Distributed collaborative context-aware content-centric workflow management for mobile devices

Distributed collaborative context-aware content-centric workflow management for mobile devices

Anna Kocurova

A thesis submitted in partial fulfilment of the
requirements of the University of West London
for the degree of Doctor of Philosophy

September 2013

Acknowledgements

From my point of view, the past four years have been a pilgrimage and I would like to thank the following people for all their support during this journey.

First, I would like to express my deep gratitude to my research supervisors, Dr Samia Oussena, Prof Peter Komisarczuk, and Prof Tony Clark. My special thanks to Samia, for the opportunity, for her valuable and patient guidance and for useful critiques of this research work. Many thanks to Peter, for his suggestions, advices and positive attitude. Many thanks to Tony, for being my external supervisor and his insightful comments and feedbacks.

I would like to thank to my fellow PhD mates, Dean Kramer, Malte Ressin, Alison Wiles and Sujan Shresta, for great office co-habitation and all those unforgettable moments. Special thanks to Dean and Malte, for sharing so many ups and downs of the PhD life with me, for being so encouraging and friendly. I am particularly grateful to Dean for his hard work on our Context Engine. I would also like to thank Tina South, for her regular visits to our office and for being so supportive and caring.

I wish to express my thanks to all members of the School of Computing and Technology at University of West London, for their help and support in completing this work. I would like to thank Maria Pennells for her indespensable support and help during the last four years, and arranging the viva. I would also like to thank Prof Thomas Roth-Berghofer for agreeing to be an internal examiner at my viva.

I would like to thank Urmi Chana and Mahendra Mahey for their continual support and their tips for improving my English.

Finally, I would like to thank to my husband, Marian, and my daughters, Veronika and Viktoria, for all their patience, love and encouragment. Many thanks to my parents, family and all friends for being there when I needed you.

'If you think you are too small to make a difference,
try sleeping with a mosquito.'

Dalai Lama XIV

Abstract

Ubiquitous mobile devices have become a necessity in today's society, opening new opportunities for interaction and collaboration between geographically distributed people. With the increased use of mobile phones, people can collaborate while on the move. Collaborators expect technologies that would enhance their teamwork and respond to their individual needs.

Workflow is a widely used technology that supports collaboration and can be adapted for a variety of collaborative scenarios. Although the originally computer-based workflow technology has expanded also on mobile devices, there are still research challenges in the development of user-focused device-oriented collaborative workflows.

As opposed to desktop computers, mobile devices provide a different, more personalised user experience and are carried by their owners everywhere. Mobile devices can capture user context and behave as digitalised user complements. By integrating context awareness into the workflow technology, workflow decisions can be based on local, context information and therefore, be more adapted to individual collaborators' circumstances and expectations. Knowing the current context of collaborators and their mobile devices is useful, especially in mobile peer-to-peer collaboration where the workflow process execution can be driven by devices according to the situation.

In mobile collaboration, team workers share pictures, videos, or other content. Monitoring and exchanging the information on the current state of the content processed on devices can enhance the overall workflow execution. As mobile devices in peer-to-peer collaboration are not aware of a global workflow state, the content state information can be used to communicate progress among collaborators. However, there is still a lack of integrating content lifecycles in process-oriented workflows.

The aim of this research was therefore to investigate how workflow technology can be adapted for mobile peer-to-peer collaboration, in particular, how the level of context awareness in mobile collaborative workflows

can be increased and how the extra content lifecycle management support can be integrated.

The collaborative workflow technology has been adapted for mobile peer-to-peer collaboration by integrating context and content awareness. In the first place, a workflow-specific context management approach has been developed that allows defining workflow-specific context models and supports the integration of context models with collaborative workflows. Workflow process has been adapted to make decisions based on context information. Secondly, extra content management support has been added to the workflow technology. A representation for content lifecycles has been designed, and content lifecycles have been integrated with the workflow process.

In this thesis, the MobWEL workflow approach is introduced. The MobWEL workflow approach allows defining, managing and executing mobile context-aware content-centric workflows. MobWEL is a workflow execution language that extends BPEL, using constructs from existing workflow approaches, Context4BPEL and BPELlight, and adopting elements from the BALSA workflow model. The MobWEL workflow management approach is a technology-based solution that has been designed to provide workflow management support to a specific class of mobile applications.

Contents

List of Figures

List of Algorithms

Part I

Foundations

Chapter 1

Introduction

Contents

1.1 Motivation

1.1.1 The Vision of Mobile Collaboration

In the era of ubiquitous computing, the view of mobile phones as devices to make and receive telephone calls has been replaced by a vision of smartphones. By combining telephony and computing capabilities, smartphones could offer users a wider variety of services, Internet connection and applications. People have been accustomed to using the mobile computing capabilities in everyday life. Mobile phones, as the always-on and always-carried devices capable of creating and distributing various content, taking and uploading pictures and capturing events (Fling, 2009), have become ubiquitous, opening new possibilities for collaboration, communication and sharing ideas in real-time. Using smartphones allows people to remain productive regardless of their geographical locations and work while on the move. However, with the spread of using mobile technologies, collaborators expect to have tools to collaborate, share and manage content as time and situation require (Erickson et al., 2009). Mobile devices are used in various collaborative scenarios including

collaboration in manufacturing, health care and education. Improved and instant communication enables interactions between suppliers and customers, nurses and patients, or tutors and students. A broad range of mobile collaborative applications such as Dropbox, or ShowDocument have been developed to support collaboration, file management, social networking, work organisation or scheduling.

Advances in mobile collaborative solutions have enriched user experience, but also have resulted in a new set of user demands. The raising user expectations for mobile collaboration, apart from just putting workload onto smartphones, include the demand for an intelligent mobile collaborative environment which responds to people′s needs. A smartphone is a device physically mobile and carried by its owner constantly. Therefore, the device and the user have a similar perception of the environment which makes possible for the mobile device to be a user's digital complement. The fact has motivated users to raise the demand for intelligent mobile systems being centred on an individual user.

Bringing intelligence into everyday life and environments is addressed by the ambient intelligence paradigm. The ambient intelligence vision is characterised by technologies which are embedded, context-aware, personalised, adaptive and anticipatory (Denning, 2001). In other words, the ambient intelligence paradigm refers to *a*) devices and technologies integrated into the environment which is sensitive to humans' presence; *b*) people′ interaction with devices is personalised, easy and natural; and *c*) devices can recognise the situational context and are tailored to user needs. The goal of ambient intelligence is to seamlessly weave intelligent technologies into everyday lives and its ongoing challenges include awareness of user preferences and needs in the systems (Cook et al., 2009).

There is no doubt that the technologies will not come from a single research field. Building a fully intelligent environment requires the development of well integrable, improvable and evolutive individual technologies that address certain aspects and requirements of it. The individual technologies need to envisage today's raising user expectations, add value to real world and contribute to the existing base of knowledge.

One of the individual technologies is collaborative workflow, a widely accepted technology which offers rich support for multi-party collaboration, coordination of distributed teamwork and content sharing. The work presented in this thesis contributes to the long-term vision of an intelligent mobile collaborative environment by investigating and adapting the collaborative workflow technology for mobile collaboration, as described next.

1.1.2 Challenges of Mobile Collaborative Workflows

Collaboration originated in the past when work was divided into separate tasks assigned to single workers. Workers, each specialising only on a particular aspect of service or goods production, had to collaborate together to reach common goals. During the workdays, people followed certain work processes and patterns to accomplish their goals. The last decades of the 20th century, a period of an information technology boom, opened new ways of collaboration. The work processes, previously managed only by people, have been handled by information technologies which could partially support, control or automate the work. New concepts such as workflow, an abstraction of phenomena of work in the real world, and workflow management systems (WfMS), generic software packages designed for workflow management and execution, have been introduced.

The impact made by the introduction of the workflow management technology is present in many areas such as transportation management, enterprise resource planning or customer relationship management. Better efficiency, process control, worker productivity and process improvement can be achieved by the deployment of workflow systems (Yan et al., 2006). By using the technology, the logistic facilities for processes automation have been provided by workflow management systems and the complexity of work processes has been reduced as the tasks have been allocated to right people at right time and in right order.

Workflow management systems belong to a class of generic software and can be used for a number of different purposes. However, in the workflow management systems the processing logic is separated from the application logic. This means that actual tasks are not performed by the systems and other software applications are needed. Therefore, the technology has been often customised, modified and adapted for various collaboration needs. As a result, a variety of workflow models, languages, modelling concepts and software packages has been developed.

With the arrival of mobile computing on the scene, mobile devices have been integrated into workflow scenarios and the workflow management technology has been adapted to those scenarios. Consider an example where a group of collaborators work on a project in which each participant is responsible for certain tasks. The tasks are in a logical order and collaborators have to follow a particular work pattern to finish the project. By deploying collaborative workflow management technologies, users can rely on a software system which would automate their work process and task scheduling. It would be beneficial if collaborators could perform little tasks such as approving or declining proposals at any time without any restrictions. Smartphones are ideal tools to achieve that. By employing smartphones to complete some workflow tasks, collaboration can become more efficient.

Some workflow solutions for smart devices have been developed, however, the approaches relied on workflow management systems deployed on servers and provide only light-weight workflow process support on mobile devices (Battista et al., 2008; Pryss et al., 2011). Using the approaches that are based on the server-client model is almost impractical for small-scaled workflows in various situations in which people could collaborate only by using mobile devices. Typical situations where purely mobile collaboration might add value are meetings between business partners who share private information, or teamwork in small companies without own server. Moreover, in the centralised architectural style, control and management of workflows is driven by a central unit with mobile devices behaving only as thin clients. The central management unit makes all important decisions and drives the whole workflow execution without considering the current situation of users and context the devices reside in. The need for a mobile device-centric workflow process, and systems that manage such workflow processes in a completely distributed manner has been recognised (Pajunen & Chande, 2007). Mobile collaboration in a distributed manner, also labeled as mobile peer-to-peer collaboration, involves a set of mobile devices connected through wireless links. Moreover, in peer-to-peer workflow management, data and content can be distributed by direct exchange and all decisions are made by mobile devices. Therefore, the decisions can be adapted to the current user and device circumstances and based on local, context information.

The evolution of solutions for peer-to-peer mobile collaboration and workflow management can contribute to the vision of building an intelligent collaborative environment. However, the mobility of collaborators imposes constraints on tasks allocation, coordination management or results marshalling (Hackmann et al., 2006a). Smanchat et al. (2008) pointed out that to utilise workflows in ubiquitous environments, adaptability and context-awareness are the features that should be included in the workflow mechanism. To increase the level of context awareness in a distributed mobile workflow management system requires a sophisticated software infrastructure to support *a*) gathering, storing, trading, and using context knowledge; *b*) dealing with uncertain and unpredictable changes in human actions (Sen, 2008).

In addition, peer-to-peer mobile collaboration often involves content manipulation, for example, when collaborators review or approve proposals or pictures. Mobile content such as picture, document or video/audio file is usually user-generated or adapted for use on mobile devices. Two workflow approaches for content processing are *a*) activity-centric; and *b*) object-centric (Nurcan, 2008). In the object-centric approach, an object is placed in the center of the workflow process rather than people or process itself. Each workflow step alters the object. This approach

is usually used in content management systems such as Alfresco[1]. In contrast, in the activity-centric or process-centric approach, the workflow process is described as a sequence of activities and the control-flow facet of workflow is emphasised. BPEL, BPMN and YAWL are examples that represent this approach. This approach is useful for representing the functional view of workflows. Although the flow of data and objects is indicated, data and objects are handled only as second-class citizens in the activity-centric approaches. The duality of information-centric and activity-centric workflow models, and the strengths and weaknesses of the two modelling approaches have been illustrated in the work of Kumaran et al. (2008).

There are several benefits to increasing content awareness in process-oriented workflows and integrating advanced content management capabilities in peer-to-peer mobile workflow management. Firstly, knowing the content state can enable content synchronisation over a number of devices, especially in peer-to-peer collaboration. There are basically two ways to share the information about content object that is processed on a particular device. In the first way, a set of all content-related data would be send to all interested devices at any time when content has been updated or modified. In this case, all devices would need to process the whole set of information and if required, undertake any consequential actions. This approach is inefficient as all devices would need to process and evaluate the incoming set of data. Moreover, the broadcasting would be irregular and chaotic, requiring an excessive amount of processing on all involved devices. On the other hand, the more organised and simpler way to share the information among the devices is by more controlled and organised content management on each device. Adding extra content management functionalities on each device enables the set of content-related information to be processed directly on the device, using content states as milestones in content behaviour. Sending the information only about certain content states can keep the workflow execution more consistent and synchronised. The number of messages would be significantly reduced, the interaction points could be better indicated and the shared messages would be more expressive. Secondly, there can be relationships and dependencies between various content items. For example, the proposal can have more sections edited by co-workers on their devices. Receiving the regular information about the sections would enable to process the proposal accordingly.

So far, however, too little attention has been paid to content processed in process-centric workflows. Many workflow approaches are organised around process-centric and activity-centric workflow models which have relatively limited focus on artifacts processed in workflows. Entities such as content or business artifacts are integrated

[1] http://www.alfresco.com/

in workflows as an input or output of an activity and the effects of how performed activities influence an entity's behaviour are not visible (Bhattacharya et al., 2009). The field of artifact-centric workflows is still in infancy (Hull, 2008) and object-awareness in process-centric workflows is still very limited (Künzle & Reichert, 2011). Moreover, there has been lack of consideration how the approaches could operate in a peer-to-peer mobile setting and the research to date has tended to focus on the lifecycle of the artifact without considering its adaptation to context changes.

The work presented in this thesis explores how the collaborative workflow mamanagement technology can be adapted to mobile platforms in order to meet interaction needs of collaborators, and how content behaviour and context awareness can be integrated in the adapted workflow management. Workflows need to be well defined because the success of the workflow management system is based on the quality of the workflow models put into it (Aalst & Hee, 2004). This thesis tackles the problem associated with the representation of context information available on a single mobile device, and using context information to shape peer-to-peer workflow execution. Context describes the current situation of a user, a device or an environment regarding a specific purpose. By expressing context explicitly for each workflow, each mobile device can monitor, acquire and process only the required context information. By adapting workflow management to consume the context information and make decision based on it, the workflow execution can become more dynamic and responsive to individual collaborator's needs and current situation.

1.2 Research Aim and Objectives

The research aim is to adapt the collaborative workflow technology for peer-to-peer mobile collaboration by targeting the challenges described in the previous section. The intention includes making workflows context-aware so the execution of workflows can be based on local, context information and therefore adapted to collaborator's needs and circumstances. Adding extra support for content lifecycle and management will increase content awareness in workflows and make content a first-class citizen in the activity-based workflows.

The main objectives addressed in this work are:

1. To design a workflow model that defines a certain class of mobile distributed context-aware content-centric workflow processes at an abstract level and in a machine-interpretable form. The characteristics of the certain class of workflows are as follows:

    - Workflows are activity-oriented and collaborative. Workflow activities are

performed by a number of roles and actors.

- Workflow tasks are accomplished by using mobile devices such as smartphones or tablets, and workflows are executed solely on mobile devices.
- Current situation and needs of stakeholders can be captured on mobile devices and used to trigger workflow tasks or influence workflow decisions.
- There is one or more artifacts such as pictures or documents processed in the workflow. Different artifact evolution states are monitored and used to communicate workflow progress among collaborators.

2. To design a generic workflow management software architecture that is capable of managing and executing such workflow processes and operates in a peer-to-peer manner on mobile devices.

3. To implement a prototype of the generic workflow management system on a mobile platform.

4. To analyse two usage scenarios and construct a concrete mobile distributed context-aware content-centric workflow process for a usage scenario.

5. To demonstrate that the workflow instances of the constructed workflow process are manageable and executable.

1.3 Research Questions and Focus

Based on the research aim outlined in the previous section, the investigation leads to the following research question and subquestions:

Research Question

- **How can distributed mobile context-aware content-centric workflows be represented, managed and executed?**

  Chapters 4, 5 and 6 are dedicated to this question and present a mobile workflow execution language (MobWEL). MobWEL is a workflow language that is context-aware, integrates content behaviour and supports mobile peer-to-peer interaction. Chapter 4 provides the definition and syntax of the workflow language. In Chapter 5, the MobWEL semantics is defined. MobWEL workflow management and execution is elaborated in Chapter 6.

Research Subquestions

1. **How can context-awareness be integrated and supported within the workflow process?**

   A workflow-specific context management approach has been developed as a part of the MobWEL workflow approach. A metamodel that defines context models is introduced in Section 4.5. Processing of context models is elaborated in Sections 6.3 and 6.4.

2. **How can content behaviour be supported in the workflow process?**

   Content lifecycles are expressed in the same way as the workflow process is described. A metamodel for the definition of content lifecycles is presented in Section 4.6. Content management support is described in Sections 6.5.

1.4 Contributions

The thesis contributes to the mobile collaborative workflow domain in the following ways:

- **Mobile Workflow Execution Language (MobWEL) Metamodel:** A mobile workflow execution language called MobWEL has been designed and its metamodel is introduced in the thesis. The MobWEL metamodel comprises four parts (Workflow-specific context definition, Context-aware content lifecycle, Group identification, and Process Control Flow). MobWEL extends $BPEL$, integrates $BPEL^{light}$'s interaction model, uses the $Context4BPEL$'s context-aware approach and adapts dimensions defined in BALSA, an artifact-centric workflow model.

- **MobWEL Syntax and Semantics:** MobWEL syntax and formalised semantics are described in this thesis. Its syntax is expressed in XML because of $BPEL$, $BPEL^{light}$ and $Context4BPEL$ are XML-based workflow languages. Concrete workflow definitions are described in XML and their behaviour can be described in a formal way by using MobWEL semantics.

- **MobWEL Workflow Management System Specifications:** The logical and run-time architecture of the MobWEL workflow management system that extends a BPEL engine are described in this work. The system is designed to carry out workflows which are described in MobWEL. Context Provider, Context Manager, Content Manager, MobWEL Engine, and Peer-to-peer Interaction Manager are main MobWEL workflow management system components.

Context Provider monitors, acquired, processes and disseminates context information. Context Manager manages and routes relevant context information to right internal components. Content Manager provides advanced content management functionalities. The MobWEL engine manages and instantiates MobWEL workflows, and executes its instances. Peer-to-peer Interaction Manager handles interaction among collaborators.

- **Workflow-Specific Context Management Approach:** A workflow-specific context management approach has been designed. The approach is introduced in the thesis and covers context modelling, context acquisition, context providing, context management and context consumption on a mobile device. Separation of the context acquisition logic from context adaptation logic is supported in this approach, meaning that the Context Provider component can be used either internally as a part of the MobWEL workflow management system, or externally as a context providing service. As an external component, the service can be used by multiple context aware applications running on the same mobile device.

- **Context-Aware Content Management Approach:** Mobile content objects processed in MobWEL workflows are considered as significant workflow artifacts, and their lifecycles are explicitly expressed. Content states are monitored and used to communicate progress to fellow collaborators.

- **Software Prototypes:** Two software prototypes have been built on the Android platform[1] and are described in this work. ContextEngine, a prototype of the context provider, monitors, acquires, processes and distributes workflow-specific context information. ContextEngine provides context provisioning services to context-aware mobile applications running on the same mobile device. This engine has been developed in collaboration with Dean Kramer, who has used it for a different research project. CAWEFA is a prototype of a mobile workflow management system. CAWEFA extends the Sliver BPEL engine, and has been developed to carry out MobWEL workflows. Both software prototypes will be available to research community as open source software.

Generally, the designed technology intends to enhance mobile peer-to-peer collaboration. This thesis presents the MobWEL workflow approach that contains all necessary information to define a certain class of workflows. The MobWEL workflow approach adds a value in the real world as follows. Firstly, it can be used by workflow designers who will be able to model and define the concrete workflow cases

[1] http://developer.android.com

for mobile applications. Secondly, the workflow management software is designed in a generic way and can provide workflow management support to multiple mobile applications running on the same mobile device. Finally, the abstracted description of the MobWEL workflow management system can be used for the implementation of the software on any mobile platform.

1.5 Approach

1.5.1 Research Methodology

Design Science Research by definition changes the state-of-the-world through the introduction of novel artefacts (Vaishnavi & Kuechler, 2004) and is particularly relevant for investigation and understanding innovation in mobile computing (Love, 2009).

By considering Hevner's Design-Science research guidelines (Hevner et al., 2004) and the design cycle of Vaishnavi & Kuechler (2004), this research has engaged the design science research approach which comprises four phases as follows (Figure 1.1):

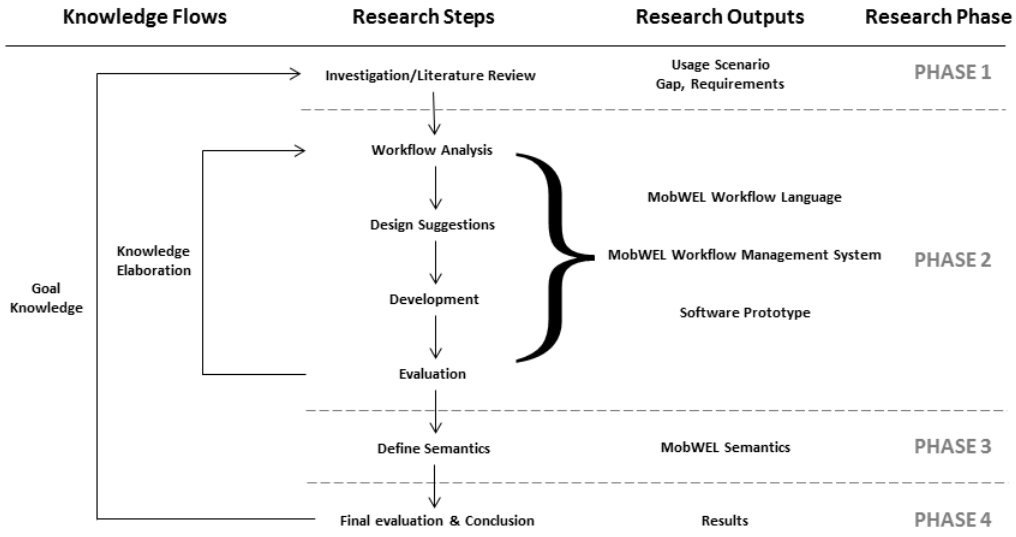

Figure 1.1: Research Methodology

Research Phase 1: During this phase, the research area is explored and the necessary insight is gained, research boundaries and requirements are defined, the gap and problem are identified. A usage scenario that represents the targeted class of mobile collaborative workflow scenarios and applications is created.

Research Phase 2: Epistemologically, the *'knowing through making'* belief is implied. Knowledge and artifacts are elaborated through an iterative constructive process which stops when the research results are acceptable and satisfactory. The process is based on the following steps:

1. *Qualitative workflow analysis:* By performing workflow process analysis, opportunities for workflow adaptation are identified.
2. *Design suggestions:* As a result of the qualitative workflow process analysis, new workflow constructs are formed, new data structures are examined, and a set of requirements and expectations is suggested.
3. *Development phase:* Development of research artifacts is based on experiments.
4. *Evaluation iterations:* Micro-evaluation accompanies each design decision.

Research Phase 3: Formalisation of the knowledge obtained in Phase 2 and developing the MobWEL semantics.

Research Phase 4: Final Evaluation, Validation and Verification of the research artifacts.

1.5.2 Development Methodology

Design Research produces general outputs in a form of constructs, models, methods, or instantiations (March & Smith, 1995). Development of the research artifacts requires various techniques that extend and enhance the design research paradigm. The following techniques have been implied.

A usage scenario presented in Chapter 2 has been created and used to navigate this research. *Scenario-based design* is a family of techniques in which the narrative descriptions of the system use are described early in the development process (Rosson & Carroll, 2002). The scenario-based design approach has been employed in various ways to guide the development of the research artifacts :

- **Requirements analysis:** The scenario has provided a rich picture on participating stakeholders, and their work pattern, and outlined the need for a technology that would enhance their collaboration and teamwork. Scenario analysis has been used to gather requirements.
- **Design and research rationale:** The domain concepts have been explored and constructed with respect to the usage scenario.

