task for \textit{Approval} will be distributed to all the peers that related to participants with the capability, \textit{course advisor}, i.e., peers SP\textsubscript{4} and SP\textsubscript{5} in Figure 7.2. Eventually, all the peers involved in this workflow obtain essential process definition according to the capabilities of their associated workflow participants. Process data which are represented by task partitions are stored in the process repositories of these peers, as shown in Table 7.2.
7.1.4 System support for run-time functions

Once a new registration request is received, a process instance needs to be created to handle this request. Various peers collaborate with one another to create a process instance, using the mechanisms described in Chapter 5. At the same time, an allocation policy is applied to this instantiation procedure to assign work dynamically in order to achieve good performance.

![Diagram of peer networks for two process instances](image)

**Figure 7.3 Different peer networks for two process instances**

When a registration request comes, the peer associated with an available enrolment officer creates an instance of the task for *Pre-examination* in order to examine the completeness and validity of the request. Then, this peer interacts with other peers to seek a capable participant who can instantiate the task for *Approval*. The peer is selected automatically through the use of an allocation policy. After that, the selected peer starts its own instantiation process to seek its successor peers. This
procedure repeats until all the required task instances are created and assigned dynamically. Eventually, each task instance \( j \) of the process instance \( i \), represented by \( i_j \), is inserted into the task repository of the corresponding peer. The process instance serving for this registration request is created finally as a network of peers performing various task instances. When the next request comes, another peer network is created dynamically, which may be different with the previous one. For example, the task instances in various peers’ task repositories, as shown in Table 7.2, represent two different process instances which can be visualised in Figure 7.3.

There is an absence of any centralised coordination for the enactment of process instances. All the communication and transmission are carried out directly between the relevant peers. A peer evaluates the start condition of a task instance independently, and starts to work when the condition is satisfied. After the work is finished, the peer performing it notifies the successor peer(s) and passes the related data by sending control and data messages, respectively. Using process instance one in Figure 7.3 as an example, the process instance is executed as follows⁶:

1. SP\(_1\) performs the task instance for *Pre-examination* in Figure 7.1 to check completeness and validity of a RSR form and then passes the form to SP\(_4\).

2. The start condition of the task instance for *Approval* is satisfied upon receipt of the RSR form. Hence, SP\(_4\) starts executing the task instance for *Approval* through the *StartTask* operation as discussed in Section 5.4.2. If the request is approved, the approved RSR form is sent to SP\(_5\). At the same time, a notification is sent to SP\(_6\), advising that the branch from SP\(_4\) to SP\(_6\) is skipped. Similarly, if the request is rejected, the RSR form with the comments is transferred to SP\(_6\) directly. A skipping notification is sent to SP\(_5\) and passed on to others.

3. SP\(_5\) executes the task instance for *Recording* upon receipt of the approved RSR form and sends control and data messages to both SP\(_7\) and SP\(_8\).

---

⁶ Erroneous situations and user interventions are not considered in this example.
Otherwise, if a skipping notification is received, SP₅ skips the task instance for *Recording* as well through the *SkipTask* operation as discussed in Section 5.4.2 and notifies the successor peers, SP₇ and SP₈, about the task skipping.

(4) Depending on the type of notification received from SP₅, SP₇ and SP₈ will execute or skip the task instances for *Account-management* and *Payment-handling*, respectively, in a similar way and notify SP₂ afterwards.

(5) SP₂ starts working when both the previous task instances have been completed because the task for *Collection* has an *And-Join* relationship with its two preceding tasks. Again, after the task instance for *Collection* is completed, SP₂ notifies SP₆. If a skipping notification is received from either SP₇ or SP₈, SP₂ skips the task instance for *Collection* because its start condition will never be satisfied.

(6) Finally, SP₆ executes the task for *File-closing* upon receipt of a completion notification from either SP₄ or SP₂. Receipt of a skipping message from either SP₄ or SP₂ will not provoke the skipping operation at SP₆, as the incoming structure of the task for *File-closing* is an *Or-Join* routing.

In this case study, an unavailable peer (staff) exception can be detected and handled automatically. For example, if SP₇ in instance one (in Figure 7.3) becomes unavailable intentionally before executing the task instance for *Account-management*, i.e., \( t_{4}^{1} \), this task instance can be re-allocated to SP₆ quickly, using the mechanism discussed in Section 5.4.3. Even if SP₇ becomes unavailable accidentally, this exception will be detected easily by SP₂ because peers exchange status messages periodically as discussed in Section 3.4.2. The status of the instance for *Account-management* can be determined as discussed in Section 5.4.3. Again, the task instance for *Account-management* will be re-delegated to an available staff member automatically. Then the registration request can proceed.
There could be other workflow participants involved in this process, playing the roles of monitoring and administration. For example, the administrators of the university could monitor this registration service and collect information such as the average processing time in order to analyse and evaluate the current process settings. Again, peers for these administrators communicate with peers for various workflow participants directly to obtain information about process status and historical data, with assistance of peers.

7.1.5 Summary of this section

Some benefits of SwinDeW are reflected within this case study. First, direct interaction between peers that perform adjacent tasks would decrease communication delay and thus may achieve good performance. Second, system robustness is likely to be enhanced because unavailability of any peer would not bring the whole system down. For example, when a peer associated with an enrolment officer becomes unavailable, the work assigned to this peer can be quickly reassigned to another enrolment officer for execution. Third, the system is much more scalable as new staff members can join the system easily to offer more processing capacity. Fourth, use of SwinDeW may satisfy staff members better. For example, enrolment officers can decide “who does what” amongst themselves since they are also involved in some managerial workflow functions. Finally, SwinDeW offers an open framework so that it is possible to incorporate some external services into this student registration service. For example, the task for Pre-examination could be carried out by some external education agents who provide services over the Internet.

7.2 Case study 2: management of a research project

The case in Section 7.1 discusses how SwinDeW supports complete workflow processes. In this section, SwinDeW support for incomplete workflow processes is illustrated in analysing a case of research project management.
7.2.1 Introduction

Research project management is regarded as an important, non-traditional application domain of workflow technology. In CICEC, workflow concepts have been long used in managing research projects, especially group projects, although formal workflow software support is less used. Very recently, some research projects are conducted in a similar fashion to the way a workflow process is executed in SwinDeW. In fact, some experience gained from real practice is eventually taken into consideration in designing the formal system framework and mechanisms of SwinDeW.

SwinDeW can serve as an effective tool for the management of research projects because of the following reasons:

• First, the project aims are normally achieved through the accomplishment of individual research tasks, which have inherently logical relationships and should be performed in a certain order. Thus, the conduct of a research project can be easily modelled as a process and SwinDeW can provide automated support for management of such a research process.

• Second, most of the non-trivial research projects are based on collaborative work. A number of researchers, who focus on various research tasks, are involved. These researchers, sometimes geographically distributed, should be coordinated properly so that research work can be passed from one researcher to another, according to a set of defined rules. Obviously, such coordination can be well supported by SwinDeW to improve efficiency and productivity.