- **Envisioning:** The scenario has been used to visualise and simulate the use of a system.
- **Software design:** The identification of key domain objects and the decisions about the design direction have been based on the scenario analysis.
- **Implementation:** The prototype implementation has been driven by the usage scenario in order to keep certain boundaries of the phase.
- **Evaluation and testing:** The scenario has been used as a skeleton for validation of the MobWEL approach.

These steps have been relevant to the development of the workflow language and the software.

MobWEL Language Definition

Metamodelling is the way to design and integrate semantically rich languages in a unified way (Clark et al., 2008). A metamodelling approach is used to define all constructs of the MobWEL workflow processes, the relationships between constructs and rules that state how the constructs can be combined to create models.

The constructed metamodel is then mapped into an XML Schema and MobWEL becomes an XML-based language. MobWEL workflows are described in XML documents which conform to the XML schema.

Software Development

MobWEL workflows are carried out by appropriate software. *Prototyping*, a software engineering developmental technique based on iterative and incremental building of software prototypes, has used to develop the software and explore design alternatives, test theories and suggestions. By using the software prototyping technique, the suggestions are implemented. Working software is a proof of concept that the theory is functional and workflows are executable. The construction of prototypes is driven by the usage scenario.

A *modularisation* approach and *component-based software engineering* practice are used in the development of the mobile workflow management system. The workflow management system is composed of loosely coupled independent components. Each component groups semantically related data and functions relevant to one management concern. Using this component-based approach enables the exploration and evolution of each concern and component on its own. In addition, the system needs to detect, consume, and react to context events. To cope with context events, the system architecture is based on the *event-driven* architecture pattern.

1.5.3 Validation Approach

Validation and verification are important phases of the research process to determine whether the designed research artifacts fit the intended purpose. In design research, the evaluation of the artifacts requires formulation of criteria for success (Blessing & Chakrabarti, 2009). Criteria are used to plan the appropriate evaluation, assess the evaluation results and judge the outcome of the research against the research goals and objectives.

This design science research produces an artifact in the form of the MobWEL workflow language. The hypothesis is that the MobWEL workflow language is a suitable technology for the definition and representation of mobile distributed context-aware content-centric workflows. To validate this hypothesis, the following criteria for success have been defined:

- A certain class of mobile distributed context-aware content-centric workflows can be defined by using the MobWEL workflow language.
- The instances of MobWEL workflows are manageable and executable.

To achieve the validation goal, the following steps have been undertaken:

1. Design and development of the MobWEL workflow management system:
   - A logical and run-time architecture of the system have been designed.
   - A prototype of the system has been developed on the Android platform.
2. Once the MobWEL workflow language and management system were constructed, the artifacts are evaluated according to the defined criteria. The final evaluation based on experimentation, one of the most fundamental methods for validation in the software engineering area (Wohlin et al., 2012), was conducted:
   - A MobWEL workflow definition for the usage scenario has been constructed.
   - The MobWEL semantics has been used to describe the expected behaviour of individual workflow instances.
   - The MobWEL workflow definition has been deployed to the MobWEL workflow management system.
   - The MobWEL workflow was instantiated and the behaviour of running workflow instances was monitored.

- Comparison of the expected with the actual behaviour of workflow instances. Quantitative data was collected and analysed appropriately to produce relevant answers.

This concludes the description of the methodologies used in this research. The next section outlines the structure of this thesis.

1.6 Scope and Structure of Thesis

The goal of this thesis is to present a first milestone in the definition, management and execution of MobWEL workflows designed for peer-to-peer mobile collaboration. The focus is put on the description and integration of two main aspects: the workflow-specific context management process and context-aware content management support. Special interest is placed on specifying the anatomy, syntax and semantics of the MobWEL workflow language.

However, to complete this work within three-and-a-half year time frame, this work does not include workflow partitioning, run-time optimisation of the workflow processes and fault handling. It is not extended to deal with tasks allocation and optimised content sharing in peer-to-peer interactions. Also this work does not cover the recent concepts of the adaptation or composition of the workflow definition at run-time. In addition, an overall logical architecture of the workflow management software package is highlighted, however, the system is complex and thereby only its major components are fully described.

Generally, there are five categories of workflow management activities: workflow design, workflow modelling, workflow execution, workflow monitoring, and workflow optimisation. It is necessary to emphasise that only first three categories are addressed in this work.

The reader should be familiar with the following concepts and technologies: BPEL, UML, Java, XML schema and XML.

This thesis comprises four parts: **Foundations, Contribution, Validation and Conclusion**.

Part I: Foundations

Chapter 1: Introduction - The current chapter motivated this work, outlining the vision of futuristic mobile collaboration and challenges in the collaborative workflow management technology. Research aims, research questions, contribution and re-

search approach were described.

Chapter 2: Background and Concepts - In this chapter a scenario of mobile peer-to-peer collaboration is presented. The scenario describes a particular workflow case which is used to illustrate the technology produced in this work. After that, a background to the topic is given and the basic terminology needed to understand the work in this thesis is defined.

Chapter 3: Related Work - This chapter provides the literature review. The review covers the various aspects of context modelling and management, content management and workflow adaptation. The support for the management and execution of context-aware content-centric workflows is also discussed.

Part II: Contribution

This main part presents the contribution of the thesis. This thesis is concerned with designing a workflow model that contains information to define, manage and execute MobWEL workflows. Figure 1.2 shows the components of the workflow model and outlines the structure of Part II. The model is based on the workflow reference model defined by Workflow Management Coalition (Hollingsworth, 1995). At the highest level, two functional areas are characterised: build-time functions and run-time functions. At build-time, workflows are defined. MobWEL workflow specification comprises three parts: Context Definition Model, Content Lifecycles and Adapted Workflow Process Definition which is an extended version of an existing basic workflow definition. The defined workflows are represented in a machine-interpretable form and are deployed to a workflow management software package. At run-time, the software package interprets the defined workflows, instantiate them and manages the execution of running workflow instances. The software package is built from two systems: Context Provider and Mobile Workflow Management System.

The contribution of the thesis is decomposed into three chapters as follows:

Chapter 4: MobWEL Definition and Syntax - The MobWEL workflow anatomy, metamodel and syntax are described here.

Chapter 5: MobWEL Semantics - This chapter defines the semantics of the MobWEL workflow language.

Chapter 6: MobWEL Workflow Management and Execution - In this chapter an

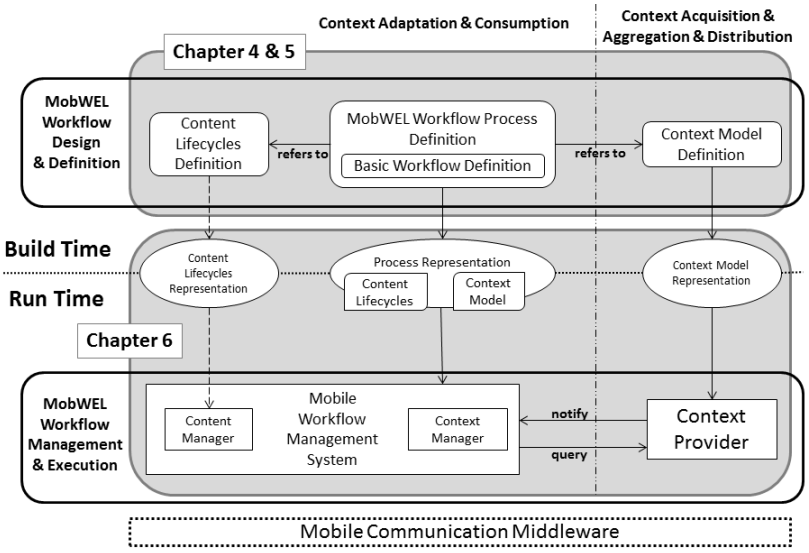

Figure 1.2: Relation between key contributions of this thesis

architecture of the MobWEL workflow management system is presented and the functioning of all system components is elaborated.

Part III: Validation

The following two chapters are dedicated to the validation of the MobWEL workflow management approach:

Chapter 7: From Design to Implementation - This chapter presents the implementation details of software prototypes implemented for the Android platform.

Chapter 8: Evaluation - Experiments have been conducted to validate the designed research artefact. The validation technique and results are described in this chapter.

Part IV: Conclusion

Chapter 8: Conclusion and Future Work - This chapter summarises the thesis and outlines interesting areas of future work.

After the conclusion chapter, appendices are included. **Appendix 1** describes a methodology how to define a MobWEL workflow. **Appendix 2** contains the XML schemas for MobWEL workflow constituents. **Appendix 3** provides details about the definition of MobWEL workflow for the usage scenario. Finally, **Bibliography** is given and **Published Papers** are attached.

Chapter 2

Background and Concepts

Contents

2.1 Introduction

This chapter presents background information for the remainder of the thesis. The objective of this chapter is to provide an introduction to the domain and define the terminology used in this work.

The remainder of this chapter is structured as follows. Firstly, usage scenarios for mobile peer-to-peer collaboration are provided in Section 2.2 in order to give some concrete ideas for the use of the designed technology. After that, the concepts related to several relevant areas are elaborated. Basic concepts and standards related to workflow management, peer-to-peer workflow management and workflow types are explored in Section 2.3. Section 2.4 provides a brief overview of mobile distributed and peer-to-peer systems. Section 2.5 is dedicated to object and content awareness. In Section 2.6, context-awareness and its association to workflow management are explored. Finally, the chapter is summarised in Section 2.7.

2.2 Mobile Peer-to-Peer Collaboration Scenario

The MobWEL approach is tailored specifically to a certain class of workflows and this section describes the type of workflows which are targeted. Two usage scenarios are described to illustrate the use of the MobWEL workflow model.

The characteristics of the targeted workflow type are as follows:

- Workflows are activity-oriented and collaborative. Workflow activities are performed by a number of roles and actors.
- Workflow tasks are accomplished by using mobile devices such as smartphones or tablets, and workflows are executed solely on mobile devices.
- Current situation and needs of stakeholders can be captured on mobile devices and used to trigger workflow tasks or influence workflow decisions.
- There is one or more artifacts such as pictures or documents processed in the workflow. Different artifact evolution states are monitored and used to communicate workflow progress among collaborators.

In the first usage scenario, consider a small business focusing on residential and commercial interior design of houses, offices and shops. In this company, design projects are carried out by a team of ten interior designers. Business success depends on close cooperation within the team and between the team and customers. Designers usually work out in the field and communicate with each other by using their smartphones. With mobile devices, designers can make important decisions right away and share images of design patterns with as little delay as possible.

Designers work on a number of concurrent projects. Although each project is assigned to a particular designer, design decisions are never done by a single person. Jane, as a specialist on private houses, has been assigned to a project to redesign a living room for a private client. The client wishes to optically enlarge the room by repositioning the furniture. Undertaking of the project requires following this work pattern:

1. Designer Jane takes a picture of a new room design by using her smart phone.
2. A simple rating system is used to quickly assess design ideas. Jane adds her own rating to the picture.
3. The picture is sent to her fellow workers. They work out in the field so each of them can review the picture by using their own mobile phone.
4. The reviewer's subjective opinion can be captured by adding a comment.

5. Reviews and comments are sent back to Jane. She finally reassesses her idea according to opinions of other designers.

6. If the idea is good, the picture is sent for final approval to customer.

7. The approved picture is added to Jane's completed work.

Designers do not want to perform all tasks, such as launching camera to take picture, or sending the picture to team workers by looking up their contact details in the contact list, manually. The team follows this work pattern on a regular daily basis, therefore, performing the tasks manually several times a day would be time consuming and inefficient. Therefore, the company expect to have a technology that automates this work pattern and minimalises the number of human input.

In addition, there are other requirements for the technology:

- The company does not have an own server, therefore, the technology has to be supported on smartphones.

- The technology needs to be adapted to individual collaborator's needs. For example, one designer might need to obtain two reviews at a certain point of time, while another designer requires at least 4 reviews to be able to assess whether the proposed design is really good.

- Not all designers are always available to review the picture. For instance, some of them might have a day off, or are busy in a meeting with clients. Thereby, sending the picture to all of them would be inefficient and the picture should be sent only to those team workers who are available to review it.

- Finally, the picture is processed on several devices. Consider a situation that the picture has been sent to all fellow collaborators to be reviewed but the interior designer who sent the picture needs to obtain only two reviews to be able to assess the picture. Therefore, when the picture is assessed, the other collaborators, who have been also asked to review it but have not done so yet, are informed that the picture has been already assessed and there is no more need for their reviews.

The second usage scenario that illustrates the use of the MobWEL workflow approach is described next. Universities often provide students with the opportunity for work placement experience. While on a placement, students closely communicate with their tutors and placement mentors who monitor their attendance and progress. There can be two workflows outlined in such collaboration.

In the first workflow for monitoring attendance, smartphones can be used to capture time when student comes to the placement venue and a message of the arrival is sent to tutor and mentor. When student enters the placement venue, the location data provided by the GPS of the mobile device can be used as a trigger for capturing the time on a daily basis. On the other hand, if student does not enter the placement venue in a certain time range, and student arrival has not been recorded in any other way, tutor and mentor are informed about student absence by their mobile devices.

In the second workflow, mentor creates a day plan of actions and tasks for student. Each task has its start time, duration, goals, learning outcomes and location. Mentor shares the plan with student and tutor. Student can indicate completion of each task, add own notes and task rating by using own smartphone or tablet. In addition, if allowed, student can take pictures by using own smarthphone and enrich task related notes by them. Once the plan achieves a certain state of completeness, the plan outcome is sent to mentor and tutor who assess it and take further actions. Using smartphones in this scenario would enable capturing of important information more accurately, for example, timestamps can be added to tasks or task-related pictures can be taken.

The second scenario highlights a need for a technology that supports collaboration on mobile devices such as smartphones and tablets, is adaptable and reactive to the current situation of stakeholders such as location, and supports monitoring of the state of shared artifacts such as student plan of actions.

The main commonalities between the two usage scenarios are that there are different roles involved in collaboration, a work pattern is followed and each role has certain activities that need to be completed. The activities can be accomplished by using smartphones. In both scenarios, current user situations are obtained by mobile devices in order to make workflow decisions. While in the first usage scenario, designer's preferences are used to decide which execution path is taken, in the second scenario, location data enables to make decision for monitoring student attendance. Moreover, artifacts are handled in both scenarios, pictures are captured and reviewed in the first scenario, action plans are created and followed in the second scenario. In both cases, states of artifacts are monitored and shared among collaborators to communicate workflow progress.

The domain analysis has been performed and in next sections, the individual features and related concepts of the technology that would support the mentioned requirements are elaborated.

2.3 Workflow Management

Collaborative workflow is a technology that supports multi-party collaboration and can be adapted for various needs. This section presents the basic concepts related to workflow and workflow management.

Basically, the work pattern described in the collaborative scenario can be abstracted into a collaborative workflow, as outlined in Figure 2.1. The simplified workflow process is illustrated from a user's point of view. It shows three roles that participate in this collaboration: Interior Designer, Reviewer and Customer. The roles can be assigned to one or more collaborators. Each role has allocated a number of tasks. The logical order of task execution is outlined, too.

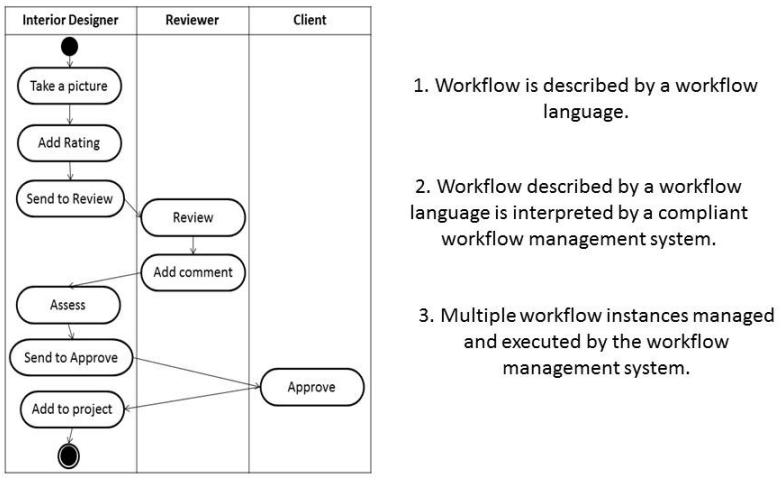

Figure 2.1: Collaborative Workflow Scenario

The work pattern can be described by a workflow language. Then the workflow description is interpreted by a workflow management system which manages and executes multiple instances of the particular workflow.

2.3.1 Basic Terminology and Concepts

Generally, workflow is an abstracted form of a real work pattern. Although there have been some efforts from various vendors to standardise the definition of workflow, workflow language and workflow management system architecture, there is no fully standardised workflow technology that would be used by all and the meaning of concepts can vary depending on the purpose of their application. The terminology used throughout this thesis follows and extends that given in (Hollingsworth, 1995) and in (Aalst & Hee, 2004):

Workflow: The computerised facilitation or 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.

Workflow Management System (WfMS): A system that completely defines, manages and executes workflows through the execution of software whose order of execution is driven by a computer representation of the workflow logic. WfMS provide support in three functional areas:

- *Build-time functions* are concerned with defining, and possibly modelling, the workflow process and its constituent activities;
- *Run-time control functions* concerned with managing the workflow processes in an operational environment and sequencing the various activities to be handled as part of each process;
- *Run-time interactions* with human users and applications for processing the various activity steps.

Workflow Schema/Metamodel: The meta structure of a workflow process defined in terms of a data model.

Workflow Process Definition: The computerised representation of a process described by using a workflow language. The definition contains all necessary information about the workflow process to enable its execution by software. This includes:

- starting and completion conditions;
- constituent activities;
- rules to navigate between activities;
- user tasks to be undertaken;
- references to applications which may to be invoked;
- definition of any workflow relevant data.

Workflow Case Type: A workflow process description for a particular usage scenario. It is a collection of related, structured activities or tasks that produce a specific output for a particular user or application.

Workflow Instance: An occurrence of a workflow process for specific input.

Workflow Partition: A fragment of workflow process executed by one device.

Activity: A logical step in a workflow process.

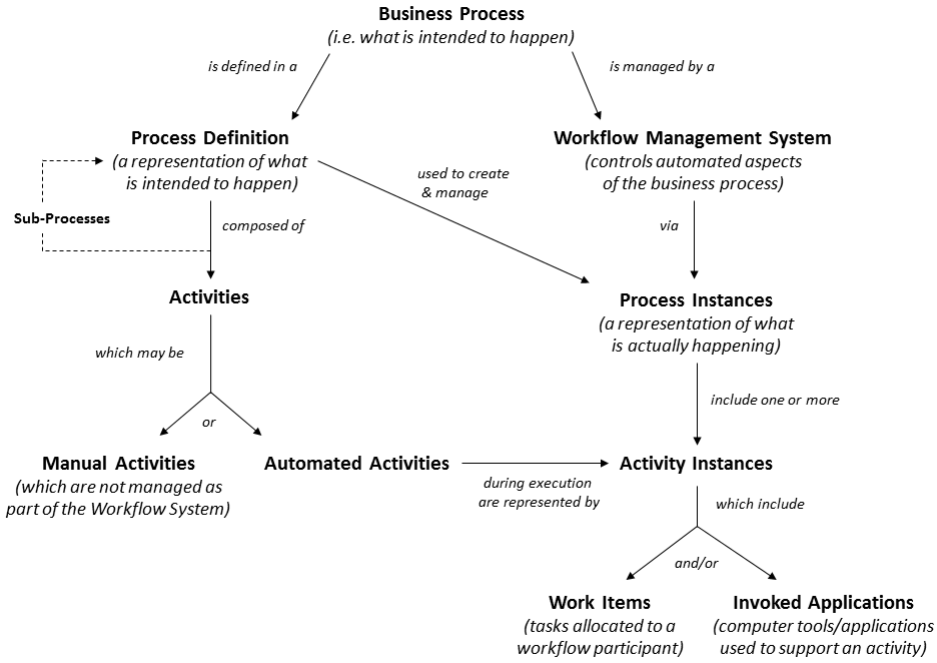

Figure 2.2: Workflow-Related Concepts (Hollingsworth, 1995)

Automated Activity: A step in a process that is performed directly by the workflow execution engine.

Task: Task is a special type of activity representing a unit of work performed by one or more human workflow participants or carried out by another application.

The relationships between the basic concepts are illustrated in Figure 2.2. To clarify, workflow is often named as a business process, however, in this work the term 'workflow' is used rather than 'business process'.

The following terms related to workflow scheduling are extracted from the taxonomy described by (Yu & Buyya, 2006). *Local scheduling* means task handling of a single resource whereas *global scheduling* involves deciding where to handle the task. Managing the task execution depends on the architecture of workflow system. Secondly, *local decision* is based on on the information of current task and *global decision* is based on the information of entire workflow.

2.3.2 Workflow Management

The role of workflow management can be described by five key aspects. The following aspects of workflow specifications, originally distinguished by Jablonski (1996) have been described in the work of (Oren & Haller, 2005):

- The *functional* aspect defines a functional decomposition of activities in the workflow and describes what should be done.
- The *behavioural* aspect defines the execution order and dependencies (control flow) of activities in the workflow and describes when the activities should be done.
- The *informational* aspect describes internal and external data used in the workflow, their dependencies and data-flow.
- The *organisational* aspect defines allocation of work to resources in the organisation: describes who should do the work including the hierarchy and the policies.
- The *operational* aspect describes interaction between the workflow management system and environment.

Mobile workflow management can be centralised with a workflow management system running on a server, or distributed with the workflow management system running solely on mobile devices. As described in the usage scenario, the company does not possess an own server, therefore, mobile peer-to-peer collaboration is required. The peer-to-peer workflow management approach on mobile devices is described next.

2.4 Mobile Peer-to-Peer Workflow Management

As mobile phones are becoming more capable and wireless networks improve, the mobile phone users are expecting the same or better services as traditionally available in the fixed networks. Classical distributed systems designed for desktop computing are usually stable, consisting of multiple autonomous computers. In contrast, systems targeting mobile devices face a number of constraints in terms of location variability, context changes, network data connectivity and resource sharing (Mahmoud, 2004). The development of distributed mobile applications introduces challenging problems as devices have also more scarce resources especially in terms of battery consumption. Thus the distributed mobile applications are often adapted for the constraints, for example, certain operations are performed only when the battery level is sufficient enough.

Mobile distributed systems consist of a set of mobile hosts, connected to the network through wireless links. High-speed wireless connections (Wi-Fi) and technology standards such as Bluetooth enable frequent use of mobile devices in var-

ious environments. Moreover, the wireless communication supports forming peer-to-peer systems which allow data and content distribution between devices by direct exchange without using centralized servers. The administration, maintenance, responsibility for the operation, and even the notion of 'ownership' of peer-to-peer systems are distributed among the users or devices (Androutsellis-Theotokis & Spinellis, 2004). By using the peer-to-peer technology, there is the potential that communication processes are accelerated and collaboration costs through the ad hoc administration of working groups are reduced (Schoder & Fischbach, 2003).

Coulouris et al. (2005) has described the characteristics of peer-to-peer systems as follows:

- Each peer contributes information resources located on the device to the system through a network.
- Each peer has the same functional capabilities and responsibilities.
- No centrally administered system.
- Each node has a certain level of anonymity to other peers.
- Efficient functioning depends on a choice of an algorithm for the placement of data across many peers.

In the pure peer-to-peer architecture, all peers are equal and each peer becomes a supplier and consumer of resources and information services. Application of the decentralized algorithm implies that no device has complete information about the system state and devices make decisions based only on local information. If the peer-to-peer model is used in mobile collaboration, data management and decision making techniques that support tasks, processes or workflows need to be adequately developed.

Workflow management systems purely designed for mobile devices has a decentralised, peer-to-peer infrastructure. The application of the peer-to-peer characteristics into workflow management means that:

- A workflow definition is decomposed into small partitions and the workflow partitions are deployed on participating mobile devices, also labeled as a workflow peer or participant.
- Each device acts autonomously and executes an allocated workflow partition in order to achieve the common goal.
- No single device has a complete view of the global workflow state.