• Third, there is a vast amount of communication amongst researchers for coordination purposes. Naturally, researchers communicate with one another directly through various means such as email, messages and face-to-face meetings. For example, a researcher who has finished some preliminary
research work might report the progress and research outcomes to other researchers. Thus, other researchers are able to start working on further activities based on the outcomes of previous research activities. SwinDeW also reflects this feature well with automatic coordination.

- Finally, managing the conduct of research projects with SwinDeW also helps to record, monitor, and trace research progress.

*Internet-based E-Business Ventures* is a CICEC group project, supported by the Swinburne Vice Chancellor’s Strategic Research Initiative Grant 2002-2004 ([http://www.it.swin.edu.au/centres/cicec/projects.htm](http://www.it.swin.edu.au/centres/cicec/projects.htm)). This project attempts to provide a leading-edge forum for the establishment, development, coordination, visualisation, and testing and evaluation of Internet-enabled e-business ventures that will lead to successful, new e-business models and supporting techniques. The specific initiatives this research targets are summarised as follows:

1. Development of a suitable e-business modelling environment utilising the current and substantive knowledge and data of many e-business descriptions and models. This initiative will extend knowledge of the domain by advancing the analysis, modelling and classification of Internet-based e-business ventures.

2. Visualisation/rendering of e-business models will be advanced by this work not only to enhance communication and comprehension of the models, but also to enable exciting possibilities in terms of simulation, feasibility testing and exploration of “what-if” scenarios.

3. Development of a suitable wide-area workflow framework as infrastructure support for e-business processes. This initiative will investigate seamless integration of portable data and tools with visualised environments. The innovative research on p2p-based decentralised workflow systems is a key area in this initiative.
7.2.2 System support for the incomplete project

The conduct of this research project is carried out according to the research proposal. However, some situations in this exploratory project are open-ended, sometimes unpredictable, and the process modelling information (how this project should be carried out step-by-step) at build-time (proposal writing and project planning) is insufficient. Hence, in the research proposal, the chief investigator and the project manager cannot model and specify the research process in depth at the early stage. In particular, the research process specified in the research proposal initially only enumerates the major research tasks and the logical relationships, execution orders and constraints among them. The objectives, methodologies and expected outcomes of each research task are clarified. However, the details of how to carry out some research tasks are not completely specified and are sometimes unnecessarily crystal clear. Therefore, researchers have to work around the system at run-time.

In view of this situation, the conduct of this project adopts the SwinDeW multi-level process modelling and execution approach. The research process initially obtained (after build-time) contains both composite research tasks and atomic tasks which are distributed to relevant researchers for execution. The composite tasks are decomposed gradually during the conduct of this research project by executing the associated decomposition tasks. Researchers are subsequently coordinated to carry out sub-tasks generated by the decomposition. These problems are addressed using the mechanisms for incomplete process support in SwinDeW as discussed in Chapter 6.

Figure 7.4 shows a part of this project in terms of the process representation at build-time. The visualised process definition using the process definition tool will be covered in Figure 8.7 (Section 8.4.4) when prototype implementation is discussed. In this process, the task for initial literature review is carried out first. Then, the tasks for e-business modelling research and workflow research are carried out by researchers in two research groups in parallel. Finally, the task for integration of facilitating workflow systems for e-business processes needs to be carried out by researchers in both research groups jointly, integrating the achievements of two
tasks. Obviously, at the early stage of the project, the tasks for *e-business modelling*, *workflow research* and *integration* can only be modelled as composite tasks. Although the goals and expected outcomes of each task are expressed, the method of achieving the goals through a set of steps remains uncertain. The particular research schedule for each task can be gained only after some initial work such as initial literature review has been completed. In other words, the decomposition of composite tasks into sub-processes should be performed on-the-fly at run-time. For this reason, decomposition work is modelled explicitly as essential steps before the execution of the composite research tasks, as shown in Figure 7.4.

![Figure 7.4 Part of CICEC research process](image)

The decomposition tasks require authorised participants to carry them out. In particular, decomposition of the tasks for *e-business modelling research* and *workflow research* needs to be conducted by the leaders of two research groups, respectively, and decomposition of the task for *integration* needs to be executed by the project manager. Therefore, these decomposition tasks, as well as the description of the three composite research tasks are distributed to authorised participants, respectively. Process instantiation is then carried out using the mechanisms proposed in Chapter 5. Various tasks are instantiated by capable researchers and the instances of the decomposition tasks and composite tasks are also created by the corresponding participants. After that, to enact this part of the process, the researcher committed to undertake the task for *initial literature review* starts executing the task first. The leaders performing the tasks for *e-business modelling* and *workflow research* are responsible for decomposing the corresponding composite tasks, using
either the push or the pull mode described in Section 6.4.4. Using the task for workflow research as an example, the leader of the workflow research group is able to specify how the research should be conducted at the right time. The corresponding decomposition task is executed, resulting in a sub-process that consists of four sub-tasks which should be executed in sequence, as shown in Figure 7.5. Subsequently, these four tasks are assigned to the researchers in the workflow research group for execution. In this case, the four tasks in this sub-process, i.e., system design, build-time functions, run-time functions, and prototyping, are executed in sequence using the mechanisms for complete processes. During the execution or after the execution of this sub-process, the project manager who is in charge of decomposing the task for integration can arrange the full specification for the integration task.

![Figure 7.5 Sub-process of workflow research](image)

After the decomposition of the task for workflow research is made, the group leader coordinates the execution of the generated sub-process. Since there is only one instance for this process normally, the decomposition can be regarded as at either the process- or the instance-level. Although instance-level decomposition is in general easy to handle, process-level decomposition is more suitable in this scenario. This is because process-level decomposition results in a complete process specification for record purposes. Hence, in practice, this decomposition is regarded as process-level decomposition. The partitions of sub-tasks are distributed to relevant peers and an instance of the sub-process is created by the researchers involved. These researchers execute various task instances and interact with one another directly for the purpose of coordination. The researchers who perform the
task for *system design* and the task for *prototyping* also interact with researchers performing preceding and succeeding tasks, respectively. At the same time, the group leader who initially instantiates this composite task plays the role of an administrator and collects information about the sub-process execution from the relevant researchers. In addition, this decomposition will not cause any inconsistency because there are no other running instances.

### 7.2.3 Summary of this section

In summary, by using the SwinDeW multi-level process modelling and execution approach proposed in Chapter 6, decomposition of complex research tasks is modelled as intermediate steps towards the completion of a research project. Qualified participants (e.g., the group leaders and the project manager) are involved in run-time process specification naturally. Moreover, SwinDeW decentralised process coordination technologies for complete processes are also used in this incomplete process support scenario seamlessly.

### 7.3 Summary

This chapter presents two cases to justify the soundness of the key ideas proposed in this thesis. The studies have demonstrated the SwinDeW system design and corresponding mechanisms clearly. The analysis also illustrates that the SwinDeW p2p-based workflow approach proposed in this thesis is practically feasible for supporting both traditional and complete, and non-traditional and incomplete processes.
Chapter 8

Prototype implementation

To demonstrate and prove the soundness and feasibility of the key ideas presented in this thesis, a prototype has been developed which is based on Sun Microsystems JXTA which is a p2p framework. In this chapter, issues of prototype implementation are discussed.

First, Section 8.1 introduces JXTA briefly. Second, the overall prototype architecture is discussed in Section 8.2. Third, major functionality implemented in the current prototype is summarised in Section 8.3. Finally, implementation of the key components in this prototype is described in Section 8.4, including implementation of the process definition tool in Section 8.4.1, implementation of peer organisation in Section 8.4.2, implementation of the peer kernel in Section 8.4.3, implementation of incomplete process support in Section 8.4.4, and implementation of the local visualisation module in Section 8.4.5.

8.1 JXTA peer-to-peer framework

The JXTA project (http://www.jxta.org) is an active and progressive project incubated at Sun Microsystems, dealing with current p2p problems. JXTA is a set of open, generalised, p2p protocols that allow any connected devices on the network to communicate and collaborate with one another. For the purpose of interoperability, the project does not bind itself to one particular company, programming language, system or network and provides platform-independent solutions for p2p applications.
For developers, the JXTA project provides a set of building components that provide infrastructure support for genuinely distributed applications. Common functions required by any p2p system can be supported by using JXTA. Therefore, software developers can focus on the application itself rather than re-developing their own framework. On the JXTA platform, a peer is any networked device that implements one or more of the JXTA protocols. Peers self-organise into peer groups, each of which is a collection of peers that have agreed upon a common set of services. To enable peers to advertise and discover each other, and communicate and route messages, the current JXTA platform supports six JXTA protocols, which are:

- Peer Discovery Protocol (PDP) which is used by peers to advertise their own resources and discover resources from other peers.
- Peer Information Protocol (PIP) which is used by peers to obtain status information from other peers.
- Peer Resolver Protocol (PRP) which enables peers to send a generic query to one or more peers and receive a response.
- Pipe Binding Protocol (PBP) which enables peers to establish a virtual communication channel with other peers.
- Endpoint Routing Protocol (ERP) which is used by peers to find routes to destination peers.
- Rendezvous Protocol (RVP) which is used by peers to subscribe a propagation service.

Since 2001, JXTA has been a popular, open-source, royalty- and license-free p2p framework which involves a large number of registered members of the
development community\textsuperscript{7}. Actively supported by a growing community of p2p developers, JXTA technology has seen strong growth in its adoption, and commercial offerings are now becoming available. For these reasons, SwinDeW places itself easily under the JXTA framework.

### 8.2 Overall prototype framework

The current prototype is written in the Java\textsuperscript{\textregmark} programming language utilising J2SE version 1.4 API. This prototype takes advantage of the JXTA grouping feature for virtual community management. Communication relies on JXTA messaging protocols such as advertisement and pipe. The messages that are traded between peers use XML format. As depicted in Figure 8.1, this prototype consists of two separate software components performing workflow functions and p2p functions, respectively.

\textbf{Figure 8.1 Framework of SwinDeW prototype}

\textsuperscript{7} More than 18,200 as accessed on 27/08/2004
The core services of JXTA include the group service, the peer service, the pipe service, the discovery service and the advertisement service. The description of the core services are given as follows:

- **Group service**: this service coordinates the grouping strategies of p2p applications. Each group can have a membership policy. For example, the capability concept in SwinDeW is the group concept in JXTA. The group service coordinates with PeerGroup of the JXTA API to provide peers with the ability to discover groups, retrieve information about all the members in a group, create a pipe to broadcast in a group, join a group, leave a group, etc.

- **Peer service**: this service is central to managing the peers in the JXTA virtual network. Each peer publishes a network interface through which direct point-to-point connections can be established between two peers.

- **Pipe service**: this service has capabilities to manage the communication between two peers. The service can search for a pipe advertisement in the JXTA virtual network using the discovery service. This service consists of two sub-services: the InputPipe service and the OutputPipe service, which deal with sending messages and receiving messages, respectively. At the same time, pipes offer two types of communication, point-to-point and propagate, which can be used in different situations.

- **Discovery service**: this service is used to search for advertisements in the JXTA virtual network.

- **Advertisement service**: this service is used to publish contents in the JXTA virtual network. There are various advertisement types in JXTA such as peer advertisement, peer group advertisement, pipe advertisement, peer information advertisement, and so on.
The core services of SwinDeW, including the peer management service, the process definition service, the process enactment service and the monitoring and administration service, are realised through the invocation of the JXTA core services. Moreover, the invocation of JXTA services is encapsulated in a set of self-developed JXTA network interfaces, which make the JXTA implementation transparent to the application development.

Such a prototype framework attempts to exploit the computing capacity of the entire system. This is because JXTA provides a genuinely decentralised environment for applications. Therefore, computation and communication loads are distributed naturally and good system performance is expected to achieve. It is also noted that the current version of the SwinDeW prototype is implemented for proof-of-concept purposes. The management of sample workflow processes, which will be described next, demonstrates the feasibility of the key ideas of SwinDeW. However, statistical results for criteria like execution time are not collected for the measurement of system performance to support the claim of benefits in terms of performance. This is because most workflow processes supported by workflow management systems (including SwinDeW) require human interaction in order to carry out tasks. Execution time of process instances, which includes user-computer interaction time, is always inaccurate in reflecting performance of the underlying system. Even if the user-computer interaction time is removed (e.g., by using timestamp), it is still not very practical to measure system performance statistically at this stage. Like Grid computing, the claimed advantage of SwinDeW in terms of performance normally becomes evident when it is used to support computation-intensive applications or when it coordinates a big number of workflow instances simultaneously. In practice, it is rather difficult to collect a huge amount of data simply based on a research prototype. At the same time, many other factors which are related to implementation methods rather than system design may affect the overall system performance. On the other hand, based on the in-depth analysis of the origins of the problems addressed in this thesis (as discussed in Section 2.1), the compelling advantages that p2p offers (as discussed in Section 3.1), and the innovative design of system framework and mechanisms (as discussed in Chapters 3, 4, 5 and 6), it is believed that the descriptive discussion in this thesis is sufficient
for the claimed advantage of SwinDeW. In the future, experimental simulation may be designed carefully to eliminate impacts of other factors and collect data properly for performance analysis purposes.

Another direct advantage of this prototype framework is increased system robustness. Based on the JXTA framework, the unavailability of a single peer normally has local impact on the system. Issues of scalability are also addressed in JXTA intensively. SwinDeW can largely use the powerful group service in JXTA to provide improved scalability. Moreover, such a prototype framework can also provide enhanced system openness by using the JXTA discovery service. It is possible for SwinDeW peers to search for external services on the Internet which are published via the JXTA advertisement service. SwinDeW peers are then able to communicate with external service providers via the peer service and the pipe service in order to integrate these services into the system.

8.3 Overview of SwinDeW functionality

So far, the major build-time and run-time mechanisms of SwinDeW for both complete and incomplete processes as discussed in Chapter 3, 4, 5, and 6 have been implemented. These mechanisms are summarised in Table 8.1.

Table 8.1 Summary of functionality in SwinDeW prototype

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process definition</td>
<td>Use WfMC’s XPDL for definition portability and XML documents for storage. Processes are defined through the use of a GUI. The implementation of this functionality will be detailed in Section 8.4.1.</td>
</tr>
<tr>
<td>Virtual community management</td>
<td>Group and organise peers in virtual communities according to capabilities, as detailed in Section 8.4.2.</td>
</tr>
</tbody>
</table>
Definition distribution | Divide a process into task notations and distribute partitions to relevant peers, as detailed in Section 8.4.3.