Distributed workflow execution is performed when each device executes an allocated partition of the workflow. Each participating device needs to be aware of the workflow process definition and also all multiple process instances the device is involved in.

It is important to outline that there exist two different views on distributed workflow management: orchestration view and choreography view (Peltz, 2003). From the orchestration point of view, the workflow logic and behaviour of a single participant is described. An orchestrated workflow does not describe coordination between two or more parties. As opposed to the orchestration view, the choreography view focuses on interactions and message exchanges between two or more workflow participants. Choreography refers to an overall process without specifying how it is implemented and describes coordination between two or more parties. The choreography may consist of several orchestrations. In this work, the focus is put more on the orchestration view and on what is happening on a single device rather than optimisation of the interaction among participating devices.

In the following section, existing workflow standards, which have been considered to be used as a base for the workflow described in the usage scenario, are presented.

2.4.1 Workflow Standards

The workflow management field has been shaped from the nineties onward. Various workflow standard models and specifications have been developed. This section presents the most known and adopted workflow approaches.

Workflow Management Reference Model: An almost two-decade-old *Workflow Management Reference Model* developed by the Workflow Management Coalition[1] represents a rich source of specifications for the design and development of workflow management systems. The reference model, described by Hollingsworth (1995), identifies the basic characteristics, terminology, general structure and components of a workflow management system. The major component within the workflow architecture is a workflow enactment service which provides the run-time environment and utilises one or more workflow engine(s). A number of core interfaces and interchange workflow definition formats are constructed around the workflow enactment service to regulate the interactions between the workflow control component and external resources such as process definition tools, workflow client applications, invoked applications, other workflow enactment services and administration tools.

[1] http://www.wfmc.org

Other standards proposed by the WfMC include XML Process Definition Language (XPDL) and Workflow API (WAPI). XPDL is used to describe workflow processes which are deployed to the enactment service. WAPI is a set of API calls and interchange functions around the workflow enactment service used to interact with other resources and applications.

BPMN: Business Process Modeling Notation (BPMN) is a graphical representation of a business process developed by the Object Management Group (OMG) (OMG, 2011). The notation facilitates businesses by giving them the ability to communicate their business procedures in a standard manner. The goal of BPMN is to provide a standardised notation that is easily understandable by various business users. Although BPMN is a useful notation tool to visualise business processes, it has been developed for graphical design and its deployment to the process engine is conditional on transformation to another process format. Because of wide-spread of BPEL in process engines, BPMN processes have often been mapped to the execution format such as Business Process Execution Language (BPEL)

BPEL: BPEL, a shortened name for Web Service Business Process Execution Language, is a standard executable XML-based language developed by the Organisation for the Advancement and Structured Information Standards (OASIS). The full specification of BPEL process can be found online (OASIS, 2007). BPEL has been designed for specifying business process behaviour based on Web Services. Web services are self-contained and self-describing software application components designed to achieve interoperability between applications over the network by using Web standards. Applications integration and interoperability requires a formally defined interaction model that supports peer-to-peer message exchanges, both request-response and one-way formats, between two or more parties. An abstract process is used to describe the interaction model without revealing the internal implementation of the parties involved. Separating the public aspects of business process behaviour from internal business aspects enables to hide internal data management and decision making from other parties and provides the ability to change the private aspects without affecting the public process behaviour. BPEL is an orchestration language, not a choreography language. BPEL defines how multiple service interactions are coordinated to achieve a business goal and provides a mechanism for dealing with exceptions and faults. BPEL utilises XML specifications such as WSDL 1.1, XML Schema 1.0, XPath 1.0 and XSLT 1.0.

BPEL Extensions: BPEL has been used for describing business processes in

many scenarios, however, its standard constructs did not support the various requirements. Therefore, more functionality had to be added and the language has been frequently extended. BPEL4Chor extends BPEL for modelling choreographies (Decker et al., 2007), BPEL4People to cover also human user interactions (Kloppmann et al., 2005) or BPEL4SWS is BPEL for semantic web services (Nitzsche et al., 2007a) are only some examples of BPEL extensions.

BPEL is *a*) platform-agnostic; *b*) expressed entirely in XML, *c*) extensible and adaptable, *d*) accepted as standard and widely adopted, *e*) provides robust interaction model that enables peer-to-peer conversation. Because of the BPEL characteristics, BPEL is used for describing the scenario work pattern and forms a base for MobWEL. Thereby, its basic constructs are described next.

2.4.2 BPEL

The structure of a BPEL process is outlined in Figure 2.3. At the top, the $<process>$ element with attributes such as name, query language, expression language, target namespace, etc. is defined. Global declarations define items that used within the process, including extensions, imports, partnerLinks, etc. For this work, the most important concepts are *variables* and *eventHandlers*.

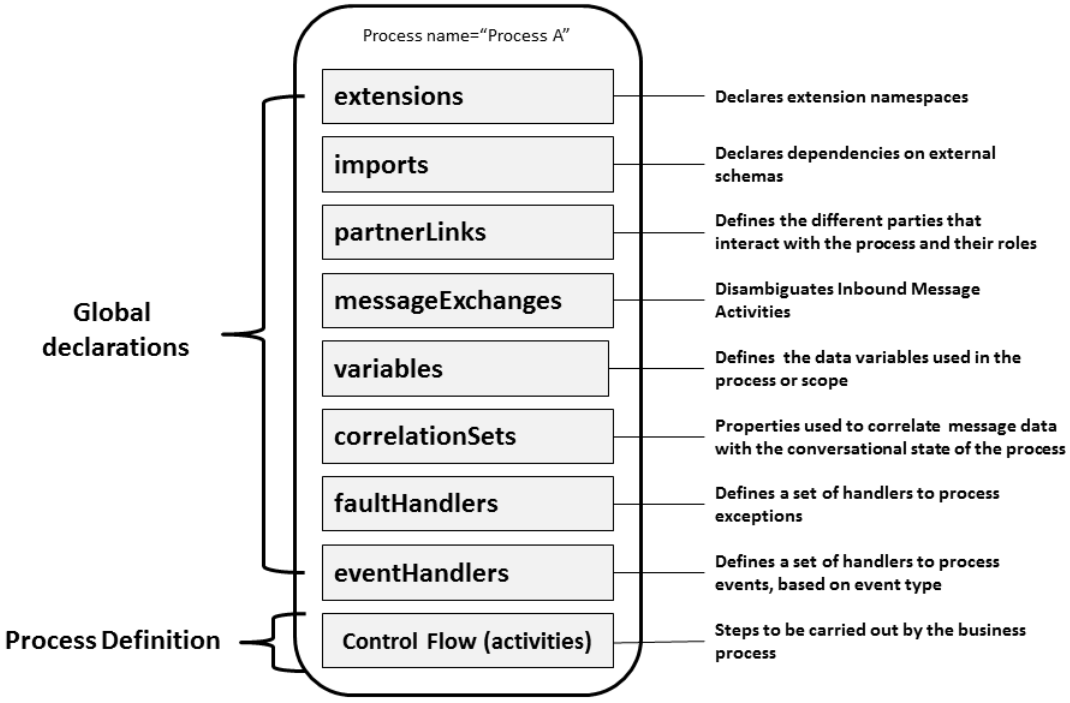

Figure 2.3: BPEL Process Structure (Informatica, 2007)

Explicit data flows are not supported in BPEL and data in BPEL is stored in

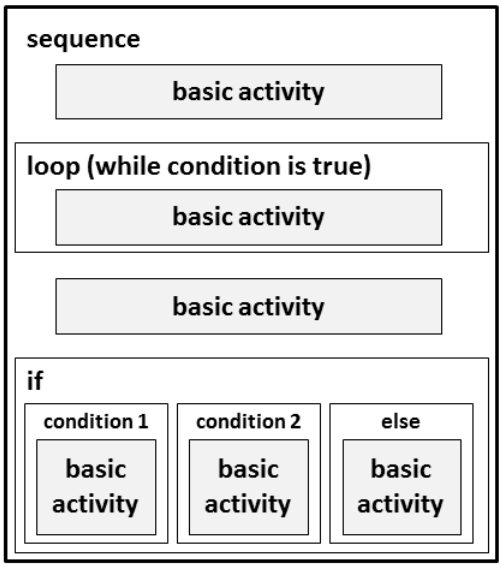

Figure 2.4: BPEL Control Flow

shared **variables**. Data in variables are accessed and modified by activities. Variables can be inputs or outputs of activities such as receive, pick or invoke. Data can be copied from one variable to another variable by using the assign activity. To obtain the value of a variable, a built-in function: '*bpws:getVariableData*' can be used. An XPath expression is used to extract data from an XML document.

BPEL processes can be organised into logical units of work called scopes. To react to certain events or to the expiration of timers during the scope execution, **event handler** can be used. The most obvious reasons for using event handlers are events, for example, a cancellation event to terminate the scope or process escalation when a timer set on the scope expires.

After the global declarations, the process control flow is defined. The flow consists of several individual activities. In BPEL, each activity represents an action that is actually performed. There are two types of activities: basic and structured. Basic activities such as invoke, receive, assign, or exit are atomic. Structured activities such as flow, if, pick or sequence are containers which can contain other activities. To describe each activity is out of scope of this thesis, however, activities that have been altered or adopted in MobWEL will be described in other chapters.

A simple example of control flow in BPEL is illustrated in Figure 2.4. It is important to outline that a process definition contains a single entry point in a form of primary activity, usually the $<sequence>$ or$<flow>$ activity.

BPEL Process Lifecycle

Once a BPEL process is designed, it is deployed to a BPEL-compliant engine. After that, the BPEL process can be instantiated. The process is typically initiated when an incoming message is received (Figure 2.5, step 1). At first, the engine would try to find and match the message to a currently running instance (Figure 2.5, step 2).

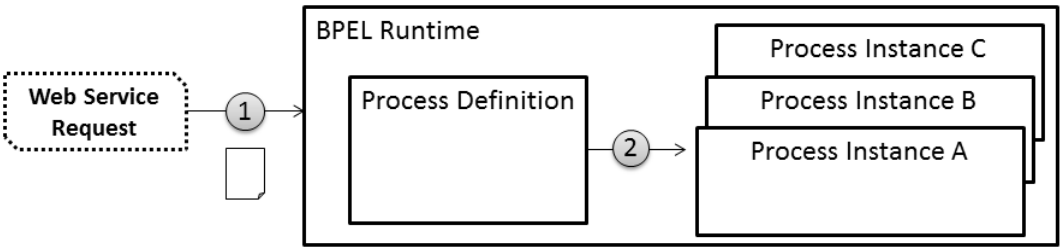

Figure 2.5: BPEL Process Lifecycle

If not successful, a new process instance is created. Each process instance represents a separated thread of execution of the single BPEL process, using its own associated data, and is controlled independently. Process instances have their own internal states which represent their progress towards completion. While the BPEL process instance is executed, corresponding activity instances are created and managed for each invocation of the process activities in accordance to the process definition.

This section has introduced the concepts related to workflow management and standards, particularly BPEL. Next section explores content and object awareness with respect to workflow management.

2.5 Content/Object Awareness

It has been outlined in the introduction that duality of content-centric and activity-based workflow models can be a way to better workflow management. In the workflow described in the usage scenario, a picture is shared among designers. Nearly all objects in their life go through a number of stages. For instance, a book is written, edited, published, read, or disposed. Usually a set of restrictions is placed on the transitions between the stages of an object. The same happens to the picture which goes through a number of states from being created, reviewed, or at the end archived. Placing the picture in the center of the workflow would be limiting for performing other actions. However, content awareness in an activity-based approach can be increased by considering the lifecycle of the processed object.

Several representations for modelling object lifecycles have been developed, often based on finite state machines or statecharts. In state machines, a set of states and directed transitions between the states form a representation (Sipser, 2006). Many of the state-machine representations use the event-condition-action paradigm and associate transitions with events that triggers transitions. On the other hand, although statecharts can be seen an extended version of state-machines, statecharts have been designed more to represent concurrency. For example, the statechart approach has been used in the UML State Machines (OMG, 2007). In this thesis, the state machine approach is used to describe lifecycles of the objects processed in workflows and the object lifecycles are integrated to the process flow. If the view on the object lifecycle is integrated in an activity-based workflow, the workflow becomes object-aware, also referred as content-centric in this thesis. In other words, the object-aware activity-based workflow captures the order of the workflow tasks complemented with the view on lifecycles of objects processed in the workflow.

The following terminology related to content awareness is used throughout this thesis:

Content/ Content Piece: An object type such as image or document processed in a workflow, and declared as variable in a workflow definition.

Content Lifecycle: Content can have a longer lifespan than workflow that creates it. Description of a period involving all stages of the content life from its creation to its disposal.

Content Item/ Content Object: A concrete workflow object processed in a particular workflow instance.

Content Metadata/ Content Attribute: Data providing information about a content object.

It has been outlined in the usage scenario that the technology needs consider needs of individual collaborators. In the next section, context awareness as a concept that can be used to adapt the technology for the requirement is presented.

2.6 Context Awareness

Mobile devices reside in extremely dynamic contexts. Context awareness might have a number of meanings, depending on the domain to which it is applied. Context awareness can be broadly defined as:

'...as an active process dealing with the way humans weave their experience within their whole environment, to give it meaning.' (Bolchini et al., 2009)

or

'Context awareness is of an entity to be aware of the surrounding situations and use the information to perform some tasks.' (Dawson et al., 2008)

In this thesis, the following definition of context is used:

'Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves. Context is typically the location, identity, and state of people, groups, and computational and physical objects.' (Dey et al., 2001)

2.6.1 Context Classification

Context information can originate from various sources. Hence, the heterogeneous context raw data need to be processed into more compact and meaningful context information that is delivered to various context-aware systems. Concerning that context diversity, context classes relevant for the MobWEL are clarified.

Based on the source, the following classes of context have been identified as relevant for the MobWEL workflow model: user context, device context, environmental context and social context, see Figure 2.6.

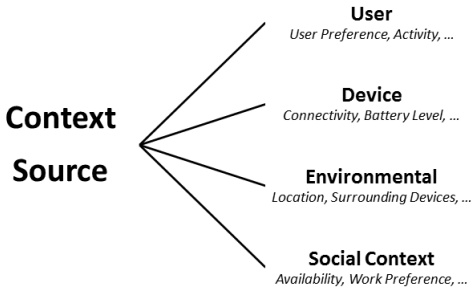

Figure 2.6: Context Types based on the context source

User context deals with all those aspects related to workflow participants and users of the mobile phone. People use mobile devices in a more personalised way and mobile information systems need to be more customisable and adaptable to user behaviour and needs. Personalisation and adaptation of information systems is

usually accomplished by considering additional information about a user, for example, by including more detailed information based on personal preferences (Bierig, 2008). Using user preferences in the workflow concept can lead to more flexible and dynamic workflow structures that can respond to collaborators' needs.

Secondly, smart mobile phones are capable of capturing various types of contextual information. Sensors that can be found in the phones include GPS, accelerometers, digital compass, or proximity sensors. These sensors with other equipment found in mobile phones can be used for acquisition of a wide range of contextual information. Context information can be related to the context of device or environment. Connectivity and battery level are examples of device's context. Environmental context groups all those aspects in which the device is embedded such as time, location or information about surrounding devices.

Finally, significant context to establish cooperative effort is social context. Social context awareness relies on knowing the work context of fellow collaborators, such as their availability, current activity and location (Bardram & Hansen, 2004). A core challenge, especially in the mobile environment, is to provide collaborators with a social context information of each another.

2.6.2 Context Management for Mobile Systems

Contextual information is consumed by context-aware mobile applications. Context aware applications have been described as:

> *'...intelligent applications that can monitor the user's context and, in case of changes in this context, consequently adapt their behaviour in order to satisfy the user's current needs or anticipate the user's intentions.'* (Daniele et al., 2009)

Context management encompasses a number of phases as context monitoring, context acquisition, context processing, context distribution, context consumption and context adaptation. The first four phases are part of context provisioning that is functionally separated from the other two phases.

Much work has been done to enable context awareness and a holistic view of the existing literature has been presented in the survey derived by (Bellavista et al., 2013). The survey of context data distribution for mobile ubiquitous systems puts together recent research efforts. Considering and comparing a significant number of solutions resulted in a unified architectural model and taxonomy for context data distribution. The concepts presented in that work is used throughout this thesis and this section briefly outlines the most important ones.

With relation to computing, context awareness can be seen as:

'...the ability to provide services with full awareness of the current execution environment.'

In the architectural model for context data distribution systems, the context data distribution function need to contain the following main facilities: Context Data Management layer, Context Data Delivery layer and Run-Time Adaptation Support. The *Context Data Management layer* is responsible for local context data handling. Context data processing in this layer include aspects such as context aggregation, context filtering, context data security and history management. Context data is stored in this layer. The *Context Data Delivery layer* supports internal delivery of context data. Finally, the *Run-Time Adaptation Support* provides the dynamic context management to application by considering conditions occurred at run-time.

In this thesis, the following terminology related to context awareness is used:

Context Representation: A model that represents context information. The model can be general or domain-specific. General models are concerned with the generic problem of context representation. Domain-specific models are used to represent data belonging to a certain domain.

Context Information: Higher-level information is derived from raw context data.

Context Situation: The set of context information which describes the current state of the execution environment.

Context Data Aggregation: Operations and reasoning techniques used to merge different context data.

Context Data Filtering: Techniques to control and reduce the amount of transmitted context data.

Context Data Dissemination: A dissemination strategy that enables data flow between sources and consumers. There are two categories of dissemination solution are relevant:

- **flooding-based:** context data are disseminated to all nodes within a scope;
- **selection-based:** context-data reach only interested nodes.

Context Provisioning Platform/Context Provider: A platform that interprets context model and adequately monitors, acquires, aggregates, filters and disseminates context data.

With this section, this chapter is concluded. Its summary is presented next.

2.7 Summary

In this chapter, the usage scenario has been presented, the individual but interrelated features have been elaborated and the related terminology used throughout this thesis has been defined.

The presented usage scenario represents a certain class of mobile applications. Those applications would benefit from a workflow management technology which is *a*) distributed; *b*) adaptive; *c*) context-aware; *d*) content-centric; and *e*) supports mobile collaboration.

There exist numerous workflow management technologies that partially address the requirements. The existing workflow approaches are reviewed and discussed in next chapter.

Chapter 3

Related Work

Contents

3.1 Introduction

This chapter reviews work related to adaptive workflows, content and context awareness in workflows, mobile workflows and is organised as follows. The techniques for workflow adaptation are discussed in Section 3.2. Section 3.3 presents context management frameworks and outlines the need for a workflow-specific context management approach that is tailored for mobile devices. Section 3.4 surveys previous work related to object-aware workflows and content-centric workflow management. Workflow models designed for peer-to-peer mobile collaboration and the support for the execution of such workflows are discussed in Section 3.5. This chapter is concluded in Section 3.6

3.2 Techniques for Workflow Adaptation

Traditional workflows have not been designed to react on the dynamic changes at run-time and the need for more flexible and adaptive workflow has long been recog-

nised (Ellis et al., 1995; Aalst et al., 2000; Buhler & Vidal, 2003). Workflows are organised in a structured way, however, the dynamic nature of the business world requires dynamic processes that deal with a wide range of variations, foreseen and unforeseen changes in the context or environment in which they operate (Schonenberg et al., 2008b).

Flexibility can influence process at any point of its lifecycle. Aalst et al. (2009) has identified five phases of a process lifecycle: design time, configuration time, instantiation time, run time and auditing time. Creation of models without connection to organizational settings is done at design time. Models become more specific and connected to organizational context at configuration time. A process instance is created at instantiation time, executed at run time and completed at auditing time.

Contrary to Aalst, dynamic processes and the impact of their characteristics on three phases of the process lifecycle, namely modelling, execution and monitoring, shown in Figure 3.1, have been discussed in the work of Weber et al. (2009) as follows.

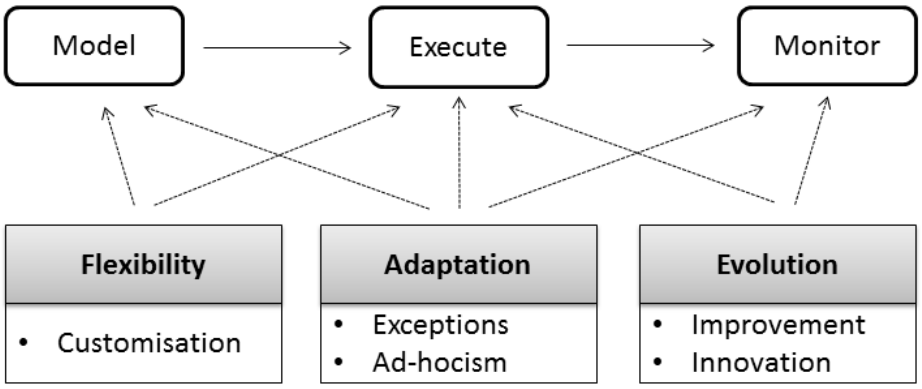

Figure 3.1: Taxonomy for Dynamic Processes (Weber et al., 2009)

Dynamic processes comprise three characteristics:

Flexibility is based on a loosely or partially specified model which is fully specified at run-time, so basically each process instance can determine its own process. Thus the process is not fully anticipated and not strictly prescriptive. Flexible processes are defined in a more relaxed manner. However, this characteristic raises challenges including flexible configuration of process models, either at build time or runtime. Flexibility refers also to customisation.

Adaptation represents the ability of the process to cope with exceptional circumstances. This ability refers to *a*) anticipated exceptions that can be captured in the process definition; *b*) unanticipated exceptions addressed through structural changes of process instances such as adding or deleting of activities.

Evolution means that the implemented process changes when the business process evolves. In this case, the assumption is that there is a pre-defined model which is modified when the changes occur. However, the modification of the process model raises a challenge to handle existing running process instances.

Although there are three phases outlined in that work, our focus is placed on the first two phases, therefore, Table 3.1 reviews only the existing strategies designed for modelling and execution of dynamic processes.

Different classifications of the existing approaches to achieve process flexibility and adaptation have been described in the work of Schonenberg et al. (2008a). In this work, four types of workflow flexibility have been recognised: flexibility by design, flexibility by deviation, flexibility by underspecification and flexibility by change. **Flexibility by Design** basically is the same ability as Weber's *Flexibility by Enumeration*. **Flexibility by Deviation** enables the process instance to deviate from a predefined path without altering its process model. For instance, the deviation allows changes of the tasks order. This flexibility type is similar to Weber's *Dealing with Unanticipated Changes* and is suitable for descriptive rather than prescriptive process models. **Flexibility by Underspecification** is a similar approach as Weber's *Late Modelling* in which the model is underspecified at build-time and completed at run-time. And finally, **Flexibility by Change** enables to modify a process model at runtime meaning that all running process instances are migrated to the new process model. This type of flexibility is particularly suitable for processes which need to adapt to operational changes and unforeseen events. This type falls under Weber's *Dealing with Unanticipated Changes* approach. It shows that Weber's classification covers all flexibility types and is more extensive in terms of categories.