Process instantiation | Create a process instance in a decentralised way, as detailed in Section 8.4.3.

Dynamic work allocation | Allocate task instances dynamically according to selection policies, as detailed in Section 8.4.3.

Instance execution | Coordinate instance execution through message exchange as detailed in Section 8.4.3.

Monitoring and administration | Perform monitoring and administrative functions, as detailed in Section 8.4.3.

Incomplete process support | Model processes incrementally and specify incomplete processes at run-time for both process- and instance-levels, as detailed in Section 8.4.4.

Local visualisation | Visualise workflow processes as directed paragraphs for the purpose of useability, as detailed in Section 8.4.4.

### 8.4 Implementation of key system components

The implementation of the key components of the SwinDeW decentralised workflow management system are described in the following sections.

#### 8.4.1 Process definition tool

The process definition tool is an integrated system component which can be invoked by authorised peers to model workflow processes, create process definitions, and distribute task partitions in SwinDeW. This tool uses different nodes and edges to
visualise key process components and their relationships such as tasks and resources, as discussed in Section 4.2. So far, this tool supports both complete and incomplete workflow processes discussed in this thesis.

Figure 8.2 Process definition tool illustrating a complete process

Figure 8.2 shows the user interface of the process definition tool. The sample workflow process modelled in this figure is the student registration service which is discussed as a case study in Section 7.1. It can be seen from this figure that task nodes, control nodes and resource nodes of the process are represented by symbols with different shapes and colours, i.e., green circles represent task nodes, red rectangles represent control nodes, and yellow, round-corner rectangles represent resource nodes. At the same time, data and control relationships between tasks and the resource requirement of each task are represented by different edges, i.e., solid edges represent data and control relationships and dashed edges represent resource requirements.
A workflow process generated as a graph by this definition tool can be converted into XML format and saved as an XML file automatically. As discussed in Section 4.1, a workflow process can be represented in different forms using different process definition languages. Of these definition languages, XPDL, which stands for XML Process Definition Language, is a workflow process definition interface released by WfMC (http://www.wfmc.org/standards/docs/TC-1025_10_xpdl_102502.pdf). The purpose of the release of XPDL is to provide a common meta-model for describing the process definition, thus supporting process definition import and export in a consistent form. As WfMC is the largest standardisation body for workflow management, its standards are most widely accepted in workflow products. Hence, SwinDeW employs XPDL as its process definition language for the purpose of interoperability.

XPDL belongs to the family of graph-structured process definition languages. The meta-model of XPDL utilises the concept of activity (workflow process activity) as the core component of the (workflow) process definition. The workflow process definition is built from individual workflow process activities that are related together to form a control flow via transitions which are governed by transition information. XPDL describes a workflow process definition in terms of what is to be done, when it has to be done, under what conditions, and by whom. To achieve this, XPDL contains most of the constructs likely to be required in the exchange of process definitions. Moreover, to deal with the circumstances under which additional information will need to be included within a process definition, XPDL also provides a standardised means for expressing the extension, which includes extended attributes and extended parameter mapping. Figure 8.3 shows part of the student registration process modelled in Figure 8.2, which is defined in XPDL.
8.4.2 Implementation of peer organisation

As discussed in Section 3.3.2, SwinDeW uses the concept of virtual communities to manage various peers in an organised manner. Peers with the same capability are grouped into one virtual community. When a peer is started up, its capabilities are normally configured by system administrators and system engineers, as shown in Figure 8.4. Accordingly, this peer joins relevant communities and acquires knowledge about other peers. This knowledge enables direct communication between two peers.

In the SwinDeW prototype, management of peer organisation is implemented in a separate package, which consists of three modules interacting with one another, i.e., Peer Pipe, PeerGroup Manager and Peer Neighbour modules. Figure 8.5 depicts the internal structure of the peer organisation package. The details of the modules in this package are described as follows:
• Peer Pipe: this module has capabilities to manage the communication between two peers. Each Peer Pipe consists of an InputPipe and an OutputPipe which are able to receive and send messages, respectively. The Peer Pipe can search for a pipe advertisement in the JXTA network using the JXTA Discovery service. A call-back event is then sent to the PipeAdDiscovery Handler when a Peer Pipe finds a new advertisement.

• PeerGroup Manager: this module manages the capabilities in SwinDeW. The PeerGroup Manager coordinates with PeerGroup of the JXTA API to provide a peer with the ability to discover virtual communities, retrieve information about members in a virtual community, create pipes to broadcast in the community, join a community, leave a community, and so on. Accordingly, various events will be fired to the PeerChange Handler which handles changes.
Peer Neighbour: this module manages the information about the known peers in the system which forms the organisational model for the particular peer. The Peer Neighbour holds the advertisement of the peers and pipes for communication purposes.

Peer Kernel in Figure 8.5 is a collection of other components and modules within a peer. Given the support of the Peer Pipe, the Peer Neighbour and the PeerGroup Manager in this package, the Peer Kernel is able to perform various workflow build-time and run-time functions, which will be detailed in Section 8.4.3.

The implementation of the peer organisation in the SwinDeW prototype enables enhanced scalability. The modules of Peer Neighbour and PeerGroup Manager help to build dynamic and scalable peer communities as discussed in Section 3.3.2. At the same time, when a new peer joins the system, it searches for any pipe advertisement through the Peer Pipe module rather than contacting a particular site. Hence, a new peer can join the system easily by sending the joining request to any discovered pipe, using the mechanism discussed in Section 3.4.2.
8.4.3 Implementation of Peer Kernel

At the heart of the SwinDeW prototype, the Peer Kernel of each of the dispersed peers collaborates with one another to provide workflow automation. Generally speaking, each peer works independently in the system, according to the application rules (workflow definition), the peer’s commitment to the process, interaction context, etc., and contributes to the operation of the whole workflow system. Moreover, peers interact with one another directly to enable the workflow system to work properly as designed, as discussed throughout this thesis.

In the SwinDeW prototype, the interaction between two peers is realised through message exchange. Therefore, many types of messages are defined in the prototype for different communication purposes. Each message is interpreted and processed by the Peer Kernel to perform the required workflow functions.

Figure 8.6 described the internal structure of a Peer Kernel and the flowchart of message processing. Messages received through an InputPipe are placed in a message queue first. Then, a message processor reads each message from the queue, interprets the message properly, and dispatches the content of the message to one of the five handler modules for processing. After a message is processed, corresponding events are fired and placed in an event queue by the processing handler. An event is handled by the event processor and where applicable, outgoing messages are generated. Finally, generated messages are placed in an outgoing message queue and are sent out by an OutputPipe.

The five handler modules are Initialisation handler, Definition synchronisation handler, Enactment distribution handler, Process enactment handler, and Monitoring & administration handler. Each of these five handler modules is designed to handle a set of pre-defined messages. The details of these handler modules are described as follows:
The Initialisation handler deals with issues occurring when a new peer is started up, including looking for global pipes, creating pipes and advertisements, joining virtual communities, etc.