The properties of flexible/adaptive workflows have been considered in the work of Nurcan (2008). The properties, analysed with respect to business process flexibility requirements, are shown in Figure 3.2. The first property is **Nature of the flexibility** which defines whether the process model incorporates the environmental change during the build-time. The capacity of the process model is characterised in two flexibility types: *the flexibility by selection(a priori)* and *the flexibility by adaptation (a posteriori)*. The *flexibility by selection* is based on modelling formalisms which offer dealing with environmental changes without any evolution of process definitions and process instances always conform to the defined model. It means that the process definition should be specified in such a flexible way that no evolution of process definition is needed. The *flexibility of adaptation* allows adaptation of the process definition or its instances during their execution. Resulting process definitions are considered adaptive rather than flexible. Other two properties are applicable only

Table 3.1: Existing Approaches for Dynamic Processes

	Technique	Description
MODELLING PHASE	Granularity Control	Activities are modelled as black boxes, thus they can comprise a number of sub-activities which existence is not disclosed to the process. The flexibility is achieved by performing the sub-activities.
	Flexibility by Enumeration	This approach requires to enumerate all possible execution paths in the process model. During runtime, one path is chosen.
	Process Configuration	Several model variants exist for a business process. Each variant is adjusted for specific requirements. The variants are either specified in separate models or are expressed within the same model in terms of conditional branches.
	Late Binding	This technique enables late binding of services or subprocess fragments at runtime. It is often used for late resource allocation. Basically, a process activity is modelled as an abstract action at build-time. An appropriate service is selected and associated with the activity at runtime.
	Late Modelling	This technique enables the definition of the whole process or its parts at runtime.
EXECUTION PHASE	Dealing with Expected Exceptions	Depending on the exception type raised during process execution, a handling strategy should be chosen. Each strategy describes (a) handling of the work item on which the exception is based; (b) handling the process instance for which the exception is detected; and (c) recovery actions.
	Dealing with Unanticipated Changes	Non-anticipated changes often require structural adaptations of a process schema. There are two options: *a*) A complex and error-prone way that requires the application of multiple change primitives such as add node, remove node, add edge, remove edge and move edge; *b*) A way based on a set of change patterns: high-level change operations like insert process fragment between two sets of nodes.
	Dealing with Uncertainty	Decisions regarding the exact flow of control are deferred to runtime. The process schema is not fully specified as some parts of schema remain unspecified. Examples for such techniques: late binding, late modelling, late composition of process fragments, multi-instance activities, data-driven processes.

when the flexibility is a posteriori. **Nature of impact** defines whether the definition or instances will be affected, and **Nature of change** defines why the transformation of the process is required.

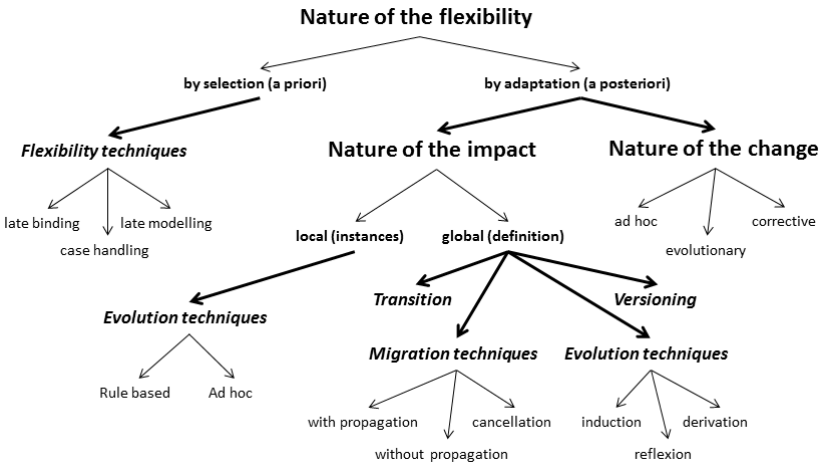

Figure 3.2: Flexibility Classification (Nurcan, 2008)

Redding et al. (2010) have pointed out that despite the large number of proposals for flexible workflow support, the targeted process model is still composed of activities and flexibility is achieved by two specific approaches: allowing runtime deviations or minimalistic specification of flow dependencies in which a larger part of behaviour is forecasted in advance.

As shown, there have been various techniques developed to achieve workflow flexibility. One question that needs to be asked, however, is which mentioned technique can be applied in the case of MobWEL workflows. In peer-to-peer mobile collaboration, the workflow management system is deployed on each single mobile device. As MobWEL workflows need to be executed on multiple mobile devices, the modifications of their workflow descriptions on each device could cause lots of inconsistencies and thereby, the techniques that include changes in the process schema during run-time are not suitable. Similarly, process definition underspecification or using the black box activities approach would increase processing on each device, so these techniques are not considered as suitable either. Because of this, the environmental changes must be anticipated and captured at build time. Thus, the flexibility techniques that have been considered as most relevant and applicable for the work in this thesis are: late binding and late modelling.

The next sections discuss workflow adaptation approaches, in particular adaptation for the integration of context changes and content behaviour. Previous work related to workflow contextualisation and context management within workflows is reviewed next.

3.3 Context Management

This section reviews previous work on context modelling and management, and highlights the need for a workflow-specific context management approach.

3.3.1 Workflow Contextualisation

Workflow contextualisation has been addressed in a number of works, however, there are many research challenges to make context-aware workflow systems ready for practical use.

The extrinsic drivers for process flexibility which can be found in the context of the process such as time, location, weather or performance requirements have been studied in the work of Rosemann et al. (2006a). The study suggests an explicit consideration of these contextual factors in the design and modelling of business processes. The notion of *context-aware business processes* has been established in the work, subsuming the questions related to context conceptualisation and the impact of context on process design. Although types of context are defined and the use of context information is depicted in a process model, the study only helps to gain a better understanding of the different types of context and their impact on business processes from the modelling perspective.

The study has been extended in the work of Rosemann & Recker (2006b), pointing to three main research challenges related to a context-aware process design: context description, design for context and process adaptation. *Context description* refers to the identification and description of context variables relevant for a business process. In other words, it refers to a relevant context model which is integrated with existing metamodel of a process modelling language. *Design for context* refers to the incorporation of contextual elements in the design of business process. In particular, how the knowledge can be embedded and utilised in the process design for context. Finally, *Process adaptation* considers the support for context-aware business processes and need for the design of adaptive process management systems which support the adaptation of processes to context changes. While this paper only envisages the requirements and challenges for context-aware business processes without their further elaboration, another work of Rosemann et al. (2008) presents context integration and a metamodel that integrates context with the traditional process perspective. Although the metamodel has formalised the idea how processes can be used to identify context that is relevant for the process, the challenges presented in the previous work have not been fully addressed.

A more comprehensive approach towards context-aware workflows has been described in the work of Wieland et al. (2007). In that work, the Nexus platform

has been chosen as a context provisioning platform. The Nexus platform supports two access patterns: context querying and context events broadcasting. In context querying, the Augmented World Query Language (AWQL) is used to obtain context information at real time. On the other hand, by employing the publish-subscribe mechanism, notifications about context events observed by the Nexus platform are sent to the application. Thus the process model is adapted to support *a*) waiting asynchronously for a context event that triggers a certain action (context event); *b*) synchronous querying of context data and its storing into internal variables (context query); *c*) branching and routing process control flow based on context (context decision). BPEL is used for the implementation of such context-aware workflows and the extended BPEL version is named Context4BPEL. The concept for modelling context-aware workflows introduced in the work and their realisation by extending BPEL is a promising practical approach for the workflows adaptation to changed environment. The workflow adaptation to context is achieved without losing generality. It means that although Context4BPEL has been coupled with the Nexus platform, the binding is only one possibility among others and the process model can be coupled with other context management platforms.

Another approach that facilitates adequate specification of context variables and contextualised business process is described in the work of Vara et al. (2010). The methodological approach to business process contextualisation called COMPRO and described in that work is based on context analysis, a technique that supports context reasoning and discovery of its relevant properties. The approach suggests to model initial business model at first. Then the business process context is analysed and context variants determined. After that, the initial model is extended and a contextualised business process model is produced. Basically, the first business process is underspecified and contains some black-box activities which are influenced by context. After the context analysis is conducted, the activities are fully specified. This approach offers a systematic way to identify and discover context variables, however, in this approach only context decisions are considered within the business process, without incorporating context events. Furthermore, there has been only a little attention paid to context provisioning and the way how context information is obtained has not been fully elaborated.

Contextualisation of a role-driven business process modelling (RBPM) is another approach that addresses integration of context awareness in business process modelling (Saidani & Nurcan, 2007, 2009). In role-driven business processes, there are roles played by actors and during the execution of a business process, actors perform activities. In this approach, contextual information is used to enhance the adequacy of the task assignments during the execution of a business process. It

refers to the fact that if all actors, who play a given role, are unavailable to perform a certain function, the function is performed by one from the available roles. Therefore, the business process contextualisation approach is very specific for operations-to-role assignments and thus not very applicable for our usage scenario.

The conceptual foundations of a discrete service named Worklet service that transforms static workflow processes into dynamically extensible process instances have been described in the work of Adams et al. (2009). To accommodate flexibility, these criteria would need to be satisfied *a*) a flexible modelling framework in which process models are considered as guides rather than prescriptions; *b*) a repertoire of actions available for each task during execution of each process instance; *c*) dynamic/contextual choice from the repertoire at runtime; *d*) dynamic process evolution meaning that the repertoire can be dynamically extended. Therefore, in this framework each task is provided with the ability to be linked with the repertoire of actions and the action is contextually and dynamically chosen at runtime to carry out the task. In addition, a repertoire of exception-handling processes needs to be provided. The repertoire-member actions are presented as worklets. The Worklet Service has been implemented as a YAWL (Yet Another Workflow Language) custom service. In addition, the following taxonomy of contextual data has been introduced with this framework:

- *Generic(case independent):* are the attributes that occur within any process;
- *Case dependent with a priory knowledge:* a set of data known to be pertinent when a case is instantiated;
- *Case dependent with no a priory knowledge:* a set of data that only becomes known at runtime when the case is active.

The Worklet Service allows a model to be considered from many levels of granularity and supports late binding of processes. It maintains a repertoire of actions that can be constructed during design and/or runtime and invoked as required. However, due to limitations of mobile devices and issues resulting from peer-to-peer workflow execution, the framework is too complex and thus not considered as suitable for our work.

The workflow contextualisation approaches presented in this chapter highlighted the need for an explicit context description and identification of context variables that influence process design and execution. (Wieland et al., 2007) has pointed out that workflow meta-models should support context modelling and its use in workflows. Thus some existing context modelling and management approaches are discussed next.

3.3.2 Context Modelling and Management

Numerous general context frameworks have been developed to facilitate context modelling, recognition, reasoning and management.

One of the earlier widely acknowledged works, the Context Toolkit, provides a framework for context modelling and management (Salber et al., 1999; Dey et al., 2001). Using context in applications is difficult because of the nature of context information. Context data is dynamic and acquired from multiple unconventional and heterogeneous sources. Context data must be abstracted to be useful for applications. The Context Toolkit framework has been developed to address the difficulties. The framework is based on the idea of context widgets, the concept inherited from the graphical user interface (GUI) widgets. Context widgets mediate between the application and its operating environment. A context widget represents a software component which provides access to context information. The widget hides the complexity of actual sensors, encapsulates and abstracts context information. Context widgets have states represented by a set of attributes and behaviour. Applications can register their interest to be notified of context changes or access the widgets through provided methods. This framework separates context acquisition from use, and also supports context composition. The framework addresses the general difficulties with using context information, however, it does not suggest the structure and overall management for organising and controlling of widgets lifecycles, and is not focused on mobile platforms.

Focus on mobile platforms with aim to simplify the development of context-aware mobile applications has been addressed in the context management framework presented in the work of (Korpipaa et al., 2003). This framework enables higher-level context abstractions and provides systematic techniques for context acquisition from multiple sources. The framework is based on four main functional entities: context manager, resource server, context recognition service, and application. Context Manager acts as a central server which can be queried to gain context data and provides context notification services for other entities which function as clients. Context data is obtained by using the resource servers which then passes them to the context manager. The framework manages context information systematically by providing a common structure and an ontology for representing context information. The framework has been designed for the Symbian platform [1]. The framework employs the context manager to provide control between the acquisition and use of context information by applications. However, the ontology is static and the higher-level context abstractions are predefined and the same context values are used by all applications. Thus applications cannot determine their own customised derivation of

[1] www.symbian.com

the context values. It influences the optimal use of context manager as applications obtain also context values they might not be particularly interested into. Although the framework focuses on mobile platforms, it is based on a centralised operating and therefore, only hardly applicable for peer-to-peer mobile collaboration.

Context modelling techniques and a preference context model for representing context-dependent application requirements are described in the work of Henricksen & Indulska (2006). In contrast to other more infrastructure-centred context management approaches, the framework developed in the work integrates a set of well-defined context modelling abstractions. The conceptual foundations of the framework lie in three separate but closely integrated modelling approaches: *a*) Context Modelling Language (CML) that supports the specification of an application's context requirements; *b*) Relational Representation that supports the management and persistence of context information in a repository; *c*) Situation Abstraction in a form of predicate logic supports the specification of abstract classes of context. Additionally, a preference model is developed that facilitates the decision-making process which is supported by the use of machine learning techniques and preferences based on the ranking of choices. The modelling approach developed in the work is based upon a set of reusable concepts such as context definition and context processing components. Further, a high degree of user customisation is supported. However, the CML models require a precise description of various aspects and their relationships. This approach can be beneficial for general context modelling, however, it is too complex and impractical to be used in conjunction with context-aware mobile workflow management systems.

There have been other context modelling and management frameworks developed, each of them targeting a certain set of challenges. For example, challenges such as distribution, mobility and resource-constraints have been addressed in the context modelling framework for pervasive computing systems called MUSIC (Reichle et al., 2008). With emphasising the need to establish an ontology of concepts, the work bases the framework on three abstraction layers: conceptual, exchange and functional. Whilst the conceptual layer enables the definition of context artifacts, the exchange layer utilises the interoperability between devices and the functional layer refers to the implementation of the context model. The framework is comprehensive but requires the use of an ontology.

With increased context data dissemination, another example of a framework addressing the corresponding issues is the Nexus Platform (Grossmann et al., 2005). The goal of the platform is to support all kinds of context-aware applications by setting up a global and detailed context model. This approach is designed to provide context information acquired from multiple geographically dispersed sources. How-

ever, the platform operates on a context server, therefore, its use is impractical in our scenario.

As discussed, there have been various context management frameworks developed. However, the frameworks are either too general and designed to be used with a wide range of context-aware applications, or the frameworks are designed for a specific class of scenarios and a particular set of requirements. As shown later in this thesis, none of the approaches sufficiently fulfills our requirements, therefore, a workflow-specific context management approach that combines the most suitable features of existing approaches is proposed as a part of the MobWEL workflow approach.

The next section presents work related to data-driven and object-aware workflow processes.

3.4 Object Behaviour Modelling and Management

The need for more data-driven and object-aware workflow processes has been recognised in the research community and various approaches, presented in this section, have been developed to outline or address the associated challenges.

3.4.1 Artifact-Centric and Object-Aware Workflows

The idea of object awareness, although in a slightly different meaning, has been developed in a framework based on Proclets (Aalst et al., 2001). A proclet represents only one aspect or element of workflow such as *a*) lightweight workflow process with a knowledge base containing information on previous interactions; *b*) object equipped with an explicit lifecycle; *c*) active document. In the approach, workflow definitions have been divided into smaller interacting proclets. Proclets interact via channels. There is also a naming service that enables for proclets to find each other. However, based on our terminology and understanding of concepts, the framework is more applicable for building flexible processes rather than for object-aware process models defined in Chapter 2.

A component model built around the concept of Adaptive Business Object (ABO) is presented in the work of Nandi & Kumaran (2005). The model allows designing workflow processes around a set of key business entities. ABOs are abstractions of business entities with managed state manifested in a *currentState* attribute. The lifecycle is defined by using a finite state machine. People and applications may interact with ABO, however, the access control to the objects is adaptive and based on business roles. The lifecycle and behaviour of ABOs are designed by using the

UML state charts and state transitions are labeled as [Event]/[Condition]/[Action]. A business process is viewed as a sequence of activities used to create, manipulate, and close ABOs. Although the component model is based on ABOs and the artifacts are considered as important as the activities, the data and remote actions are part of state transitions. That makes the component model more artifact-centric because only actions which are relevant for that artifact can be expressed.

The belief that traditional process modelling approaches which focus on activities fail to capture informational structure relevant to the business contexture has been supported in the work of Liu et al. (2007). Business artifacts, such as Purchase Order or Insurance Claim, are seen as an additional dimension with which business analyst can model their business. So that work is based on the idea that business process operational modelling engages also business artifact discovery and modelling the lifecycle of the discovered artifacts. The discovery phase produces the actual product or artifact of the process. A business artifact, initially defined by Nigam and Caswell (2003), has been described as *'identifiable, self-describing unit-of-information through which business stakeholders add value to the business'*. Therefore, the artifact has ID, uniquely identified within the business, and attributes, named in the way that their use within the domain is apparent. Information contained in the artifact form the information model. The modelling phase of the lifecycle of the discovered artifacts is achieved via adapted business tasks and using the concept of *input and output ports* which live in the context of task. The ports are the recognisers for artifacts. The business operational modelling approach incorporates the contexture of a business and the behaviour of the business is described in the context of artifacts. This approach highlights the need for discovering and modelling of the artifact behaviour, however, the way how the artifact behaviour and the control process are integrated is not suitable for running on mobile devices.

Business artifact integration is usually limited on the inputs and outputs of certain workflow activities, and impact on its lifecycle is neglected or hardly visible (Bhattacharya et al., 2009). Thereby in the work, the notion of Business Artifacts with Lifecycles, Services and Associations, referred as the BALSA workflow model, is developed. The artifact-centric workflow model comprises four key elements or dimensions. The elements are as follows:

Business Artifact Information Model: An information model that holds information about a given business artifact and contains information needed to complete workflow execution. The artifact is self-contained and has an identity. A set of artifact attributes stores the data needed for the workflow execution and use services in workflow which can create, update and delete the values of attributes.

Business Artifact (Macro-Level) Lifecycle: The possible evolution of a business artifact is captured in its lifecycle that is usually described in terms of stages and represented using a specific version of finite state machines.

Services (Tasks): A unit of work is encapsulated in a service (in the Service-Oriented Architecture), workflow task or activity. The service makes changes to one or more business artifacts.

Associations: There is a family of constraints for using services including procedural specification, precedence relationships among the services, between services and external events.

By varying the paradigms used to specify the information model, lifecycle, services and associations, numerous BALSA models can be obtained. For example, the information model might be specified as attributes with scalar values or XML; the lifecycle might be specified by using flowcharts or finite state machines; and the services might be specified in black box or BPEL activities. The choice among the various paradigms really depends on the intended area of application. Therefore, the BALSA workflow model is flexible in its use and provides a guided framework how to develop a customised artifact-centric workflow approach. This motivated to use the BALSA model in building the MobWEL workflow approach.

The specification of the artifact-centric workflows with the vision of four explicit and inter-related dimensions described in the BALSA workflow model has been also supported in the survey of Hull (2008). As a data-centric approach to workflows has emerged, the survey highlights the research results and challenges raised by the approach to date. The challenges in building the data-centric and artifact-centric workflows are as follows:

Design principles in support of usability and flexibility: The first challenge is to find a mechanism which would enable to specify data-centric workflows in an intuitive and concise way. At the same time, two forms of flexibility have to be supported: *a*) highly varied operations in workflows for different clients, products, context and regions; *b*) evolution of the workflow schema.

Componentisation and composition: Easier refinement and quick constructions of workflows can be addressed by improvements in workflow re-use and composition. Workflow partitioning into natural components ensures that different combinations of the components can be built, workflow schema evolution is more flexible and open for globalisation or outsourcing.

Implementation and optimisation: Efficient implementation of the data-centric workflows is still open with challenges in concurrency control, indexing, data staleness, performance monitoring and workflow schema evolution.

Foundations, static analysis, and synthesis: There has been little understanding about modelling of data and workflow process in a interconnected setting. Other issues are related to constructions or synthesis of workflow schemas from high-level perspective.

User-centric aspects: Development of a natural approach for visualisation and representing of workflow schemas remains open.

Monitoring and tuning: Challenges in monitoring and the performance of operations.

Linkage to business strategy: A need to specify the business strategy by using a relatively small number of workflow conceptual constructs.

Evaluations of paradigms, tools, and methods: A very challenging area is the evaluations of the artifact-centric approaches. It is hard to measure in an objective manner the costs or benefits of different approaches and therefore, compare them and decide what approach should be used.

The list shows that there are numerous challenges that should be taken into account and considered in building the content-centric MobWEL approach.

Object awareness and object lifecycle modelling can be seen as the subjects of some other recent studies. A framework for integrated process and object life cycle modelling has been developed by Wahler (2009). Business objects processed by business processes can be associated with distinct states abstracted from the details of the performed tasks. The states mark the milestones of the overall processing and are useful in communicating progress to stakeholders who are unaware of the exact process logic. The state and state transitions are captured in an object lifecycle model. The object lifecycle model abstracts from the underlying business process. The object lifecycle model and process model represent complementary and overlapping views on the operations of a business. Consistency of two overlapping models is seen as a critical issue that is addressed in the framework.

Another approach to workflow process design is the Product-Based Workflow Design (PBWD) (Vanderfeesten et al., 2008, 2011). As opposed to evolutionary approaches that try to improve existing situations, the PBWD approach starts the design from scratch. The base of the approach is a description of the product called a Product Data Model that is produced by the process under consideration. Workflow execution is based on the product data model without the need for a process

model that would guide the execution. Although the approach is dynamic and offers some benefits, its use is more suitable for centralised workflow management solutions.

Unlike other approaches which base entity lifecycles on variants of finite state machines, the work of Hull et al. (2011) introduces the guard-stage-milestone lifecycle model. This model is evolved from previous, already described approaches which have been based on business entities with lifecycles. The Guard-State-Milestone (GSM) approach for specifying Business Entity Lifecycles (BEL) is a more declarative than finite machines and supports hierarchy within a single entity instance. The notion of *stage* can be found at the core of GSM lifecycle models, based on three constructs: *a*) milestone - an objective expressed using a condition over the informational model and possibly triggering event; *b*) stage body - containing one or more activities intended to achieve a milestone; and *c*) guard - a condition that enables entry into the stage body. So the GSM metamodel has four key constructs: information model, guards, stage bodies, and milestones. All business-relevant information about an entity instance is recorded in the information model. This contrasts with typical process-centric approaches in which business-related data arises in process variables but is almost impossible to access the data from outside the scope of use. The information model, therefore, includes *a*) data provided by human stakeholders; *b*) data related to activated external services; and *c*) data that holds a log of what has happened to the entity instance so far. Following this, the attributes are divided into three categories: data attributes, event attributes, and milestone and stage info. The GSM approach is a new variant of entity lifecycle specification. The use of milestones in the GSM approach is an interesting concept because it enables to track the status of object evolution and communicate the progress among collaborators. So it can bring many benefits in peer-to-peer workflow management, and thereby the concept has been adapted in the MobWEL workflow approach.

Integration of object-awareness into process management system is addressed in the PHILharmonicFlows framework (Künzle & Reichert, 2011). In the framework, a holistic approach to integrate data, processes and users is undertaken and the following characteristics for process and data integration have been derived:

- During process execution, the behaviour of objects needs to be considered;
- Interactions between objects must be taken into account;
- Process execution needs to be accomplished in data-driven manner;
- Process-related objects can be accessed and managed at any point of time.

As opposed to other approaches, there is a tighter integration of processes and data in PHILharmonicFlows. To capture both object behaviour and object interactions, two levels of granularity, namely micro and macro processes, are defined. Micro processes are defined for each object type and express object behaviour by using states and transitions. Macro processes model multi-object processes and specify object interactions. In PHILharmonicFlows, object behaviour is combined with data-driven process execution. However, the requirements set that is met in the development in the framework differs from the requirements for MobWEL workflows usage.

Most of the recent studies support the use of a process model and an object model as two complementary assets in object-aware workflows. However, there has been only little attention paid to the integration of context awareness into both models and the existing approaches do not provide the necessary level of expressiveness to capture it. Moreover, the approaches are not tailored for mobile platforms. This has motivated the need for MobWEL workflows to support context-awareness in both models and be more mobile-specific.

3.4.2 Support for Content Lifecycle Management

The future of content-centred collaboration based on building solutions around the content items, tasks and ideas has been envisioned by Erickson et al. (2009). Content-centred collaboration is based on the vision of hosted collaborative spaces called content spaces. Content spaces represent work and authoring environments in which collaborators can customise functionality around content items. Active behaviours, simple automated coordination of human tasks, can be attached to a content space so collaborators can easily develop and share work patterns. The model of active behaviours is based on events processing as content is created, modified, and deleted within a content space.

The idea of collaboration based on content spaces has been also used in Alfresco (Shariff et al., 2009; Potts, 2008). Alfresco is an open source platform for Enterprise Content Management solutions. At its core is a repository for content like documents and multimedia. The repository can be extended to accommodate business-specific metadata and business logic. A space in Alfresco is a smart folder that contains content and sub spaces. The additional features of a space include security, business rules, workflow, notifications, local search capabilities and special views. Content-centric workflow in Alfresco are defined and managed by using content spaces. Therefore, various spaces are created, each dealing with a different stage the content flows through.

The idea of content space has been applied in the management of context-aware content behaviour. The state machine approach is used to describe the lifecycle of objects/content. The lifecycle model is defined independently but similarly as process model at build-time. However, when the workflow process is instantiated and multiple content items are created at run-time, the management of such lifecycle requires an approach that would enable processing and monitoring of all content items without creation multiple state machines. This can be achieved when each content state is represented by a content space which is capable to hold multiple content items at the same time.

3.5 Mobile Peer-To-Peer Workflow Execution

To manage and execute workflows in a peer-to-peer manner require a workflow model and workflow management system adapted for mobile platform. In this section, work related to workflow adaptation for mobile platforms is reviewed.

The need for device-oriented processes has been highlighted in the work of Pajunen & Chande (2007) in which a mobile workflow system that includes a mobile workflow engine is discussed. In the work, BPEL is used to describe the mobile processes. BPEL-based process can interact with internal mobile services such as Browser service, Calendar service, Map service, or Messaging service; and cooperates with external services and other external BPEL-based processes which reside on other mobile devices. The deployment of workflows in a mobile environment is more dynamic and requires the support of extra process capabilities. Three kinds of extensions have been suggested: XPath extensions to query local deployment settings, extensions to BPEL implementation to better leverage mobile execution environment, and finally support for mobile specific protocols such as SMS, MMS, Bluetooth. Although the system has been designed for the mobile environment, context awareness has not been considered.

A different approach to integration of processes into mobile computing system in which mobile devices do not behave only as process participants is proposed in the work of Kunze et al. (2006, 2007). This approach introduces a process description language and an execution model for distributed business processes. The requirements for the description of distributed mobile processes include the ability to express *a*) the business logic with its data and control flow; *b*) the participating parties; *c*) routines to recover failures; *d*) handling of communication failure; *e*) handling of the distribution of the process itself. Activities of mobile processes can last very long like hours, days or weeks, and the device environment can change dramatically in that time. Thereby mobile process execution relies on contextual information

and the Context Toolkit is mainly suited for context data acquisition. The DEMAC process description language is an XML-based language that inherits some XPDL constructs and extends it by introducing new constructs such as exception handlers, connection reset handlers, transaction, and transaction activities. The new constructs allow a distributed handling of the process over heterogeneous systems. The approach provides the support for cooperation of mobile devices, however, it is not demonstrated how the language is adapted to use the context information. In addition, it is outlined that despite the fact that BPEL offers very powerful elements to link tasks and deal with unexpected circumstances, it does not provide concepts for distributed process execution.

Although BPEL has been designed for processes in a Web Service world, it can be adapted for mobile peer-to-peer workflow execution. $BPEL^{light}$ is a WSDL-less BPEL extension which decouples process logic from interface definition (Nitzsche et al., 2007b). Communication between two services in BPEL is enabled by using a partner link type. The $BPEL^{light}$ approach allows modelling of the workflow process independently of Web service technology by introducing a single type of interaction activity. The interaction activities ($<receive>$, $<reply>$, $<invoke>$, $<pick>$) defined in BPEL 2.0 are resumed by this single activity - the $<$interactionActivity $>$activity. In addition, $BPEL^{light}$ processes can be modelled without specifying interface definitions (port types), hence they can be used in non-WS environment. By discarding the static specification of port types, this approach enables direct message exchange between workflow partners. The $BPEL^{light}$ approach offers better flexibility than BPEL and is more suitable for the description of peer-to-peer mobile workflows in which all participating and interacting parties are known and determined. Hence some concepts defined in $BPEL^{light}$ have been adopted in MobWEL.

BPEL has been initially deployed only on heavyweight server platforms such as Apache Tomcat. However, to support device-oriented workflow process and its execution, the engine has been also adapted for mobile devices. One of such solution is Sliver, an open source workflow engine that supports the execution of SOAP services and BPEL processes on a wide range of devices (Hackmann et al., 2006b). The design of the lightweight engine has been influenced by three traits: *a*) it needs to have a smalll storage and memory footprint (including all libraries); *b*) it depends only on the JAVA APIs that are available on all devices; *c*) it must supports a wide range of communication protocols. Sliver depends only on two external libraries and can be deployed on a broad range of devices. The architecture of Sliver is shown in Figure 3.3.

Communication components are separated from processing components. At the lowest level, the transport layer supports the exchange of message object in the form

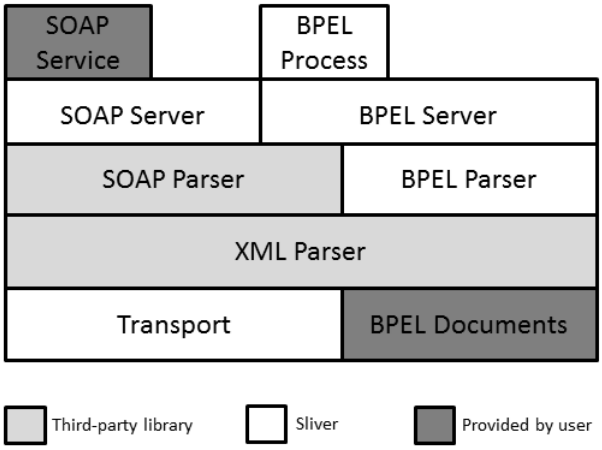

Figure 3.3: The architecture of the Sliver execution engine (Hackmann et al., 2006b)

of serialised XML strings via various network media and protocols. XML and SOAP parsers are used to convert these strings to and from Java objects. SOAP components do not depend on BPEL components. BPEL parser with XML parser layer converts BPEL documents into executable processes. The processes are hosted by the BPEL server layer. The BPEL and SOAP servers consume requests coming from the transport layer and routes them to the corresponding processes or services. Sliver engine offers many benefits such as it has a relatively small footprint, supports many communication protocols and is capable of executing BPEL processes. Another point in favour of Sliver is that it provides the core features for building of more complex mobile workflow management systems and supports numerous communication protocols. This motivates to use the Sliver engine as the basic stone in the MobWEL workflow management system.

3.6 Summary

In this chapter, previous work related to workflow adaptation, context modelling and management, object-awareness in workflow approaches, and mobile peer-to-peer workflow execution, has been reviewed. Firstly, the techniques for workflow adaptation have been discussed. These techniques are applicable either in the workflow modelling phase or in the workflow execution phase. After that, existing approaches for workflow contextualisation and context management have been presented. It has been highlighted that a context model should be included in the workflow metamodel, and workflow contextualisation requires applying of a context management approach that would be designed for the given area of use. Object behaviour management approaches have been presented next. Modelling of content object lifecy-

cle has been recognised as an additional dimension in workflow modelling, and this idea has been elaborated in some recent studies. Finally, work related to mobile peer-to-peer workflow execution has been discussed.

The goal of this work has been to adapt the collaborative workflow technology that would *a*) enhance mobile peer-to-peer collaboration; *b*) integrate context awareness so that workflows can be adapted to individual collaborators' needs and circumstances; and *c*) incorporate extra management support for content behaviour in order to increase content awareness in workflows and enable communicating progress among collaborators.

The existing workflow approaches presented in this chapter only partially supports all listed requirements. The summary of the most relevant technologies and approaches is presented in Table 3.2.

Table 3.2: MobWEL Constructs

Workflow Approach	Mobile Collaboration	Context Awareness	Artifact-Centric Approach
BPEL	Not considered in design	NO	NO
BPEL$_{light}$	Provides light interaction model	NO	NO
Context4BPEL	Not considered in design	YES	NO
BALSA	Not considered in design	NO	YES
MobWEL	YES	YES	YES

With this, the first part of the thesis is concluded. In the following part of the thesis, the MobWEL workflow approach is presented. MobWEL extends BPEL, using constructs from Context4BPEL and BPELlight, and adopting elements from the BALSA workflow model.

Part II

Contribution

Chapter 4

MobWEL Definition and Syntax

Contents

4.1 Introduction

Mobile workflows capabilities can be increased by making mobile activity-oriented workflows context-aware and content-centric. With context awareness integrated, workflows are adapted to collaborator's needs and circumstances. Extra management support for content behaviour added in activity-based workflows enables communicating progress among collaborators. Chapter 2 has provided the background to the domain and introduced the usage scenarios with listed requirements. Chapter 3 has discussed suitability of existing workflow technologies to be applied for the given class of mobile collaboration scenarios. As shown, none of the existing technologies fully meets the requirements. Thereby, to address the need, the MobWEL workflow approach has been designed. In this chapter, the MobWEL workflow language is defined, and its syntax and metamodel are presented. The mobile workflow execution language can be used to specify mobile context-aware and content-centric workflows. MobWEL extends BPEL, using constructs from existing workflow

approaches, Context4BPEL and BPELlight, and adopting elements from the BALSA workflow model.

The remainder of this chapter is organised as follows. An anatomy of a MobWEL workflow process and an adapted workflow from the usage scenario are presented in Section 4.2. The MobWEL workflow process is constructed of a number of workflow parts which are individually described in next sections. The control flow of the MobWEL workflow is described in Section 4.3. Section 4.4 is dedicated to the workflow representation of a group of collaborators. Workflow-specific context definition is explored in Section 4.5. Finally, Section 4.6 describes context-aware content lifecycles. This chapter is summarised in Section 4.7.

4.2 MobWEL Workflow Approach

In this section, the MobWEL workflow approach is introduced and the anatomy of MobWEL workflows is outlined.

4.2.1 Adapted Collaborative Workflow

The MobWEL workflow approach is illustrated by using the workflow from the usage scenario, as shown in Figure 4.1.

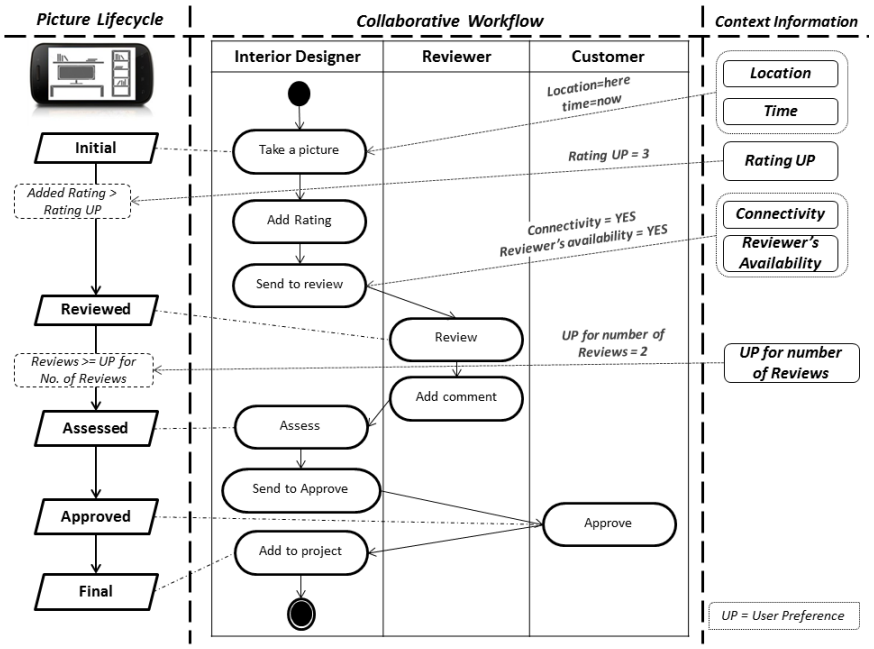

Figure 4.1: Collaborative Workflow Scenario

The simplified control flow of the workflow process is illustrated from a user's point of view in the middle of the figure.

In the first place, this workflow process becomes content-aware by integrating the lifecycle of the picture processed in the control flow, as shown on the left side of the figure. The picture can go through a number of states such as *Initial, Reviewed, Assessed, Approved or Final*. Tasks or activities that can influence a change of the picture state are highlighted. For instance, if the *Review* task is performed, the picture should move to the *Reviewed* state or if the *Approve* task is performed, the picture may go to the *Approved* state. The benefit of knowing the content state is that the state represents a brief and concise information about what is happening in the control flow with the particular content piece. The information can be then easily broadcasted to other collaborators and help either in results marshalling or in further workflow execution. If collaborators, who have had assigned a role of *Reviewer* and are supposed to review the picture, receive a message that the picture has been already assessed and is in the 'Assessed' state, they are informed that they do not need to review the picture anymore. The depicted lifecycle shows only the basic picture states in order to illustrate the concept. The lifecycle is formed from other content states such as *Ready To Review, In Review, In Assessment, etc.*.

Context integration is shown on the right side of the figure. Firstly, context information such as *location* and *time* is needed when the picture is taken. Latitude and longitude are coordinates that are usually captured as default by most mobile devices. However, adding name of place or building is more useful. Thereby, the picture can be enriched by context metadata. Secondly, picture movement between two states can be influenced by context information. For instance, Jane can specify that only pictures with rating larger than her *Rating User Preference* can move from the *Initial* state to the *Reviewed* state. So if the *Rating User Preference* is set to 3 and the rating added to the picture is 4, then the picture can be sent for review. A similar situation is between the *Reviewed* and the *Assessed* states. The picture can move to the *Assessed* state only if a number of obtained reviews has reached Jane's requirement. Jane sets her preference for the number of reviews, *NoOfReviewsUP*, to 2, in order to determine that only two reviews are needed for her to assess the picture. Finally, the workflow process can be driven by context information. For example, reviewer can write an extra comment while reviewing the picture, but not always willing to do so. By setting the preference for comments writing, *AddCommentUP*, the decision in the control flow and the choice of the further execution path can be based on this context value. Furthermore, knowing work context and current availability of fellow co-workers would enhance the mobile collaboration. For instance, the picture can be sent only to those co-workers who are currently available to review it (*Reviewer's Availability = YES*).

The adapted workflow shows that there are numerous workflow parts that have

to be described in the MobWEL workflow process such as the adapted process control flow, the group of collaborators and their roles, the picture lifecycle, and the context variables. To outline the structure of such process, the anatomy of MobWEL workflows is described next.

4.2.2 MobWEL Workflow Process Anatomy

MobWEL workflows are designed for mobile peer-to-peer collaboration and are composed of several parts. The overall anatomy of a MobWEL workflow process is illustrated in Figure 4.2.

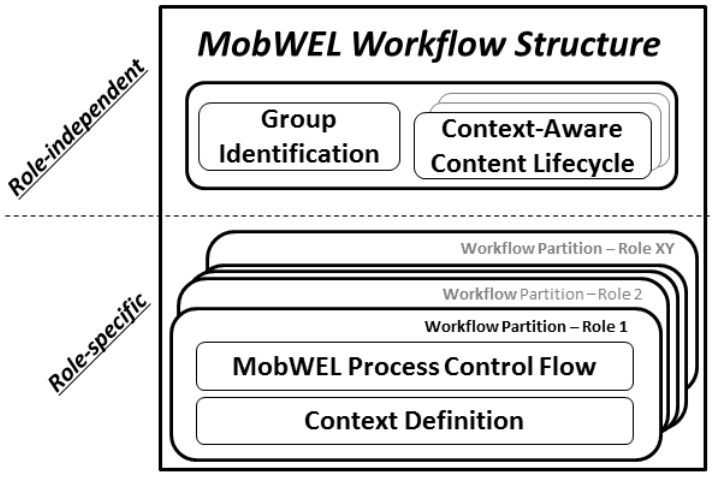

Figure 4.2: Anatomy of the MobWEL Workflow Definition

The workflow definition has *role-independent* parts and *role-specific* parts. The *role-independent* parts are defined for all participating roles, whereas the *role-specific* parts need to be explicitly defined for each participating role. The parts are described next.

Role-Independent Parts

The role-independent parts of the workflow are as follows:

Group Identification This part specifies all stakeholders participating in a workflow process and their roles.

Context-Aware Content Lifecycles A number of content objects can be processed across multiple mobile devices in a MobWEL workflow process, and each content object follows its own content lifecycle. Therefore, a set of content lifecycles can be defined. The evolution of content items is distributed across all collaborators, so the set is the same for all workflow participants.

Role-Specific Parts

The workflow MobWEL definition includes a number of role-specific parts, each part defined for a particular collaborator's role or workflow partition. The role-specific parts are as follows:

Context Definition Different context variables are related to different roles. To ensure more efficient context monitoring, context variables need to be modelled and defined extra for each workflow partition.

MobWEL Process Control Flow The activities performed by a particular role and adapted process flow have to be defined in each workflow partition.

The MobWEL workflow definition is composed of parts which are related but defined independently. The benefit of defining workflow parts independently is that *a*) the semantically related concerns are functionally separated; *b*) a workflow part can be used in multiple workflows; *c*) the evolution of each workflow part is independent from the remaining parts.

4.2.3 Overview of MobWEL Metamodel

To define MobWEL workflows and their operational instances in a consistent manner, the workflows have to conform to a workflow language metamodel. The MobWEL metamodel expresses the workflow language constructs and their relationships at a higher level of abstraction. The MobWEL workflow metamodel supports modelling of all MobWEL workflow elements, and thereby, the metamodel has the following components (Figure 4.3):

- a part for modelling adapted process control flows;
- a part for modelling a group of collaborators and their roles;
- a part for modelling context definitions; and
- a part for modelling context-aware content lifecycles.

This section has presented an anatomy of MobWEL workflows and MobWEL workflow parts. The metamodel parts are described in details in next sections.

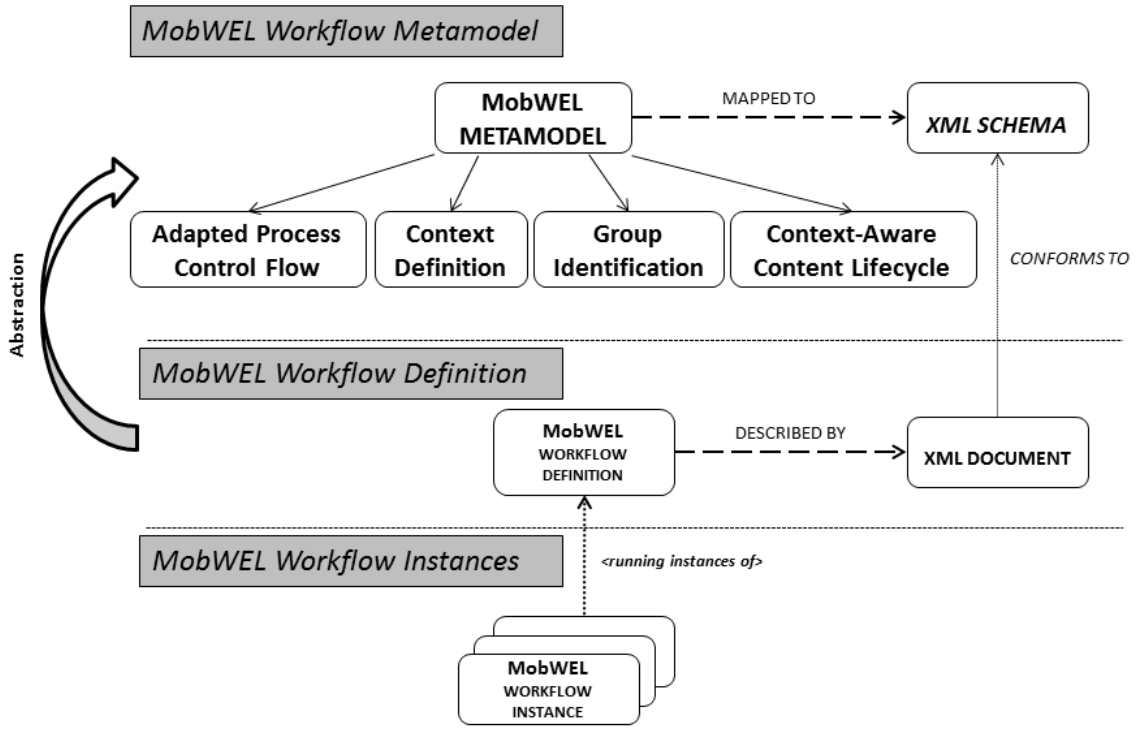

Figure 4.3: Overview of MobWEL Metamodelling

4.3 MobWEL Workflow Process Definition

The first part of the MobWEL metamodel that defines the adapted process control flow is described in this section. Firstly, the structure and the metamodel of the adapted process control flow are presented. After that, the MobWEL constructs designed for the integration of other MobWEL workflow parts are introduced. Next, the support for context and content awareness in the MobWEL process control flow is described. Finally, the MobWEL interaction model is presented.

4.3.1 MobWEL Process Control Flow Representation

BPEL is a widely accepted workflow standard language that can describe the activity-based control flow. The MobWEL workflow language extends BPEL, however, BPEL is not context-aware, does not support explicit data flows and is coupled with web service technology. Thereby, various extensions, changes and adaptations have been done in BPEL in order to make the workflow language *a*) content-aware, *b*) context-aware, and *c*) executable in a mobile peer-to-peer environment, as described throughout this section.

The brief summary of workflow constructs introduced or adopted in MobWEL is given in Table 4.1 and the metamodel of the adapted process control flow is shown in Figure 4.4.

Table 4.1: MobWEL Constructs

MobWEL Construct	Description
MobWEL Process	A workflow process identified by its unique *targetNamespace*. The *groupIdentification, contentLifecycleDescription and contextDefinition* attributes specify namespaces of group identification, content lifecycle and context definition schemas.
Variable	A variable is used in a standard process. A special type of variables is used in MobWEL workflows. The variables are used for content items processed in workflow process and the default value for type is 'content'.
Lifecycle	A lifecycle is a description of an explicitly defined content lifecycle. Each lifecycle is specified by its *name* and *resource*, a path to the explicit definition. A *variable* of a content type must be also associated with the lifecycle.
ContextDefinition	Context hierarchy is defined explicitly too. The *contextDefinitionSource* attribute specifies the path to the context definition. A workflow process can have zero or one context definition.
CollaboratorsGroup	A group of collaborators is explicitly defined and imported in the workflow definition. It is specified by its name and *groupSource*, a path to the group description. A workflow process has one group of collaborators.
Conversation	A declaration of conversation between collaborators. The construct is adopted from $BPEL^{light}$ and enables WSDL-free contract between parties.
Partner	A declaration of partners involved in collaboration. The construct is adopted from the $BPEL^{light}$ workflow definition. A workflow process can have one or more partners.
MobWEL Activity	A base type for MobWEL activities. The BPEL activity extension mechanism has been used to define the *MobWEL activity* construct. The mechanism enables to extend BPEL and introduce new activity types.
InteractionActivity	An interaction activity adopted from $BPEL^{light}$ used for all interaction activities between partners.
ContentActivity	An activity that triggers an action on a content item. It is a structured activity that can contain other activities.