The Definition synchronisation handler looks for neighbour peers and requests corresponding process definitions from neighbour peers, as discussed in Section 4.4. A MSG_REQ_PROCESS_DEFINITION message with the capabilities of the sender is sent to neighbour peers. The definition synchronisation handlers of neighbour peers respond with the MSG_PROCESS_DEFINITION messages upon receipt of a request message.

The Enactment distribution handler creates task instances according to process instantiation algorithms and allocation selection policies, as discussed in Sections 5.1, 5.2 and 5.3. A MSG_INITIATE_INSTANCE...
message is sent to capable peers in order to request the creation of a task instance. Then, the selected peer sends out a MSG_ACCEPT_INSTANCE message to accept a request. After that, further MSG_INITIATE_INSTANCE message(s) will be sent out to seek successor peer(s). In addition, the Enactment distribution handler also supports centralised instantiation of (sub-)processes in the situation that instance-level decomposition may require client-server based coordination as discussed in Section 6.4.5. In this case, all the MSG_INITIATE_INSTANCE messages are issued by the server peer which is in charge of the composite task.

- The Process enactment handler manages the execution of task instances, including evaluating the start condition, updating the instance status, and so on, as discussed in Section 5.4. During process enactment, a number of MSG_PREDECESSOR_STATUS and MSG_SUCCESSOR_STATUS messages are transmitted between adjacent peers, indicating the updated status of task instances to predecessors and successors, respectively. At the same time, MSG_PREDECESSOR_DATA and MSG_SUCCESSOR_DATA messages are transferred accordingly to deliver application data related to the process instance. Again, the process enactment handler also supports centralised process enactment where the server peer evaluates the start condition of each task instance.

- The Monitoring & administration handler monitors process execution and records execution information, as discussed in Section 5.5. Various types of MSG_ADMIN_MSG messages flow from one peer to another, delivering administrative information. At the same time, messages regarding exceptional situations are also processed by this handler for the purpose of exception handling.

Based on the implementation of the Peer Kernel, ordinary peers are involved in process enactment actively through Process enactment handler in order to enable process instance execution step-by-step in the right order. Moreover, ordinary peers
are also involved in administration duties more-or-less through handlers such as Definition synchronisation, Enactment distribution and Monitoring & administration handlers. This involvement may provide ordinary participants with more information, control, freedom and self-realisation. Therefore, it is clear that user support is emphasised in the implementation of the SwinDeW prototype.

8.4.4 Implementation of incomplete process support

Incomplete processes are supported in the SwinDeW prototype at both build-time and run-time, as discussed in Chapter 6. The process definition tool described in Section 8.4.1 also supports incomplete process modelling. Figure 8.7 depicts an incomplete process which is modelled using this process definition tool. This process is part of a research project which is discussed as a case study in Section 7.2. In this figure, an incomplete process is represented through the modelling of composite tasks and decomposition tasks. Composite tasks are defined explicitly and represented by blue circles (e.g., T5: Workflow research) in this figure. At the same
time, decomposition tasks which are represented by purple circles (e.g., T3: Decomposition) in this figure are also created and associated with composite tasks appropriately. Both the composite and the decomposition tasks require special capabilities to carry them out. For example, the composite task for workflow research and the associated decomposition task in Figure 8.7 require the capability of workflow research leader.

**Figure 8.8 Run-time support for incomplete processes**

At run-time, an incomplete process is supported using the process definition tool, the existing message exchange mechanism and the handler modules. Figure 8.8 describes run-time support for incomplete processes in SwinDeW. First, the enactment of a decomposition task decomposes an associated composite task into a sub-process by invoking the process definition tool to draw the details of the sub-process. For example, the modelling of the sub-process of workflow research discussed in Section 7.2 is shown in Figure 8.9. If the decomposition occurs at the process level, the Definition synchronisation handler and the relevant message set are used to re-distribute the definition of the sub-process. This step is skipped if the decomposition occurs at the instance level. After that, an instance of the sub-process is created for the present process instance using the Enactment distribution handler.
and the relevant message set. Finally, the Process enactment handler and the relevant message set are used to enact the instance of the sub-process properly as defined.

![Figure 8.9 Run-time modelling of the a composite task](image)

### 8.4.5 Implementation of local visualisation module

In order to offer a friendly user interface to workflow participants, the local visualisation module of each peer is implemented. The functionality of this module is to convert the XML formatted process representation into graph-based representation. Hence, the local visualisation module provides a graphical user interface for a workflow participant to view both process graphs for the details of process elements and instance graphs for the status of process instances, locally.

The implementation of the local visualisation module is illustrated in Figure 8.10. This visualisation module consists of two elements: a visualisation Servlet and a user interface. The visualisation Servlet of the visualisation module retrieves relevant process information from data repositories and interprets the information properly. At the same time, the user interface of the visualisation module draws different symbols and lines on the screen to represent various process elements and
their relationships graphically. From Figure 8.10, it is also clear that the local visualisation module can read data from two types of data repositories directly: the process repository and the task repository. The former stores process definition data and the latter stores process instance data. Therefore, the visualisation module can visualise both process- and instance-level data.

Since each peer normally has partial process data, the local implementation module of a peer does not display the graph of the complete process. Instead, the implementation module displays individual tasks and fits them into the context of a process. That means for each individual task, only its direct preceding and succeeding tasks will be displayed when the participant views this task. Figure 8.11 shows the visualised representation of a peer’s process definition. The task partitions described in this figure are process data of a peer in the student registration service discussed in Section 7.1. The process definition table lists all the task partitions in accordance with the peer’s capabilities. Each task only concerns the adjacent tasks and their relationships. The visualised representation of a peer’s task instances, as depicted in Figure 8.12, is very similar to the visualised representation of process definition in Figure 8.11. However, the instance representation also shows information about predecessor and successor peers in this process instance. Moreover, statuses of task instances, i.e., unacted, enacting and enacted as discussed in Section 5.4.2, are indicated by different colours in this prototype.

Figure 8.10 Implementation of local visualisation module
Figure 8.11 Visualised representation of a peer’s process definition

Figure 8.12 Visualised representation of a peer’s task instances
Although this module is designed to support local visualisation for ordinary peers, it can also be extended to support global visualisation for administrator peers which may have global view of a process and/or a process instance. In this case, various task partitions with the same Process-id will be connected together and visualised as a single graph.

8.5 Summary

Based on the JXTA framework, the SwinDeW prototype has been implemented for demonstration and proof-of-concept purposes. XPDL is used in this prototype as the process definition language for the purpose of interoperability. So far, the results of implementation are promising. In particular, the JXTA technology is very suitable for the implementation. Complexity of the support infrastructure and code is reduced largely, as the JXTA technology provides complete, developer transparent functionality and rich interfaces. The implementation is complete, as all the mechanisms for build-time and run-time workflow functions as discussed in Chapters 3, 4, 5 and 6 have been realised. Therefore, the approach proposed in this thesis is proven to be technically sound and feasible.
Chapter 9

Discussions

This chapter discusses the pros and cons of SwinDeW, the p2p-based decentralised workflow management system proposed in this thesis. The advantages of this approach are summarised in Section 9.1. On the other hand, the tradeoffs to using the p2p computing technology in comparison to the client-server architecture are explained in Section 9.2 in order to give a balanced view of this research. Moreover, a discussion of what application domains SwinDeW may be more suitable for is given in Section 9.3

9.1 Discussion on the advantages of this research

Processes are an essential element in all workplace organisations. In this Internet era, workflow management for processes is becoming an increasingly important part of organisational information systems. So far, the reality is that most workflow systems have been criticised because of the unsolved problems discussed in this thesis, i.e., poor performance, vulnerability to failures, limited scalability, user restrictions, unsatisfactory system openness, and lack of incomplete process support.