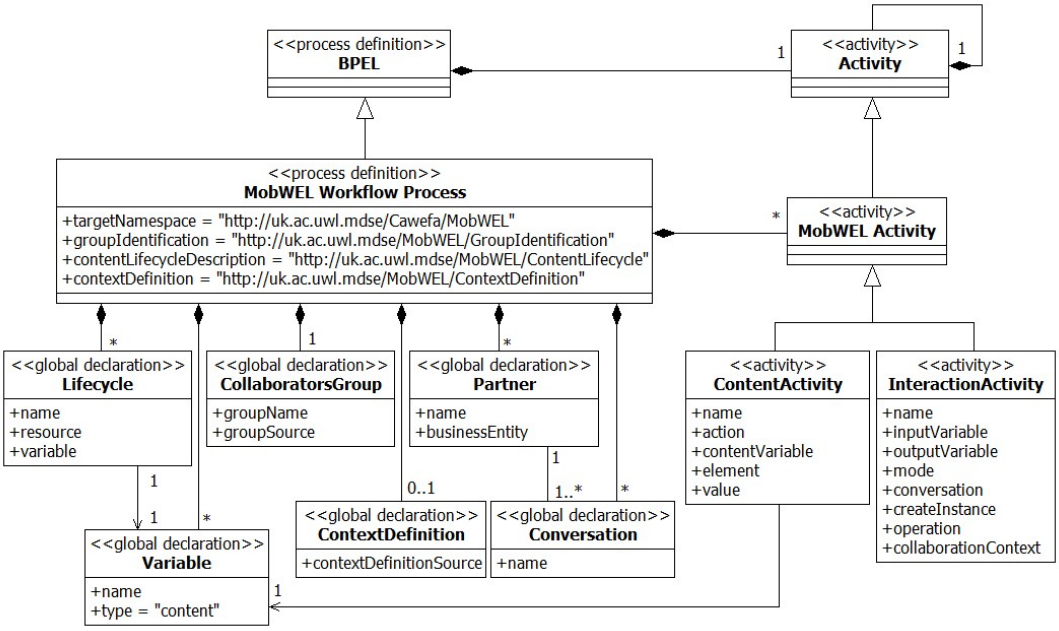

Figure 4.4: MobWEL Metamodel of Adapted Process Control Flow

The structure of the adapted BPEL process is outlined in Figure 4.5. Firstly, the Global Declarations part has been extended. Constructs for the peer-to-peer interaction and integration of other workflow elements have been added. Secondly, two extra activities, *interactionActivity* and *contentActivity*, can be used in the process definition of the control flow.

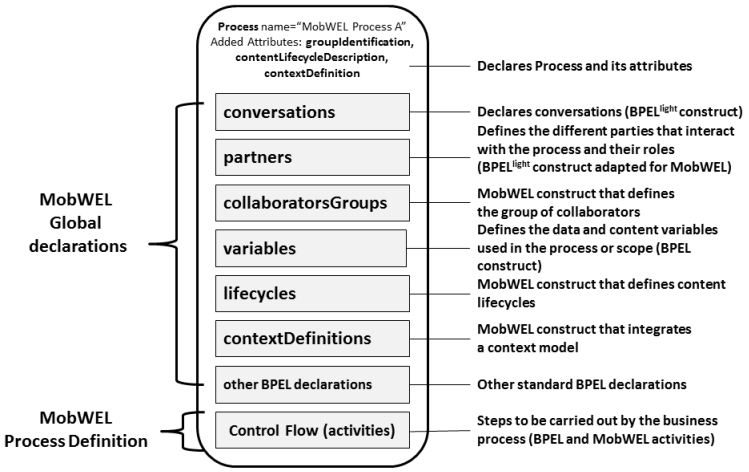

Figure 4.5: The Structure of the MobWEL Workflow Process

MobWEL constructs that extend the BPEL process definition are described in details next, starting with the global declarations part.

4.3.2 MobWEL Global Declarations

The adapted global declaration part in a MobWEL workflow process is shown in Listing 4.1. The $< process >$ element as a top-level container defines the basic properties of the MobWEL workflow. It is an extended version of the BPEL $< process >$ element. The additional attributes: *groupIdentification*, *contentLifecycleDescription* and *contextDefinition* refer to fully qualified namespaces of XML schemas developed for group identification, context-aware content lifecycle description and context definition. The workflow elements are integrated in the process definition through the *collaboratorsGroup, lifecycle, and contextDefinition* tags.

Listing 4.1: MobWEL Workflow Process - Global Declaration Part

```
<process name="NCName"
   ...
   groupIdentification="http://uk.ac.uwl.mdse/MobWEL/GroupIdentification"
   contentLifecycleDescription="http://uk.ac.uwl.mdse/MobWEL/ContentLifecycle"
   contextDefinition="http://uk.ac.uwl.mdse/MobWEL/ContextDefinition"
   standard attributes
>

<collaboratorsGroups>
  <collaboratorsGroup groupName="NCName"
        groupSource="NCName"/>
</collaboratorsGroups>

<variables>
  <variable name="NCName" type="QName"/>?
  <variable name="NCName" type="content"/>
<\variables>

<lifecycles>
  <lifecycle name="NCName"
        resource="NCName"
        variable="NCName"/>
</lifecycles>

<contextDefinitions>
  <contextDefinition contextDefinitionSource="NCName"/>
</contextDefinitions>

<other BPEL global declarations>

...control flow description...

</process>
```

MobWEL Variable

During the workflow execution, the application data are in BPEL maintained by using variables. Variables are declared in the workflow definition, initialised when the

workflow instance begins and modified usually through invoked web services. Mobile content is basically a specific type of application data, thereby variables are used for declaration of content objects. To differentiate the content variable from other variables, the type of the variable element is set to *"content"* as follows:

$$< variable\ name = "NCName" \\ type = "content" / >$$

Therefore, variables which contain content objects can be clearly identified. Not each piece of content requires to be explicitly managed and have its lifecycle visible. The specification of the variable type enables to recognise the variables which have lifecycles explicitly managed in the workflow process. All content variables are declared together with other variables in the $< variables >$ tag.

MobWEL Lifecycle

To integrate context-aware content lifecycles in the workflow process control flow, the *lifecycle* element has been created:

$$< lifecycle\ name = "NCName" \\ resource = "NCName" \\ variable = "NCName" / >$$

A lifecycle is identified by its name. The resource specifies the path to the definition file. The variable refers to a content variable which has been declared in variables. This attribute is used to highlight the content piece which behaviour is described in the lifecycle. More content variables can be associated with one lifecycle, however, for each content variable the declaration has to be made. All the $< lifecycle >$ tags are grouped in the $< lifecycles >$ tag.

MobWEL Context Definition

Workflow-specific context definition is specified for each workflow role or partition. To associate a particular context definition model with the workflow process, the $< contextDefinition >$ element is created. The element is nested within the $< contextDefinitions >$ element and multiple declarations are allowed. This enables to decompose more complex context definitions into more manageable context hierarchies and integrate all of them as if they were defined together. The element is declared as follows:

$$< contextDefinition \;\; contextDefinitionSource = "NCName" / >$$

There is no need to specify anything else, apart from the *contextDefinitionSource* path to the actual resource.

MobWEL Collaborators Group

A group of collaborators is predefined and this resource is integrated through the *collaboratorsGroup* construct as follows:

$$< collaboratorsGroup \; groupName = "NCName" \\ groupSource = "NCName" / >$$

The group has assigned a name and again, the path to the actual resource is specified. Nesting the element within the $< collaboratorsGroups >$ tag enables declaration of several groups. The advantages of this multiple declaration are *a*) possible decomposition of a large group, *b*) reusability of one group definition in a number of workflow processes.

To sum up, the elements defined in the global declarations part of a MobWEL workflow process have been created to support the integration of other MobWEL workflow parts. The next section is dedicated to the description of MobWEL constructs related to the peer-to-peer interaction model.

4.3.3 Peer-To-Peer Interaction Model

Peer-to-peer collaboration imposes requirements on functional aspects of workflows. Workflow partitions are executed on mobile devices and messages are exchanged directly between peers. BPEL supports asynchronous communication, however, strong coupling of process logic with the web services technology in BPEL is not very convenient and flexible approach to be used for mobile peer-to-peer collaboration. Workflow participants and their roles are known beforehand so a much lighter interaction model can be used. Therefore, the $BPEL^{light}$ approach is used for the interaction in MobWEL. BPEL interaction activities are replaced by the single $BPEL^{light} < interactionactivity >$. However, this $BPEL^{light}$ model has been developed for a broader range of scenarios, and thereby, is further adapted to be specific for the class of applications targeted in this work.

The elements: *partner, conversation* and *interactionActivity* have been adopted from $BPEL^{light}$ to form a peer-to-peer interaction model. Collaborators and their de-

vices are known, therefore, the communication between them can be direct without the use of web service technology. The mentioned $BPEL^{light}$ constructs have been adapted in the following way:

- **Conversation** is an element that enables WSDL-free contract between parties. It replaces the function of *partner links* in BPEL by specifying interaction between two collaborators, but without referencing the WSDL port type. The element is identified only by name and declared within the $< conversations >$ tag as follows:

  $$< conversation\ name = "NCName" / >$$

  The $< conversation >$ element has not been modified in MobWEL. Each interaction between two roles is labeled as a conversation. For example, there can be two conversations, *approvePicture and receiveApproval*, defined in the designer's workflow partition for two interactions between the role of designer and customer.

- The meaning of the $< conversation >$ element can be better understood by introducing the **Partner** element. The $< partner >$ element was originally defined in BPEL 1.1. It has been removed in BPEL 2.0 and with different meaning recreated in $BPEL^{light}$. This $BPEL^{light}$ element has been adopted in MobWEL but has been conceptually modified. The definition of the $< partner >$ element is as follows:

  $$\begin{aligned} &< partner\ name = "NCName" \\ &\qquad collaborator = "QName|all|any" > \\ &\qquad\quad businessEntity = "QName" > \\ &\quad < conversation\ name = "NCName" / > \\ &< /partner > \end{aligned}$$

  The role of the element is to group conversations with one party of collaborators group. A partner can specify either a component such as a content management unit, or define collaborator(s) who have assigned a particular role. In $BPEL^{light}$, the *businessEntity* attribute has been defined to specify the name of an organisation. In MobWEL, this attribute specifies whether the partner is a component or collaborator(s). For example,

  - If the partner is one of a component type and:

* the component is a content management unit, then
  $< partnername = "contentManager"$
  $businessEntity = "mobwel : contentManagementUnit" >$;
* the component is a context management unit, then
  $< partnername = "contextManager"$
  $businessEntity = "mobwel : contextManagementUnit" >$;
* the component is a worklist manager, then
  $< partnername = "worklist"$
  $businessEntity = "mobwel : worklistManager" / >$

– If the partner represents collaborator(s), then
$businessEntity = "mobwel : collaboratorRole"$
and the optional attribute, *collaborator*, is also specified as follows:

* There can be only a concrete collaborator specified:
  $< partner\ name = "reviewer" collaborator = "Jane"$
  $businessEntity = "mobwel : collaboratorRole" >$
  $< conversation\ name = "reviewPicture" / >$
  $< conversation\ name = "sendConfirmation" / >$
  $< /partner >$
* There can be a subgroup of all reviewers specified by setting the *collaborator* attribute to *any*:
  $< partner\ name = "reviewer" collaborator = "any"$
  $businessEntity = "mobwel : collaboratorRole" >$
  ...
  $< /partner >$
* When all reviewers are considered, the *collaborator* attribute is set to *all*:
  $< partner\ name = "reviewer" collaborator = "all"$
  $businessEntity = "mobwel : collaboratorRole" >$
  ...
  $< /partner >$

- The actual interaction between collaborators is realised through the BPEL^{light} **interaction activity**. In MobWEL, this activity has been enriched by two optional attributes, *minCollaborators* and *collaborationContext*, in order to support collaboration context, and is defined in the following way:

$$
\begin{aligned}
&< interactionActivity\ name = "NCName" \\
&\qquad inputVariable = "NCName" \\
&\qquad outputVariable = "NCName" \\
&\qquad mode = "in-out|out-in" \\
&\qquad conversation = "NCName" \\
&\qquad createInstance = "yes|no" \\
&\qquad collaborationContext = "yes|no" \\
&\qquad minCollaborators = "xs:int" \\
&\qquad standardAttributes > \\
&< /interactionActivity >
\end{aligned}
$$

The $< interactionActivity >$ covers four different interaction activities for both, synchronous and asynchronous, communication. Modelling of the interaction activity types is shown in Table 4.2.

Table 4.2: MobWEL $< interactionActivity >$ types

Activity Type	**mode**	**input variable**	**output variable**	**create instance**	**collaborationContext**	**minCollaborators**	**partner type - collaborator attribute**
Activity that only receives a message		must not	must	may			any *or* one
Activity that only sends a message		must	must not		may	may	all *or* any *or* one
Activity that first receives and then sends a message	in-out	must	must	may			any *or* one
Activity that first sends and then receives a message	out-in	must	must				any *or* one

As shown, only in the activity that sends a message, the *minCollaborators* and *collaborationContext* attributes can be specified. Consider a situation that a picture is sent for review. There is a group of ten collaborators who play the role of reviewer but only five of them are currently available to review it. If *col-*

laborationContext is set to *yes*, then the picture is sent only to the available reviewers. However, the number of available reviewers might not be satisfactory. For instance, there can be only one available reviewer. To ensure that the picture is sent to sufficient number of collaborators, the *minCollaborators* attribute has to be specified. If *minCollaborators* is set to 3, then the picture is sent to the available reviewers and if there are less than 3 available reviewers, to meet the minimum request, the picture is sent also to randomly chosen, not currently available, reviewers. The definition of such activity type is as follows:

$$
\begin{aligned}
&< conversation\ name = "reviewPicture" / > \\
&< partner\ name = "reviewer" \\
&\qquad collaborator = "any" \\
&\qquad businessEntity = "mobwel : collaboratorRole" > \\
&\qquad < conversation\ name = "reviewPicture" / > \\
&< /partner > \\
&< interactionActivity\ name = "reviewpicture" \\
&\qquad inputVariable = "message" \\
&\qquad conversation = "reviewPicture" \\
&\qquad collaborationContext = "yes" \\
&\qquad minCollaborators = "3" \\
&\qquad standardAttributes > \\
&< /interactionActivity >
\end{aligned}
$$

This section presented the MobWEL interaction model. In the next section, the MobWEL support for advanced content management functionalities is described.

4.3.4 Content Management Support

The behaviour of content variables is described in their lifecycles which are managed explicitly from the MobWEL process control flow. Questions have been raised how a content object can move through its lifecycle and the content management unit can be informed about any content manipulation during the process execution. In BPEL, the variables are not accessible from outside of the process or scope. Therefore, the need for a MobWEL workflow construct that would communicate the values of variables out has been recognised.

One possible option to manage such interaction would be by using the $< interaction\ Activity >$ construct. However, there are two drawbacks with the use of this element:

1. The element has been designed for interaction and its use is already too general and broad.

2. The process control flow is usually composed of many $< interactionActivity >$ elements, thus using the element also for this purpose would make the control flow conceptually non-transparent.

Thereby, a new MobWEL workflow construct called $< contentActivity >$ has been created. By introducing this element, interaction with the content management unit is allowed directly, and all interactions can be clearly visualised in the process flow. Another advantage of using this construct is that if management of content lifecycles is not required, all $< contentActivity >$ elements can be simply removed from the workflow definition without affecting the rest of the control flow. Simply said, the $< contentActivity >$ has an informative character and has been defined to support extra content management functionalities. The primary role of the content activity is to inform the content management unit about a content-related data change or a content object manipulation or update in order to invoke a corresponding content management operation in the content management unit.

MobWEL ContentActivity

The definition of the $< contentActivity >$ element is as follows:

$$\begin{aligned} < contentActivity\ name = "NCName" \\ action = "add|update|remove" \\ contentVariable = "NCName" \\ element = "NCName" \\ value = "NCName" / > \end{aligned}$$

The <*contentActivity*> element has the following attributes and elements:

name - represents a name of the activity;

action - specifies the content management action that needs to be performed such as add, update or remove;

contentVariable - defines the content variable that is passed in;

element - specifies the attribute of the content item;

value - contains the element value.

There are three possible actions. To illustrate each action, the following declarations need to be made, see (Listing 4.2).

Listing 4.2: Workflow Declarations

```
<variables>
  <variable name="picture" type="content"/>
  <variable name="pictureURI" type="xs:string"/>
  <variable name="pictureRating" type="xs:float"/>
</variables>

<lifecycles>
  <lifecycle name="pictureLifecycle"
        resource="pictureLifecycle.xml"
        variable="picture"/>
</lifecycles>
```

A content variable for *picture* is declared. The lifecycle of *picture* called *pictureLifecycle* is defined in the $< lifecycle >$ element. The *pictureURI* and *pictureRating* are content-related attributes, also declared as variables.

The use of $<$*contentActivity*$>$ element is shown in three different situations:

- **Add Action** When a picture is taken on a mobile device, the content management unit needs to be informed about it. So after the $< interactionActivity >$ has been performed, the $< contentActivity >$ element gets executed. The name of the activity is 'createPicture' and the action is set to 'add'. The content variable is 'picture'. And its actual location specified in the 'contentURI' variable is passed as the 'pictureURI' element or content attribute. It means that the content management unit will create and process the content object, based on the associated content lifecycle.

$$
\begin{aligned}
&< interactionActivity\ name = "takePicture" \\
&\qquad\qquad partner = "worklist" \\
&\qquad\qquad outputVariable = "pictureURI" \\
&\qquad\qquad mode = "out - in" \\
&\qquad\qquad operation = "TakePicture" / > \\
&< contentActivity\ name = "createPicture" \\
&\qquad\qquad action = "add" \\
&\qquad\qquad contentVariable = "picture" \\
&\qquad\qquad element = "contentURI" \\
&\qquad\qquad value = "pictureURI" / >
\end{aligned}
$$

- **Update Action** This action is used to update values of metadata associated with a given content item. In this example, rating has been added to picture in the interaction activity and the output of it has been stored in the *pictureRating* variable. This value is passed to the content management unit through the content activity.

$$
\begin{aligned}
&< interactionActivity\ name = "addRating" \\
&\qquad partner = "worklist" \\
&\qquad inputVariable = "pictureURI" \\
&\qquad outputVariable = "pictureRating" \\
&\qquad mode = "out - in" \\
&\qquad operation = "AddRating" / > \\
&< contentActivity\ name = "updatePictureRating" \\
&\qquad action = "update" \\
&\qquad contentVariable = "picture" \\
&\qquad element = "rating" \\
&\qquad value = "pictureRating" / >
\end{aligned}
$$

- **Remove Action** To remove the content object from the content management unit, the *remove* action is used. The example and definition of the $< contentActivity >$ element for removing the picture is as follows:

$$
\begin{aligned}
&< contentActivity\ name = "removePicture" \\
&\qquad action = "remove" \\
&\qquad contentVariable = "picture"
\end{aligned}
$$

4.3.5 Support for Context and Content Awareness

This section focuses on the support for content and context awareness within the process control flow. To make process control flow context-aware and content-aware, the MobWEL workflow process needs to have the ability to react to context changes and content state changes, and adapt its behaviour to changed environment or content behaviour. To achieve this, the MobWEL workflow process needs to be adapted for the use of context information or content state information in two ways:

- context information or information about content state can be queried when needed, especially at workflow decisions points,

- context or content state-related changes and events need to be captured and handled at any time of the workflow execution.

To address the need, the Context4BPEL concepts: context query, context decisions and context events have been adopted in MobWEL. However, the concepts are not only used for context-related information, but also for content state-related information. Therefore, the concepts have been generalised and named as *MobWEL query*, *MobWEL decision* and *MobWEL event*.

MobWEL Event is triggered when a context or content state change occurs. Both events can happen at any time, thereby, the MobWEL workflow process needs to have a mechanism to listen for, and reacts to, the events. On the other hand, information about current context situation or content state can be obtained at any time by using so called *MobWEL query* from: *a*) a context provisioning platform when context change occurs, or *b*) a content management unit when content state change occurs. *MobWEL queries* are used in *MobWEL decisions*. These three concepts are described in details next.

MobWEL Query

A condition in BPEL is usually written in the form of an XPath expression. BPEL uses some built-in functions such as the *bpel:getVariableProperty()* function. To support expression and query writing in MobWEL, several extension functions has been added, see Table 4.3. By using the *getContextValue()* and *getContentState()* functions, context values, or content states can be obtained at any decision point in the control flow. To obtain a content-related attribute from the content management unit, the *getContentAttribute* function can be used. The *getCollaboratorID* function has been designed to recognise the identity of the collaborator through the phone number. To recognise the MobWEL-specific extension functions, the prefix *'mobwel:'* is used.

Each MobWEL extension function has a signature that specifies its input parameters and exactly one output parameter. The use of the functions is illustrated throughout this chapter.

MobWEL Event

To deal with the events, **Event Handlers** can be used. Event Handler is a construct developed in BPEL to deal with events independently from the process control flow. Using event handlers allows workflow to wait asynchronously for an event notification in parallel to its process execution. Event handlers define how the workflow will

Table 4.3: MobWEL Extension Functions

Function:	**mobwel:getContextValue()**
Signature:	(String contextValue) mobwel:getContextValue(String contextName)
Description:	This function is used to query a context provider and obtain a context value of particular workflow-active context. Context is specified by its name and described in the context definition file.
Function:	**mobwel:getContentAttribute()**
Signature:	(String attributeValue) mobwel:getContentAttribute(String contentVariable, String attribute)
Description:	This function is used to obtain a content attribute value.
Function:	**mobwel:getContentState()**
Signature:	(String contentState) mobwel:getContentState(String contentVariable)
Description:	This function is used to set a state of a content item processed in the particular workflow instance.
Function:	**mobwel:getCollaboratorID()**
Signature:	(String myPhoneNumber) mobwel:getCollaboratorID()
Description:	This function is used to identify the collaborator by obtaining the mobile phone number of the device on which the particular workflow partition is executed.

deal with events. Event handlers can be defined for a scope of the whole workflow and remain active as long as the scope or workflow is active. There are two types of events handlers: *onEvent* and *onAlarm*. The former event handler is considered in MobWEL. By way of illustration, the handler can be used if a content state change occurs and this change needs to be broadcasted to peers by using $< interactionActivity >$ as demonstrated in Listing 4.3. This situation is handled independently from the workflow process execution. For example, if a picture reached the 'Assessed' state, reviewers are informed about the state and in case they have not reviewed the picture yet, the task request can be terminated. Similarly, if there is a context change in *collaboration-related context*, for instance when collaborator changed a preference and is not available to perform tasks, the notification about his unavailability needs to be sent to team workers. These actions should be handled asynchronously to the workflow process control flow. The event handler can be used to receive the context-related information (*Availability-NO*) and send the information about the context change to collaborators.

Listing 4.3: Event Handler

```
<conversations>
   <conversation name=newContentState/>
</conversations>
<partners>
   <partner name="ContentManager"
        businessEntity="mobwel:contentManagementUnit">
        <conversation name=newContentState/>
   </partner>
</partners>
<eventHandler>
   <onMessage partner="ContentManager" conversation="newContentState"
              variable="contentStateMessage">
        ...extract the content state and inform collaborators about it...
   </onMessage>
</eventHandler>
```

MobWEL Decision

To query context-related or content state-related information, and make decisions based on the obtained result, **Conditional Branching** can be used. Conditional branching in BPEL introduces decision points in the control flow.

By the way of illustration, two BPEL activities are used to demonstrate querying and the decision making process withing the control flow.

1. **Switch Activity** is an exclusive-OR activity that enables to set up one or more branches. Each branch is represented by a *case* structure that has a conditional expression and an activity. The *otherwise* clause with no condition can be used at the end to run an activity if none of the conditions in preceding cases has been evaluated to true. This construct is used to make decision based on local information related to the current context or content state as illustrated in Listing 4.4. The extension functions such as *mobwel:getContextValue()* and *mobwel:getContentState()* can be used to write queries.

Listing 4.4: Switch Activity

```
<switch>
        <case condition="mobwel:getContextValue('AddCommentUP') = 'yes' ">
                ...add extra comment to the reviewed picture...
           </case>

        <case condition="mobwel:getContextValue('AddCommentUP') = 'no' ">
                ...
          </case>

        <otherwise>
                ...
           </otherwise>
</switch>
```

2. **While Activity** is a looping construct in BPEL which has a continuation condition and a child activity. Before each iteration, the condition is evaluated. If the condition is true, the child activity is executed, if not, the loop exits. By using this construct, a certain activity can be executed as long as the context or content state information remains same or does not reach certain value. The examples are shown in Listing 4.5.