The approach presented in this thesis views the above problems of workflow from a different perspective. It is believed that most of these problems, if not all, arise due to the mismatch between system requirements and system realisation. Hence, centralised management based on client-server architecture which is not suitable for decentralised workflow applications needs to be replaced by an open,
collaborative, and decentralised, framework. With this observation, the aim of this research is to address these unsolved problems rudimentally by exploiting the unique features of p2p technology which provides a genuinely decentralised architecture to support workflow. Subsequently, a new framework and corresponding process coordination technologies are presented. As a response to the research problems analysed in Section 2.1, the advantages of the proposed workflow system, SwinDeW, are summarised as follows:

(1) SwinDeW enables better system performance because it completely distributes both data and control to highly utilise the computing and storage capabilities of the entire enterprise. The decentralised coordination provided by SwinDeW is regarded as light-weight and cost-effective, which may minimise interaction-related delays. In addition, the dynamic work assignment based on load-balancing, which is another distinguishing feature of SwinDeW, also contributes to achieving better system performance and increasing agility.

(2) Compared with the client-server based, or partly client-server based approaches, SwinDeW involves much less risk where the whole system fails or declines simply because some individual peers (bottlenecks) are overwhelmed by heavy computation. This is because

   (i) the computation and communication loads are well balanced across the network in SwinDeW to avoid bottlenecks,

   (ii) peers’ departure is supported explicitly in SwinDeW, and

   (iii) the automatic handling of “unavailable peer” exceptions also adds robustness to the system.

(3) System scalability is also enhanced as peers retain a loosely-coupled topology in SwinDeW. Virtual communities are dynamic so that workflow participants can come and go. These participants’ dynamic behaviours do not require modifying and updating the centralised workflow server as in the client-server
approaches. New peers can easily join the system at any time and through any existing peer with no need to change the settings of a particular site. Therefore, this perspective of system scalability of SwinDeW is enhanced. The system is capable of coping well with dynamic changes of the system size.

(4) SwinDeW loosens restrictions on workflow participants and optimises human involvement. Workflow participants, assisted by peers, are autonomous in the system. The novel philosophy of “know what you should know” is proposed for peers to gain more data. With essential data, human beings are able to participate in workflow systems more actively than ever before. Ordinary participants can express themselves (e.g., preference for tasks) and enjoy the abilities of being involved in system management when necessary (e.g., work allocation). At the same time, direct communication amongst the participants which is assisted by peers allows for coordination in a more suitable way. SwinDeW could also help to provide knowledge-based repositories containing information on similar cases, participants, best practices, and so on for potential contributors. Hence, this approach offers broader user assistance which is an important feature needed in teamwork.

(5) SwinDeW utilises novel techniques involving p2p execution of workflow processes. This research delivers research results useful for research topics such as Web service workflow and Grid workflow, as certain requirements such as p2p service interaction were found lacking in the existing Web services and Grid services technologies [KWL02]. Since the proposed open model naturally exploits the distributed nature of the Internet, it would increase the system openness and support service-oriented workflow well. In particular, the composition and execution of Web services can be facilitated properly by peers through techniques such as publish and subscribe [BDSN02, PFW03].

(6) SwinDeW provides adequate support for incomplete processes. By using hierarchical process modelling and execution, and decentralised task decomposition, support for incomplete processes is incorporated within the existing system framework and mechanisms seamlessly. This extended feature
makes SwinDeW capable of supporting processes in non-traditional application domains where processes cannot be completely specified beforehand.

9.2 Discussion on the possible Tradeoffs of the proposed approach

However, moving from client-server to p2p may bring some tradeoffs which can be potential limitations. Some of the tradeoffs of the proposed approach are summarised as follows, although they are outweighed by many advantages it offers:

(1) Although SwinDeW decentralises both build- and run-time functions, system initialisation, which is normally carried out centrally in client-server based workflow approaches, still needs to be carried out centrally, sometimes through more elaborate configuration and management. Ordinarily, starting this system might involve system engineers and administrators who manually configure the system and start up the services as needed. The initial status of each peer, including the capability and authority, needs to be configured properly. Also, some external tools such as organisational management and organisational configuration tools may need to be used to ensure community connectivity. However, this tradeoff can always be ignored because system initialisation is normally a one-off occurrence which has to be carried out in all circumstances one way or the other.

(2) As a tradeoff, management and monitoring of workflow execution may become more difficult in the SwinDeW p2p-based workflow system. Extra peers (administrator peers) need to be developed for administrators in order to collect the related information (workflow events, instance status, performance data, historical information, etc.) through communication with ordinary peers. Since peers associated with administrators normally do not coordinate process enactment, they may require more complicated mechanisms to manage process execution when needed. This issue can be further addressed by designing peers for global view and global information, which will be enhanced in the future.
(3) The ability to handle exceptions and erroneous situations may be impaired in SwinDeW. Unlike client-server based workflow systems where errors and exceptions can be detected and handled by centralised servers, SwinDeW requires more complicated mechanisms to deal with aspects of flexibility such as general exception handling and dynamic change handling, which are not addressed sufficiently in this thesis and are regarded as future work.

(4) P2p applications enable networked access to resources. This can lead to security problems [VCM01]. The potential openness of SwinDeW may make it even easier for hackers to attack the system maliciously, to steal sensitive information, or just to run unauthorised commands on someone else’s machine. Therefore, issues of authentication and security may need to be addressed particularly for certain applications. Some existing security mechanisms such as digital signature can be used to counteract this tradeoff.

(5) The approach presented in this thesis assumes that ordinary workflow participants have the abilities to perform some basic management operations. This approach is probably not appropriate when the management operations are very complex and require high-level skills. Moving towards p2p-based workflow requires the workflow participants to be further trained in p2p concepts, role profiles, and workflow processes. However, such training is always worthwhile in any organisations as part of staff development.

(6) The still nascent p2p industry lacks common protocols. There lacks standards for p2p scalability, security, management and interoperability, which does create some concerns. For this reason, WfMS interoperability in intra- or inter-organisational settings is not addressed in the proposed approach and will be discussed as future work. However, with the rapid growth of industry standards in areas such as Web services and Grid computing, this tradeoff will diminish in the near future.
(7) Due to the features of p2p, there may be some other potential concerns of p2p solutions for workflow systems. These may include increased network usage and possible network overload, difficulties in coordinating large number of peers, possibilities of inconsistency introduced by data replication, as discussed as follows.