Listing 4.5: While Activity

```
Example 1:
<while condition="mobwel:getContentState('picture') != 'ReadyToAssess' ">
    <sequence>
        <interactionActivity>...</interactionActivity>
        <contentActivity>...</contentActivity>
    </sequence>
</while>

Example 2:
<while condition="mobwel:getContextValue('UPNoOfReviews') != '2' ">
        <wait for=''PT1H''/>
</while>
```

Example 1 shows a situation when the picture has been sent to a number of reviewers and the process control flow waits till a certain number of responses is obtained back. When a response is obtained through the interaction activity, the content activity is executed to inform Content Manager about content updates. If the picture moves to the *Ready To Assess* state only when three reviews have been obtained, then this loop has to be executed 3 times. After that, the continuation condition is not evaluated to true and therefore, the loop exits.

Example 2 illustrates a situation when the process flow will not continue unless a particular context state is achieved. The context value for *'UPNoOfReviews'* is basically checked every hour. If the condition is evaluated to true, the process wait for an hour and then the condition is evaluated again.

This section has described the support for content and context awareness in the MobWEL process control flow. This concludes the description of the MobWEL process control flow and its adaptation. In next sections, other workflow constituents (Group Identification, Context Definition, and Context-Aware Content Lifecycle) are presented, starting with the description of the *Group Identification* workflow part and its representation.

4.4 Representation of Collaborators Group

In previous section, the adapted process control flow, the first and main part of the MobWEL workflow, has been presented. This section describes another part of the MobWEL workflow metamodel, the representation of the collaborators group.

4.4.1 MobWEL Group Identification Metamodel

In collaborative workflows, tasks are performed by collaborators. In this concept, workflow collaborator is a person who uses a mobile device to collaborate, share content and communicate with other team members in order to achieve a common goal. To execute workflows in peer-to-peer manner, each mobile device needs to be given all relevant details about fellow collaborators and their roles. All co-workers are known beforehand, therefore, the group of collaborators can be fully pre-described and deployed on each device together with the workflow definition.

The structure of the *Group Identification* MobWEL metamodel part is shown in Figure 4.6.

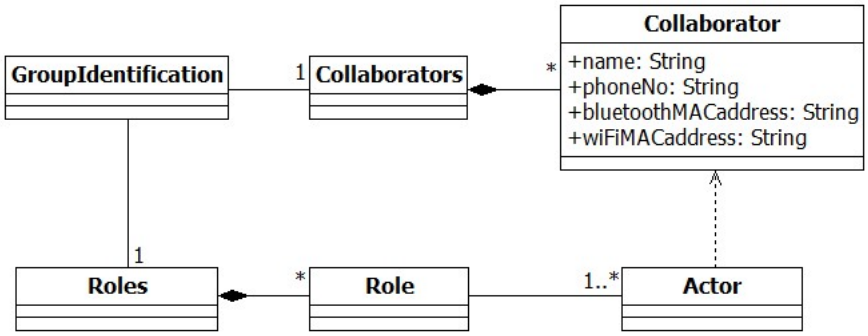

Figure 4.6: Group Identification

This MobWEL metamodel part specifies *Collaborators* and *Roles*.

The *Collaborators* element is a set of collaborators participating in a workflow. A collaborator's identity has to encompass several attributes and identifiers in order to enable interaction and messages exchange among devices. Each *Collaborator* is described by own name and mobile device which is identifiable by its phone number, Wi-Fi and Bluetooth MAC addresses. Therefore, each participating device gains all necessary information to start interaction with any other fellow peer.

The *Roles* element represent a set of roles involved in the workflow. Each *Role* is associated with a number of actors, each actor referring to a particular collaborator. The *Actor* element uses only the *phoneNo* attribute of *Collaborator*. The set of collaborators is defined separately from the set of involved roles. There are two

reasons for this. The first reason is that one collaborator can participate in multiple roles and by separating these two sets, the collaborator's details are not duplicated in the workflow definition. A collaborator can play a number of roles in various workflow instances. For instance, Jane with the role of *Interior Designer* takes a picture and creates a workflow instance. Meanwhile, another team member can take a picture that needs to be reviewed by Jane and creates another workflow instance. Jane can have a role of *Reviewer* in the coexisting workflow instance.

Secondly, the details about collaborators might already exist in the identity management unit as collaborators could have been involved in some previously deployed workflow definitions. Thus there is no need to persist the details again. Clearly, the part specifying collaborators' details can be seen basically as workflow-independent and the part describing the roles details as workflow-specific.

4.4.2 Group Identification XML Schema

MobWEL is based on BPEL, and BPEL is an XML-based language. To keep the format consistency for the full MobWEL workflow description, groups of collaborators should be also describable in the XML format, too. Thereby, the metamodel part for group identification has been mapped into an XML schema, as illustrated in Figure 4.7 and the XML documents describing groups of collaborators have to conform to this schema. See Appendix B for full listing of the *groupIdentification.xsd* XML Schema.

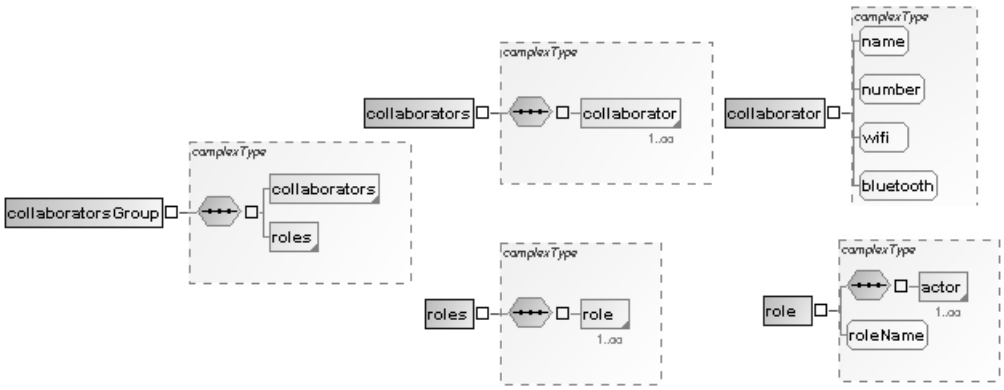

Figure 4.7: Group Identification XML Schema

This section has described the *Group Identification* MobWEL metamodel part. Next section focuses on another MobWEL metamodel part which has been designed for the definition of context variables.

4.5 Representation of Workflow-Specific Context Definition

Context modelling is supported in the MobWEL workflow metamodel and described in this section. A number of existing context management frameworks have been already described in Chapter 3, however, they are often very general, tailoring a wide spectrum of context-aware applications without awareness of the execution environment and the use of context. In the MobWEL workflow approach, the context modelling and management approach is adapted for the class of applications it is applied to.

The requirements and considerations for the definition of context models, which have to be taken into account, are well known and been summarised in the work of (Bettini et al., 2010):

Heterogeneity and mobility: Context information can be acquired from a variety of sources. For example, whilst sensors placed on mobile devices can provide numerical data that need additional interpretation, user preferences are usually comprehensible enough. In order to make the context data useful for the workflow management system, the raw context data need to be processed and provided in a concise and more uniformed format.

Relationships and dependencies: Some context data can be related or dependent on other context data. To reduce the amount of context information delivered to the workflow management system, the relationships should be depicted in the context definition model.

Reasoning: A reasoning technique to derive higher level context information can be adopted. For example, a finite set of context values can be derived for numeric raw data.

Usability of modelling formalisms: Context definition models are created by workflow designers. Thereby, real world concepts should be translated into the context definition easily. Based on the model, context information can be and manipulated by an application at run-time.

However, the listed considerations have been elicited for general and broad use. As mentioned, other aspects for the definition of context models have been considered:

Execution environment: The context is acquired and processed on a mobile device, therefore, the context model can be designed only for context data acquirable on the device.

Consumption: The main context consumer will be the MobWEL workflow management system, thus the context model has to be workflow-specific to enable easier, more convenient and efficient context consumption and workflow execution. It means that only relevant context information should be delivered to the workflow management system.

Following the considerations, a workflow-specific context definition metamodel has been constructed, as presented next.

4.5.1 MobWEL Context Definition Metamodel

Context Definition Metamodel, see Figure 4.8, expresses all the constructs, concepts and relationships between the constructs needed to build MobWEL workflow-specific context definition models.

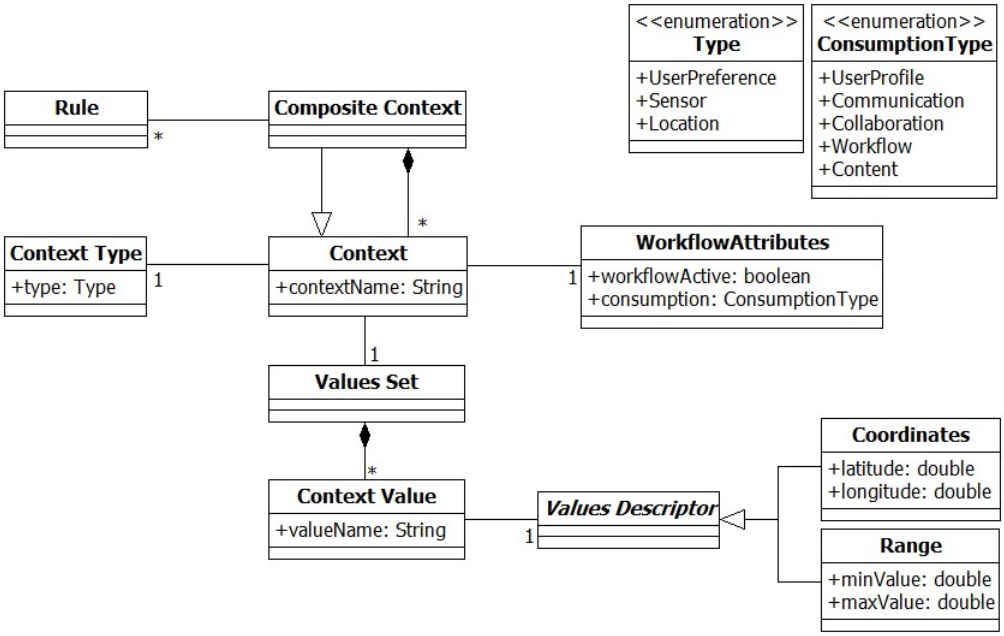

Figure 4.8: Context Definition Metamodel

Context is described by its *contextName*, ***Context Type*** and ***Values Set***. For example, context can have its name set to *Status*, its type set to *User Preference* and the context values set defined as {*Busy, Available, Not Set*}. Context data is acquired on a mobile device and ***Context Type*** contains information about the context source, which can be used as an indicator how the context data can be obtained. For example, if the context data is related to user preferences, the context type is set to *User Preference*.

High-Level Context Information

The context values set contains high-level context values. A high-level ***Context Value*** can be derived from raw context data. We use the ***Values Descriptor*** construct to associate the high-level context values with raw data. Two examples: ***Range*** and ***Coordinates*** are outlined. For instance, *LOW* as a high-level context value for battery level can be defined for a range with *minValue* set to 0 and *maxValue* set to 5 (%). Coordinates are used to specify a location or a place of interest.

Further processing of context raw data is needed to make context information meaningful and concise. There is a number of reasons why the context raw data should be processed before its dissemination to context-aware applications and workflow management systems. Firstly, the raw context data is heterogeneous and inconsistent. Secondly, if a context provisioning platform distributed the raw context data, context-aware applications would need to make the context related decisions very often. That can lead to inefficient data dissemination and worse performance of the applications. Finally, the set of context raw values might be too large or infinite, for example, in case of numeric context data.

To ensure that the consuming applications obtain only relevant and consistent context data, the abstraction level of the delivered context data is raised. The applied reasoning technique, in which high-level context information is derived from raw context data before their delivering to the applications, makes also the broadcasting frequency more adequate.

For example, battery usage can be critical for some context-aware applications. Knowing when battery level is low might influence decisions whether certain operations should be performed or not. If a consuming application requires to be informed only about the change when the battery level drops down to 5 %, constant notifications about every battery level changes would be completely inefficient. Each notification would trigger a decision-making process in the application or workflow management system. Therefore, specifying a high-level context information such as *LOW* for battery level in a range between 0-5% and *HIGH* for range 5-100% would ensure that the context-aware application is notified only when the context value is changed from *HIGH* to *LOW*.

By using the reasoning technique, the requirement regarding heterogeneity of context data is met. The technique allows to process raw context data and build concise and uniformed context information. Although, only two examples of context values derivation have been outlined on the metamodel, this approach for derivation of high-level context values is expandable and other values descriptors can be added, if need to be.

Context Aggregation

Composite context, dedicated for context aggregation, is designed as a context container. It is a subtype of ***Context*** and inherits all its attributes and behaviour. A composite context can be built as an aggregation of other composite and atomic contexts. Based on the context values of child contexts, ***Rule*** is used to determine a context value of the composite context. Therefore, the context values set can be fully defined and customised by its designer. For example, the context value of *Connectivity* can be defined as *YES* iff the current values of *DataSync* is *ON* and *Battery* is *FULL*.

Relationships and dependencies between context are identified through the accommodation of context aggregation and composition. Two examples of context aggregation are illustrated in Figure 4.9.

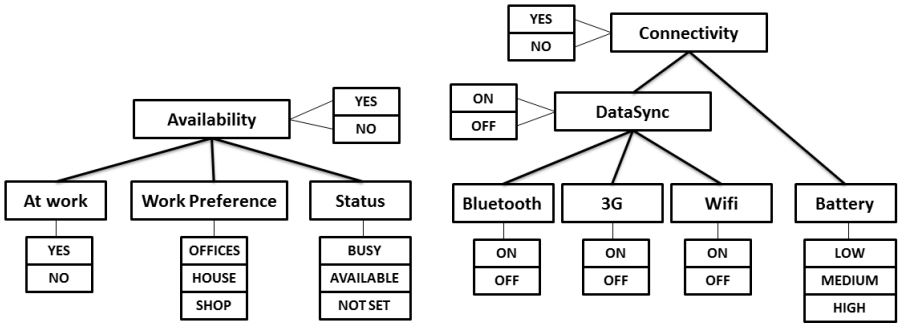

Figure 4.9: Examples of Context Aggregation

Work context of fellow interior designers is expressed as a context aggregation of: *At work, Work Preference* and *Status*. Context values for the 'At work' context are *YES* and *NO* representing the designer's work status. Each designer can specify own work preference, for illustration purposes only *OFFICES*, *HOUSES* and *SHOPS* are shown. The 'Status' is used to show whether collaborator is currently busy. *Availability* of the designer has an informative character to indicate whether the task can be taken by the person. The context value of *Availability* is determined by using context values of children and associated rules. The aggregated context value is broadcasted as follows: *(Availability-YES)* or *(Availability-NO)*.

The other example shows a context hierarchy for *Connectivity*. The *DataSync* component is composed of *Bluetooth*, *3G* and *WiFi*. The value sets for all of them are specified as $\{ON,OFF\}$. In case that any of the context is *ON*, the context value of *DataSync* is *ON* and content can be shared between devices. Content sharing might consume battery, so ideally, this operation should be performed only if battery level is not *LOW*. Thus the *Connection* component is aggregated of the

DataSync and the *Battery* contexts. The context value of *Connectivity* is set to *YES* iff *DataSync* is *ON* and *Battery* is *MEDIUM* or *FULL*. The workflow management system gets a notification only about the aggregated context information, therefore, either *(Connectivity-YES)* or *(Connectivity-NO)*.

Context Aggregation Rules

To derive the aggregated context values, a rule-based approach is used. As the rules are used to determine the aggregated context value based on child context values, a matching technique, using the Boolean operator *AND*, is applied in the rules-based strategy. The rules for the composite context, Connectivity, can be constructed by identifying all possible combinations of child context values. Aggregated context values are determined for each combination, as illustrated in Table 4.4. Each row in the table represent one rule. This approach might be a little inefficient but is simple and straightforward to be modelled and used.

Table 4.4: Rules Example

DataSync	Battery	Connectivity
ON	**LOW**	**NO**
ON	**MEDIUM**	**YES**
ON	**HIGH**	**YES**
OFF	**LOW**	**NO**
OFF	**MEDIUM**	**NO**
OFF	**HIGH**	**NO**

By enabling context aggregation, dependencies and relationships between various context data are modelled, and context tree structures and hierarchies can be built.

Workflow-Active Contexts

In addition, ***Workflow Attributes*** are used in the context modelling approach to make context consumption easier in the workflow management system. The first workflow attribute is *workflowActive*. A workflow does not need to be aware of all defined contexts. For example, in order to execute a workflow step, the workflow might need to obtain only context information: *Connectivity-YES* without knowing that *DataSync* is *ON* and *Battery* is *FULL*. Therefore, each context is associated

with the *workflowActive* attribute which can be set to *true* if the workflow is interested in listening to changes of this context, respectively to *false* if the context is only auxiliary and its values do not influence the workflow execution.

Therefore, the term of *workflow-active context* determines the context components which values are used directly in the workflow execution. In the example, only the *Connectivity* context is workflow-active and none of its children is needed to be defined as workflow-active. By specifying workflow-active contexts, the amount of context information delivered to the workflow management system is reduced.

Context Consumption

The second workflow attribute is the *consumption* attribute which is used to identify the consumption part of workflow. By knowing the value of this attribute, context routing in the workflow management system is more straightforward.

For easier context adaptation and decision making process in the workflow management system, the context use is indicated for each context. Two categories of the context use have been defined, see Figure 4.10.

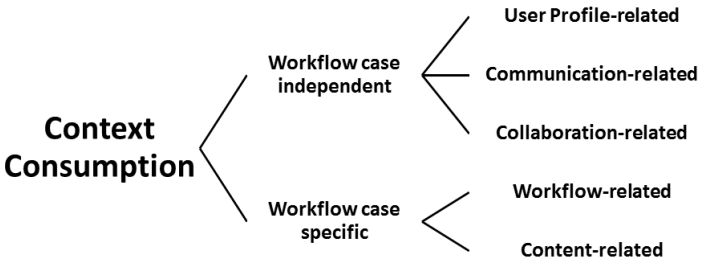

Figure 4.10: Context Use

The first category is *workflow case independent* context which is relevant to the workflow management system in a global manner. It means that the context influences operating of the whole workflow management system and is relevant for all workflow cases. This category includes three classes of context. *User Profile-related* context describes user's preferences such as preference for number of possible workflow instances that can run at a certain moment of time. This type of context is used to configure the workflow management system. *Communication-related* context is needed to determine whether the communication between devices is possible, for example, whether there is a WiFi or 3G connection. *Collaboration* related context is used for social awareness to effectively mediate the constraints of content sharing between workflow participants and determines availability of collaborators. The second *workflow case specific* category of context consumption defines context related to a workflow case. Context information can be *workflow-related* influenc-

ing the workflow execution, or *content-related* influencing the evolution of a content lifecycle.

By adding the workflow attributes to context definition models, the context modelling approach becomes workflow-specific.

The XML schema of this MobWEL metamodel part is presented next.

4.5.2 Context Definition XML Schema

The metamodel is mapped to an XML schema. The graphical view of the XML schema is shown in Figure 4.11. The full *contextDefinition.xsd* XML schema for context definition can be found in Appendix B.

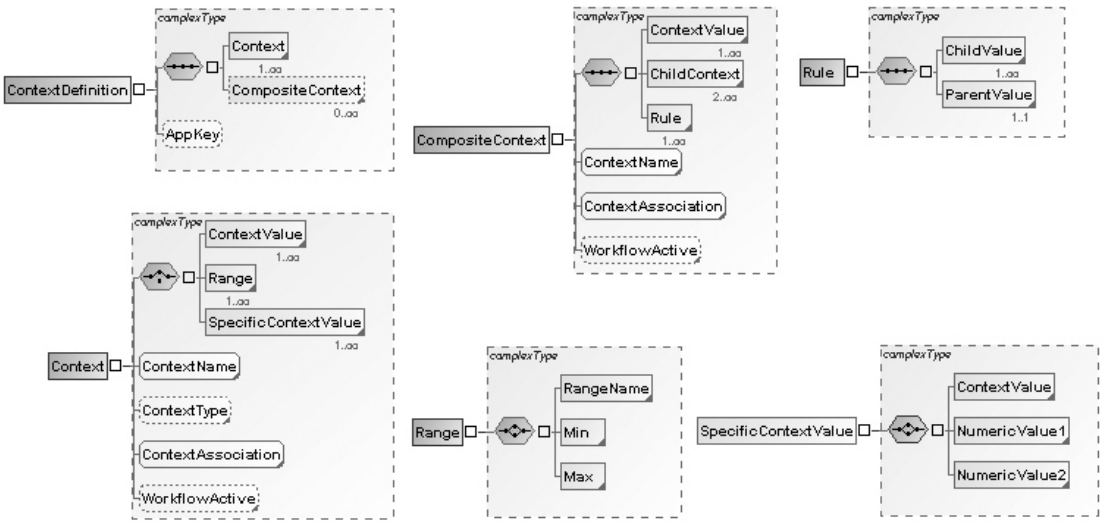

Figure 4.11: Context Definition XML Schema

The description of the MobWEL workflow-specific context modelling approach has been presented in this section. The description of *Context-Aware Content Lifecycle*, the last MobWEL workflow part, follows next.

4.6 Representation of Context-Aware Content Lifecycles

So far, three MobWEL workflow parts (process control flow, collaborators group and context definition models) have been described. This section describes the last MobWEL metamodel part, designed for the definition of context-aware content lifecycles.

4.6.1 MobWEL Context-Aware Content Lifecycle Definition

Mobile content in peer-to-peer workflow management is shared among various workflow participants with different roles. Therefore, each mobile device needs to have a system to store, manage and provide access to that content. When content is received, created or generated, it needs to be stored on the particular device. Content is manipulated by workflow activities, therefore, an easy access to content need to be provided at anytime. Content piece processed in a workflow has its own lifecycle and can go through a number of states. The states of evolution might play an important role in the workflow execution. This can be illustrated briefly by a real-life situation. For example, a buyer is interested in buying an item, but before the purchase, the buyer needs to know the state of the item, in other words whether the item is new, used or damaged. Obtaining the information about the item state enables the buyer to make the decision much easier. A similar situation happens in the workflow management world where the workflow decisions can be based on the particular information about content states. For example, if a picture has been already assessed, it can be said that it is in the *Assessed* state, and the information about its state can be broadcasted to other interior designers who will immediately know that there is no more need to review the picture. Obviously, this will be an event automatically handled by the workflow management system, not by collaborators themselves. Based on this vision, the states of single content pieces should be monitored and managed accordingly on each device.

In content lifecycles, content movements between lifecycle states are usually indicated by conditions but can also be influenced by context changes. In BPEL, there is only a little support for the recognition of a context-aware content lifecycle. Content is handled as a workflow variable and its behaviour, apart from being used as an input or output of certain activities, is hardly recognisable. In MobWEL, content is not just a workflow variable but it has its own identity. Its management is independent of the control flow execution and needs to be appropriately supported.

The other aspect of content lifecycle management envisioned here are transi-

tions between two content states. Moving of a content object to another state can be caused *a*) internally, when the values of content attributes have been changed; *b*) externally, either by an external event or by a change of external but related data. The first case of a content state change is more typical and has been considered in other works described in Chapter 3. In the second case, the external events depends on actual circumstances and the given execution environment. By following this fact, the external events need to be identified adequately for mobile peer-to-peer workflow management. As already discussed, context awareness and context events are quite significant external events that can influence running of mobile workflows. Promoting context changes in the content behaviour management can reshape and enrich its evolution, and make the whole lifecycle management approach more flexible and adaptable. Consider an example from the usage scenario about setting the user preference for the picture rating, which specifies the rating score that need to be achieved in order the picture being sent to reviewers. Thereby, to integrate context awareness into content lifecycles, transitions between content states have to be adapted for to handle this type of external context events.

To support the visions, two elements of the BALSA workflow model, the Business Artifact Information Model and the Business Artifact Lifecycle, have been adopted in MobWEL and in the modelling approach of context-aware content lifecycles. Based on that, the metamodel for context-aware content lifecycles has been constructed (Figure 4.12).