(i) Although the overall network usage may be increased, the increased traffic is distributed over the network, e.g., broadcasting within the virtual community, and is very unlikely to cause network overload. Most of the messages exchanged in SwinDeW, e.g., the messages to coordinate peers, are also needed in client-server based workflow systems, e.g., the messages to coordinate the clients and the server. Moreover, SwinDeW may even reduce some network usage, as intermediate data are passed between two peers directly without exchanging through a third peer (server).

(ii) Coordinating large number of peers seems not to become obviously more difficult in SwinDeW. By using virtual communities to organise peers, SwinDeW is able to coordinate large number of peers for coordination.

(iii) Process data replication may introduce possibilities of inconsistency. This issue has been addressed partly in this thesis. For example, a peer removes its process definition data when it leaves the system to avoid data inconsistency when it re-joins the system later on, as discussed in 4.4.3. In the future, investigation into issues such as support for multiple versions of the same process may be carried out to address this problem further.
9.3 Discussion on suitable application domains of SwinDeW

Given the discussion of the advantages and tradeoffs of SwinDeW outlined in Sections 9.1 and 9.2, it is clear that p2p-based workflow systems, like SwinDeW, may display greater benefits within some application domains whilst it may not particularly suit some other workflow application domains. Therefore, an analysis of the application domains where SwinDeW is more suited can provide a greater understanding of p2p-based workflow systems and will contribute to the future development and deployment of p2p-based workflow systems. This thesis discusses this important issue based on the workflow continuum specified in Figure 2.2, which identifies four categories of workflows.

In general, SwinDeW is capable of providing better support for standard workflows (Category Four, Standard WF shown in Figure 2.2), providing the advantages outlined in Section 9.1. Moreover, SwinDeW is capable of providing adequate support for incomplete processes (the first two subcategories in category three in Figure 2.2), as discussed in Chapter 6. However, SwinDeW may not be ideal for supporting workflows in other categories and subcategories outlined in Figure 2.2. First, SwinDeW is not ideal for supporting non-repetitive workflows like ad-hoc workflows (category one in Figure 2.2) as partition, distribution and maintenance of non-reusable processes may result in more costs than benefits. Second, category two in Figure 2.2, i.e., autonomous workgroup, is not discussed in this thesis, as it is not related to workflow management exactly, as outlined in Section 2.5.1. Finally, the SwinDeW system presented in this thesis is also not ideal for supporting workflows with exceptions, i.e., workflows in the third subcategory in category three (semi-structured WF in Figure 2.2), as exception handling becomes more difficult in decentralised workflow systems. As indicated in Section 9.2, this aspect is not addressed sufficiently in this thesis and forms a section of future work.

9.4 Summary

In summary, SwinDeW exploits the strengths of p2p technology to better manage and automate critical processes within an organisation, especially standard
workflows and workflows with unspecified elements. This approach is utterly pragmatic and will provide significant value to organisations. The deployment of SwinDeW will contribute much to empower organisations and provide a competitive edge.

On the other hand, the advantages of SwinDeW p2p-based decentralised workflow are achieved at the cost of some compromises. Particularly, ad-hoc workflows and workflow exceptions have not been supported sufficiently so far. Some of these tradeoffs have been discussed acceptably in this thesis. However, extra attention should be paid to some other tradeoffs in future research and practice of SwinDeW in order to counteract and minimise the limitations.
Chapter 10
Conclusions and future work

10.1 Summary of this thesis

The objective of this thesis was to investigate an innovative framework and process coordination technologies for peer-to-peer based, decentralised, workflow management systems. The thesis was organised as follows:

- Chapter 1 introduced workflow concepts as well as the state-of-the-art and the state-of-the-practice of workflow. Chapter 1 also described the aims of this work, the key issues addressed in this thesis and the structure of this thesis.

- Chapter 2 analysed the research problems, i.e., poor performance, vulnerability, limited scalability, user restrictions, unsatisfactory system openness, and lack of incomplete process support, in detail. Based on the problems analysis, it was argued that most, if not all, of these problems are caused due to the mismatch between workflow’s inherent nature, i.e., distributed, and system design, i.e., centralised management. After reviewing the major related work, a claim was presented that workflow’s increasingly distributed nature needs to be reflected much more naturally by the p2p decentralised and collaborative framework. This is the philosophy of this research.
• Chapter 3 combined workflow technology and the p2p computing model to propose the unique system design of SwinDeW: a p2p-based, decentralised workflow management system. In particular, the architectural layers of SwinDeW as well as the major services in SwinDeW were introduced. The structure and organisation of the elementary entity—the peer—were given. In addition, based on the system design, the decentralised peer management service and the centralised system initialisation were also discussed.

• Chapter 4 described the process definition service of SwinDeW, which provides support for build-time workflow functions. In particular, a graph-based process representation approach was presented, which identifies and describes the key elements of workflow processes with a graphical user interface. A conceptual task notation was also presented which represents each individual task within the context of a workflow process. Moreover, a novel process data storage approach was designed, which divides process definition into partitions and distributes individual partitions to relevant peers, thus enabling process definition to be stored in a p2p decentralised environment.

• Chapter 5 described the process enactment service of SwinDeW, which provides support for run-time workflow functions, and the monitoring and administration service, which provides the administrative capability. The mechanisms for decentralised process instantiation, decentralised instance execution and decentralised process monitoring and administration based on a message-passing model were discussed. Three selection policies were described for dynamic work allocation. Basic operations and message types and format were also introduced. In addition, automatic handling for “unavailable peer” exceptions was discussed to increase system reliability.

• Chapter 6 extended the SwinDeW system for incomplete process support without significant impact on the existing design. The causes of incomplete processes were identified. The critical issues and challenges of extending
SwinDeW for incomplete process support were also discussed. Based on the original SwinDeW system, a multi-level process modelling and execution approach was proposed, which specifies workflow processes incrementally in a hierarchical, stepwise manner. In this paradigm, incomplete factors are able to be determined on-the-fly at run-time.

- Chapter 7 studied two cases to illustrate the key ideas proposed in this thesis. The first one was a traditional, complete process of a student registration service. The second one was a non-traditional, incomplete process of research project management. The studies of these two cases showed that the concepts of SwinDeW are practically feasible to support real-world applications.

- Chapter 8 described a prototype of SwinDeW (designed in Chapters 3, 4, 5 and 6), which was implemented for demonstration and proof-of-concept purposes. This prototype is based on Sun Microsystems JXTA underlying infrastructure to utilise its open-source building components. The formal process definition language used in this prototype is WfMC XPDL for the purpose of interoperability. The implementation of this prototype showed that the concepts of SwinDeW are technically feasible.

- Chapter 9 gave an in-depth discussion of the pros and cons of the proposed approach. The advantages of SwinDeW were summarised. At the same time, the tradeoffs of SwinDeW were also pointed out. Based on this discussion, suitable application domains of SwinDeW are analysed.

## 10.2 Contributions of this thesis

The significance of this research is that it fundamentally addresses some of the unsolved problems in the workflow area, i.e., bad performance, vulnerability to failures, poor scalability, user restrictions, unsatisfactory system openness, and lack of incomplete process support. Based on p2p computing technology, this research
investigates a new framework and process coordination technologies for workflow systems comprehensively, which can be regarded as a paradigm shift. This new framework and corresponding coordination technologies exploits the features offered by p2p computing technology naturally in order to reflect workflow’s inherently distributed nature better. This research contributes a lot to the challenging research area of p2p-based workflow which opens new ground in workflow, and process support area in general. Moreover, this research also delivers useful results to the emerging research area of the service-oriented workflow. The major outcomes of this research, namely, the genuinely decentralised and open workflow framework, the decentralised data storage support, the decentralised process enactment environment and the decentralised incomplete process support, provide a better solution to the existing problems. Therefore, this research illustrates cutting-edge technologies for unique workflow system support offering the critical features which are missing in most conventional systems. This adds new, advanced knowledge to the area with which workflow can be deployed to much wider application scenarios in various kinds of organisations. In particular, the major contributions of this thesis are:

- **The identification of the causes of the existing problems in conventional workflow management systems.**
  This thesis has shown that most of the existing problems in conventional workflow management systems are ultimately caused by the mismatch between application nature and supporting technology. Based on this finding, it has been advocated that peer-to-peer computing technology can serve as a suitable underlying infrastructure to support workflow applications better. This finding can also serve as the philosophy for future research and practice on peer-to-peer based decentralised workflow.

- **An innovative system framework and design for peer-to-peer based decentralised workflow management systems.**
  Using peer-to-peer computing technology to support workflow is considered a paradigm shift. However, the system design of the few existing so-called
peer-to-peer based workflow approaches is normally superficial and conceptual, or even not genuinely p2p-based. The proposed four-layer system architecture, as well as the detailed design for each layer, have contributed a complete, concrete, and fully decentralised system design for p2p-based workflow applications. The deployment of the systems built on the basis of this design is practically and technically feasible.

- **The innovative design of decentralised process data storage.**
For the few existing peer-to-peer based workflow approaches, aspects of data storage are either ignored or addressed unsatisfactorily (e.g., through the use of centralised data repositories). This thesis has proposed a unique data storage approach, “know what you should know”, to address the data storage problems in decentralised workflow systems. By partitioning process definition and distributing data properly, workflow participants are allowed to access the essential process information without the assistance of centralised data sources. Note that this approach is independent of process definition language and physical data format. That means this approach can be deployed easily in any p2p-based workflow systems.

- **Dynamic work allocation and the selection policies.**
This thesis has adopted the dynamic allocation of work items, which assigns work items to workflow participants dynamically at run-time. To enable this allocation, three allocation policies have been introduced which have different characteristics and can be applied to the system under different circumstances. The direct advantages of this allocation approach are to enhance system dynamism and flexibility, improve system performance, and achieve better user satisfaction.

- **The innovative design of decentralised process enactment.**
Process enactment coordination plays a key role in any workflow management system. Based on the system design and decentralised process data storage, this thesis has presented innovative coordination support for
decentralised process enactment. Process instantiation and instance execution involve multiple decentralised workflow participants coordinating with one another. This coordination is achieved through direct message passing amongst the participants. Wherever a coordination decision needs to be made (e.g., assign some work or route some work), individual participants make the decision according to either the knowledge or the negotiation results. By these means, the workflow is executed step-by-step from one participant to another according to the pre-defined rules. Moreover, decentralised monitoring and administration are also accomplished.

• The innovative design of a multi-level, hierarchical process modelling and execution approach which is able to support incomplete processes. This research so far has revealed the necessity of extending SwinDeW for incomplete process support without a redesign. The traditional “Hierarchical Task Analysis” has been used in SwinDeW to build a hierarchical process modelling and execution approach. This approach models and executes workflow processes incrementally, thereby enabling the SwinDeW decentralised workflow system to be deployable in non-traditional workflow application domains where processes can only be specified partially at build-time. The most distinguishing point of this approach is to model the run-time process specification work as essential steps towards the process objectives and fulfil run-time process specification by enacting these steps.

• The JXTA-based SwinDeW prototype. The SwinDeW prototype based on JXTA has been implemented for demonstration and proof-of-concept purposes. The current version has demonstrated the feasibility of using SwinDeW to support both complete and incomplete processes. This prototype serves a good basis for future extension, evaluation and improvement of the approaches proposed in this thesis.
10.3 Future work

In the future, further investigation into peer-to-peer based decentralised workflow will be carried out. Future research includes workflow interoperability and system/tool support, improvement of the prototype and the wider evaluation of performance, handling of dynamic changes and generic exceptions, enhancement of process monitoring and administration, security of system, run-time verification of workflow instances, and so on.

The research to date has focused on the four elements identified in the WfMC’s workflow reference model, i.e., client application, process definition, workflow engine, and administration and monitoring. However, workflow interoperability and invoked applications have been omitted temporarily. The aspects of integrating SwinDeW with a wide range of distributed IT tools and integrating different data for workflow applications, and the issues of interoperability between SwinDeW and other workflow systems, either centralised or decentralised, will be addressed in the near feature.

As indicated in Chapter 8, at this time, the SwinDeW prototype is for demonstration and proof-of-concept only. After continual extension and improvement, especially with the great enhancement of monitoring and administration facility, some real-world applications will be developed on top of this prototype as experiments in order to collect more quantitative results. Experimental simulation may be designed for purposes like performance evaluation. A more comprehensive comparison of different workflow systems, either centralised or decentralised will be conducted.

At this moment, SwinDeW has very limited ability to support dynamic changes and more generic exceptions, which makes it weak to deal with ever-changing workflow environments in the real world. Therefore, the next logical step is to explore methods and techniques in support of various kinds of workflow changes and exceptions in the decentralised environment. Especially, the current coordination mechanisms should be extended to facilitate detection, diagnosis and
handling of dynamic changes and exceptions. Moreover, the management and administration in SwinDeW will also be enhanced by developing special peers with ability to have a global view of the system, as indicated in Section 9.2.

Finally, some other important issues such as more complicated modelling of incomplete processes, organisational management, run-time verification of workflow instances in the decentralised workflow system, security of p2p-based workflow are temporarily ignored to simplify the problem. In order to have a practical workflow solution, collaborative research will be carried out to investigate all of the above issues.
Bibliography


[BDSN02] B. Benatallah, M. Dumas, Q. Z. Sheng and A. H. Ngu, Declarative Composition and Peer-to-Peer Provisioning of Dynamic Web Services, Proc. of the 18th International Conference on Data Engineering (ICDE’02), 297-308, San Jose, US, 26 February - 1 March 2002


http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm


[Yan02a] Y. Yang, Tool Interfacing Mechanisms for Programming-for-the-large and Programming-for-the-small, in Proc. of the 9th Asia Pacific Software Engineering Conference (APSEC’02), 359-365, Gold Coast, Australia, Dec. 2002


[YYR03b] J. Yan, Y. Yang and G. K. Raikundalia, Enacting Business Processes in a Decentralised Environment with p2p-based Workflow Support, in Proc. of the 4th International Conference on Web-Age Information Management (WAIM’03), Lecturer Notes in Computer Science, vol 2762, 290-297, Chengdu, China, Aug. 2003


The author’s publications