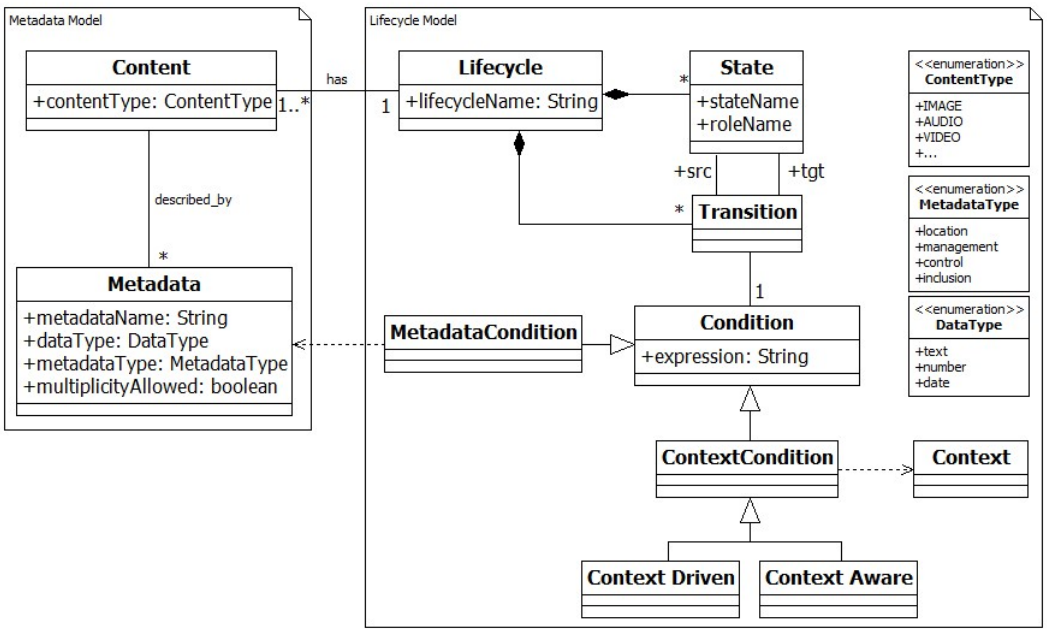

Figure 4.12: Metamodel for Content Lifecycle Definition

There are two parts, one representing content *metadata model* and another rep-

resenting content *lifecycle model*. ***Lifecycle*** is described by its *lifecycleName*. The lifecycle can describe the behaviour of several ***Content*** objects, however, in a particular definition it can be associated only with one ***Content*** object.

Content Metadata Model

The ***Content*** item is characterised by its *contentType* such as image, audio or video and a set of workflow-specific ***Metadata***. ***Metadata*** is described by its *metadataName*, *dataType:text, number, date*, *metadataType: location, management, control, inclusion*, and *multiplicityAllowed: yes, no*.

Content Metadata Model has adopted the idea of the BALSA's Business Artifact Information Model which describes the content-related data, respectively *metadata*. Boiko (2005) describes 'content management as the art of naming information' and metadata as 'the small snippets of information that are attached to content'. Mobile content has a package of default metadata. For instance, a picture taken by phone's built-in camera might be described by data such as creation date, creator or GPS coordinates. This metadata package is created and managed by the software system that actually creates or alters the picture or other content.

As mentioned, the workflow management system only coordinates the flow of the workflow activities and does not perform the actual activities. Therefore, the package of default content metadata is not accessible by the workflow management system. To gain control over a processed content piece and its evolution in workflow, a set of content-related metadata that is accessible and manageable by the workflow management system, needs to be defined beforehand. This can be briefly illustrated by an example from usage scenario. If there is a rating system created for pictures, an attribute called *'ratingScore'* should be associated with the picture to hold an information about added rating. By adding the metadata to picture, the information can be accessed at anytime and can be used to make certain management decisions.

To identify the roles and types of metadata that need to be associated with mobile content, Boiko's classification of metadata has been followed and adapted as follows (Boiko, 2005):

Location: Mobile content is often semantically enriched by context information about location. GPS coordinates are taken when content is created. For example, a picture when created is associated with information about the physical location. However, using name of the building or place might be more meaningful information. If the picture of an interior has been taken in a particular building, having the name of building associated with the picture might be useful later in

the workflow execution when, for example, the interior designer would like to see all pictures taken in the building.

Management: Content is administered through management metadata. In the workflow description, content is referred and processed only as a variable. Therefore, content needs to have also an identifier that helps to manage content. Other data such as *name, creator, creation date, modification date, and owner* might be considered as management metadata. The metadata is general, associated with almost each piece of content.

Control: Control metadata is used to guide transitions between two states and two devices. In comparison to the management metadata, the control metadata is more workflow-case specific, thereby needs to be defined specifically for a particular workflow case. The examples of control metadata are *numberOfReviews, or ratingScore*.

Inclusion: This metadata enables to reference content that physically stored in different file system. For example, on the Android platform, pictures are stored in the default file system on a given mobile device and are accessible only through an Android-specific content provider. The content provider shares data between mobile applications and it exposes a unique *URI* (Ableson et al., 2011). This *URI* is used to query data or content. Therefore, the inclusion metadata holds the information about actual content URI.

Content metadata plays an important function in content management, thereby, a set of workflow-specific content-related metadata needs to be defined for each content item.

Content Lifecycle Model

The concept of the BALSA Business Artifact Lifecycle element has been adopted in the content lifecycle model. Similarly, the context-aware content lifecycle is represented by a variant of finite state machines. Generally, a state machine contains a number of states, each state corresponding to a stage in the content lifecycle. Therefore, a *content state* is an essential construct in the content lifecycle. Content might move from one stage to another. The connections between two states are called *transitions*. When content is created, it is in an initial state with no incoming transition. At the end of its evolution, there is a final state that indicates the end of its lifecycle and has no outgoing transition. Between the initial and the final state, there are states with incoming and outgoing transitions. *Conditions* may be attached to these transitions. A condition can depend on a certain value of content attribute.

However, in MobWEL, also the aspects related to *a*) context awareness; and *b*) content evolution across multiple devices have been taken into account as follows:

- *Context Awareness:* It has been outlined that a context change can trigger a transition between two content states. Because of this, conditions placed on transitions need to be modified to deal with context. Two possible situations can happen.

  – In the first situation, the further evolution of a content piece depends on the current context in the execution environment. For example, if rating has been added to a picture, the picture moves to the *Rated* state. After that, based on the available context information, the picture can go to either the *Ready to Review* or *Ready To Archive* state, see Figure 4.13. So if *ratingScore* of the added picture is greater than or equal to the value

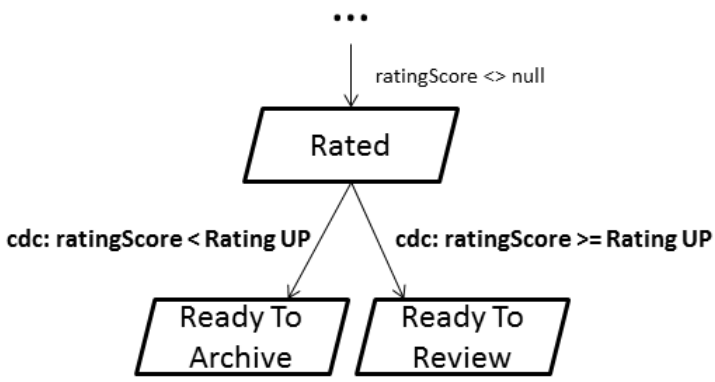

Figure 4.13: Example of Context-Driven Transitions

  of the user preference related to picture ratings, the picture goes to the *Ready to Review* state. Otherwise, it is moved to the *Ready To Archive* state. In this situation, the context information has to be obtained at the given instance of time and when it is obtained, the conditions are evaluated. The context information presented in this example is simplified, considering only *Rating User Preference* as a representant of current context situation. But as shown in the previous chapter the context information can be aggregated and context situation can be more complex.

  – In the second situation, the transition between two content states occurs only when specific context emerges. Figure 4.14 illustrates the situation. Rating added to the picture is stored in the *ratingScore* attribute. For example, it can have a value of '2'. Current *Rating User Preference* is set to '3', therefore the picture cannot move to the *Ready to Review* state

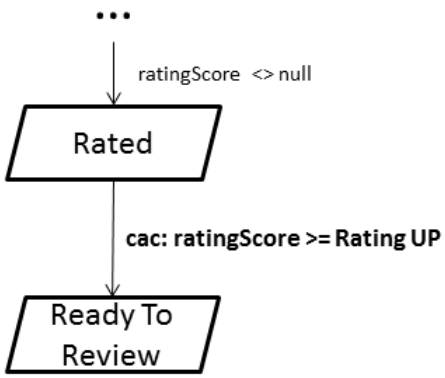

Figure 4.14: Example of Context-Aware Transition

at this time. But it waits whether any context change in the *Rating User Preference* happens. The picture remains in the *Rated* state until the *Rating User Preference* is changed and becomes less than or equal to '2'. In contrast to the first situation, this situation requires an awaiting, monitoring and filtering mechanism for context events.

To distinguish between these situations, two types of context conditions are created: *context-driven condition (CDC)* and *context-aware condition (CAC)*.

- *Content evolution across multiple devices:* Content is shared among collaborators and devices, and its lifecycle evolves on numerous devices. Thereby this requires some extra considerations how to deal with the distributed behaviour. One of the options how to deal with the content evolution on multiple devices would be to partition the content lifecycle in a similar way as distributed workflows are often partitioned. However, by using this method, the overall behaviour of content would be hardly visible. Thus a more pragmatic way is considered that allows the whole content lifecycle to be described in one piece. The technique outlines the 'ownership' of content states. In other words, each content state is associated with a name of the role which is accountable for the corresponding workflow steps. For example, if the task of rating picture is assigned to the role of designer, the *Rated* state is owned by the role.

Thereby, in the MobWEL metamodel for definition of context-aware content lifecycles, ***Lifecycle*** is composed of a number of ***State***s and ***Transition***s. A ***State*** is identified by its *stateName* and the *roleName* attribute which specifies the role accountable for the state. Each ***Transition*** has a source and target ***State*** and can be associated with a ***Condition***. To trigger the transition of the content item between the source and target state, the associated condition must evaluate to true. There are two subtypes of conditions, ***MetadataCondition*** and ***ContextCondition***.

Fulfillment of ***MetadataCondition*** depends on the value of particular metadata. ***ContextCondition*** depends on certain context and can be *ContextAware* or *ContextDriven*. The fulfillment of the *Context Aware* condition depends on particular context. The transition is triggered when the expected context emerges. The fulfillment of the *Context Driven* condition depends on current context. So it requires immediate context querying. Based on the returned context value, the transition is evaluated. Only if the condition is evaluated to true, the transition is triggered.

The metamodel represents an abstract form of a context-aware content lifecycle and has been mapped into an XML Schema which is shown next.

4.6.2 Context-Aware Content Lifecycle XML Schema

The metamodel has been mapped into the XML schema, see Appendix B for the full listing of the *contentLifecycle.xsd* XML Schema. The graphical view of the XML schema is shown in Figure 4.15.

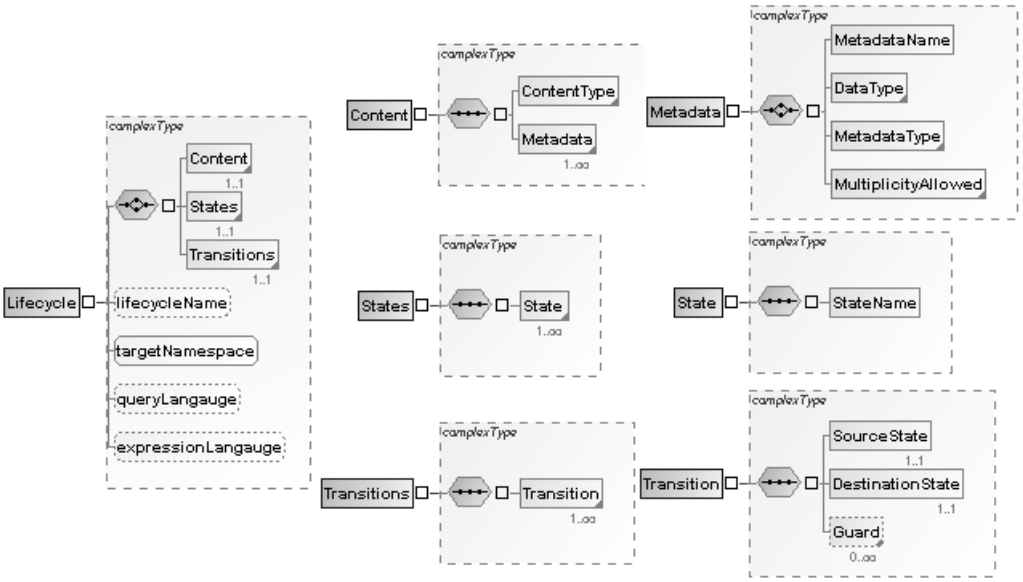

Figure 4.15: XML Schema for Content Lifecycle

Expression Building

The conditions placed on transitions are characterised by *expression*s which are built in the precise format. The expression building is shown in Table 4.5.

An expression has a prefix that determines the type of condition, a left side and a right side. MobWEL extension functions can be used to build the left and/or right side of expressions. The left side and the right side of an expression are joined by using a comparison operators such as:

Table 4.5: Expression Builder

Condition	**Prefix**	**Left Side**	**Right Side**
Context-Aware condition	**cac:**	getContextValue()	concrete context value
Context-Driven condition	**cdc:**	getContextValue()	concrete context value
Attribute-Related condition	**def:**	getContentAttribute()	concrete context value
Mixed condition (cac+def)	**cac:**	getContextValue()	getContentAttribute()
Mixed condition (cdc+def)	**cdc:**	getContextValue()	getContentAttribute()

- *equal to* $(=, eq)$;
- *not equal to* $(!=, ne)$;
- *greater than* $(>, gt)$;
- *less than* $(<, lt)$;
- *greater than or equal to* $(\geq, ge)$;
- *less than or equal to* $(\leq, le)$.

The symbols for the comparison operators are used in figures and in text, whereas the two-letter abbreviations are used to describe conditions in XML documents.

For example, this context-aware condition:

$$ratingScore \geq RatingUP$$

would be expressed as:

$$cac{:}getContentAttribute(ratingScore) \geq getContextValue(RatingUP)$$

and in the XML description as:

$$cac{:}getContentAttribute(ratingScore)\ \&ge\ getContextValue(RatingUP)$$

In this section, the metamodel for the definition of context-aware content lifecycles has been presented as the last component of the MobWEL workflow language metamodel. This section concludes the definition of the MobWEL workflow language. A brief chapter summary is given next.

4.7 Summary

One of the objectives of the thesis was to design a workflow language that can be used to represent workflows for a certain class of mobile applications. This chapter has introduced MobWEL, a context-aware content-centric workflow language designed for mobile peer-to-peer collaboration. MobWEL extends BPEL, using constructs from existing workflow approaches, Context4BPEL and BPELlight, and adopting elements from the BALSA workflow model. In this chapter, four MobWEL metamodel parts have been presented which define MobWEL process control flow, MobWEL collaborators group, MobWEL context model, and MobWEL context-aware content lifecycles. Additionally, the MobWEL XML-based format and MobWEL syntax have been described.

Formal semantics of MobWEL workflow language is elaborated in next chapter.

Chapter 5

MobWEL Semantics

Contents

5.1 Introduction

The metamodel and syntax of the MobWEL workflow language has been presented in Chapter 4. Many MobWEL constructs have been adopted from existing workflow approaches, therefore, most of MobWEL structural and operational semantics is inherited. However, several concepts have been introduced in MobWEL workflow schema that influenced the way how MobWEL workflow processes are executed. Thereby, this chapter completes the definition of the MobWEL language by providing its semantics. This semantics is used in the validation part for the description of the expected behaviour of MobWEL workflow instances.

The remainder of this chapter is organised as follows. Firstly, the definition for context situation is given in Section 5.2. After that, semantics for MobWEL control flow is defined in Section 5.3. Section 5.4 extends the semantics with data flow. Semantics for content behaviour is elaborated in Section 5.5. Definitions for consistency between process flow and content lifecycles are given in Section 5.6. This chapter is summarised in Section 5.7.

5.2 Context Situation

The workflow execution is adapted to the actual context situations in which the mobile devices reside at that time. The context situation depends on workflow-active context components that are defined for each workflow partition. Therefore, the context situation on a mobile device is defined as follow:

Definition 5.1 (**Context Situation**). *Let Ctx =* $\{ctx_1, ctx_2, ..., ctx_k\}$ *be a set of workflow-active context components defined in a context definition model M, and let each component* Ctx_i, $i \in \{1, .., k\}$, *be associated with a set of context values* $\{cv_{i1}, ..., cv_{ij}\}$, $j \in \mathbb{N}$. *A context situation c is a k-tuple of current context values of all context components* $< (ctx_1, cv_{1x}), (ctx_2, cv_{2y}), ..., (ctx_k, cv_{kz}) >$ *where each context value* cv_{ij} *belongs to the values set of the corresponding context component* Ctx_i.
A context situation in which only a particular context value of a particular context component is important is labeled as c/ctx_i, $c/ctx_i =< (ctx_i, cv_{ij}) >$.

By the way of illustration, let us consider that there are three workflow-active context components defined in the context definition model: *RatingUP* (values set={1,2,3}), *Connectivity* (values set={YES,NO}), and *Location* (values set={BuildingA, BuildingB}). If current context values are: '1' for *RatingUP*, 'YES' for *Connectivity* and 'BuildingB' for *Location*, then the current context situation c is

$$c =< (RatingUP, 1), (Connectivity, YES), (Location, BuildingB) > .$$

5.3 Control Flow Semantics

This section introduces MobWEL control-flow semantics. Some definitions have been adapted from the work of Wahler (2009).

Firstly, the definition of a MobWEL workflow process is provided.

Definition 5.2 (**MobWEL process flow**). *A MobWEL process flow is a directed graph G = (N,E), where a node* $n \in N$ *is one of the following: a start node, a stop node, an activity, a fork, a join, a decision, or a merge. The functions* in, out *: N* $\rightarrow \rho(E)$ *map a node to the set of its incoming and outgoing edges. The following conditions hold with respect to a MobWEL process flow G = (N,E):*

- *there is exactly one start node and exactly one stop node in N;*
- *the start node has no incoming edges and exactly one outgoing edge, whereas the stop node has exactly one incoming edge but no outgoing edges;*

- *each fork and each decision has exactly one incoming edge and two or more outgoing edges, whereas each join and each merge has exactly one outgoing edge and two or more incoming edges;*
- *each action has exactly one incoming and exactly one outgoing edge;*
- *each node $n \in N$ is on a path from the start node to the stop node.*

According to Aalst et al. (2003), a join, also known as AND-join, is different from a merge in that the join is a point in the workflow process where multiple parallel subprocesses converge into a one single thread of control. A merge, also known as a XOR-join or asynchronous join, is a point where two or more alternative branches come together without synchronisation.

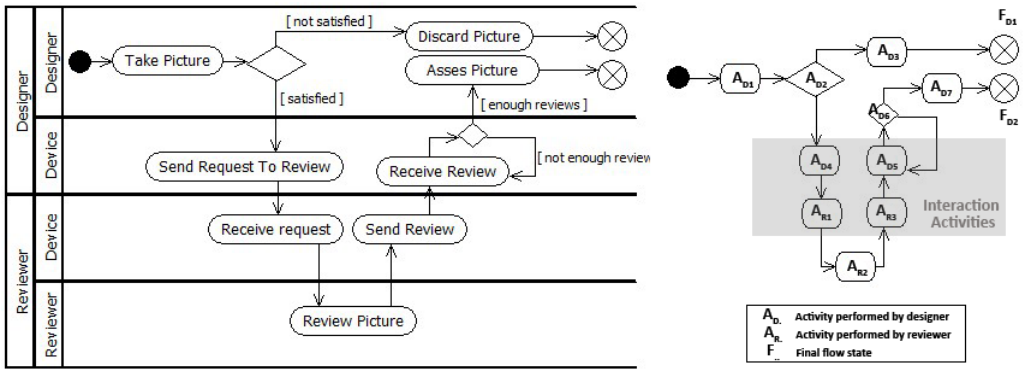

Figure 5.1: MobWEL Process Flow as a Graph

Consider a workflow model depicted in Figure 5.1. This model is mapped into a graph in which all activities are nodes and flows between activities are edges. For conciseness, the activities are labeled. Labeling enables to highlight the performer's role such as $A_{D.}$ is an activity performed by designer, and $A_{R.}$ is an activity performed by reviewer. The final flow states are labeled with the letter 'F' and interaction activities are highlighted in the grey area.

An execution state of a MobWEL process flow is represented by the distribution of tokens.

Definition 5.3 (**MobWEL Process Flow Execution state**). *Given a MobWEL process flow G = (N,E), its execution state is a mapping $\omega : E \to \mathbb{N}$ that assigns natural numbers to all edges in G. The number assigned to an edge in an execution state ω represents the number of tokens carried by that edge in ω.*

The definition for control-flow semantics has been adapted for context situation. It means that a change of the execution state happens in a particular context situation. In the particular context situation the semantics of the various nodes is defined as follows:

- One token is removed from each of its incoming edges and one token is added to each of its outgoing edges when the node is an activity, a fork, or a join. However, if the activity is an interaction activity, this rule is altered. In case of sending a request to x actors with the same role, one token is removed from its incoming edge and x token is added to its outgoing edge. In case of receiving a reply from x actors with the same role, only one token is removed from its incoming edge at the time and one token is added to its outgoing edge.
- One token is removed from its incoming edge and one token is added to one of its outgoing edges when the node is a decision
- One token is removed from one of its incoming edges and added to its outgoing edges when the node is a merge.

As the workflow graph abstract from the decision logic, in last two cases the edge is chosen nondeterministically.

Definition 5.4 (**Context-aware MobWEL Control-flow semantics**). *Let ω and ω' be two execution states of a MobWEL Process Flow G = (N,E), $c \in C$ be a context situation, and $n \in N$ be a node that is neither a start nor a stop node. The change of the execution state from ω to ω' by the execution of node n in the context situation c is written as $\omega \xrightarrow{n,c} \omega'$. The following four rules define $\omega \xrightarrow{n,c} \omega'$ for different types of nodes:*

1. *n is an activity, interaction activity (IN), fork, or join and*

$$\omega'(e) = \begin{cases} \omega(e) - 1 & \textit{for} \quad e \in in(n), \\ \omega(e) + 1 & \textit{for} \quad e \in out(n), \\ \omega(e) & otherwise. \end{cases}$$

2. *n is an interaction activity (OUT)*

$$\omega'(e) = \begin{cases} \omega(e) - 1 & \textit{for} \quad e \in in(n), \\ \omega(e) + x & \textit{for} \quad e \in out(n), x \; is \; number \; of \; actors \\ \omega(e) & otherwise. \end{cases}$$

3. *n is a decision and there exists an outgoing edge $e' \in out(n)$ of n such that*

$$\omega'(e) = \begin{cases} \omega(e) - 1 & \textit{for} \quad e \in in(n), \\ \omega(e) + 1 & \textit{for} \quad e = e', \\ \omega(e) & otherwise. \end{cases}$$

4. n is a merge and there exists an incoming edge $e' \in in(n)$ *of n such that*

$$\omega'(e) = \begin{cases} \omega(e) - 1 & \textit{for} \quad e = e', \\ \omega(e) + 1 & \textit{for} \quad e \in out(n), \\ \omega(e) & otherwise. \end{cases}$$

Definition 5.5 (**Initial and terminal execution states**). *Given a workflow graph G, its initial execution state is the execution state* ω_i *that has exactly one token on the outgoing edge of the start node and no tokens elsewhere. The terminal execution state of G is the state* ω_t *that has exactly one token on the incoming edge of the stop node and no tokens elsewhere.*

To validate the workflow execution, other related concepts are defined.

Definition 5.6 (**Activated node, execution sequence**). *A node n is said to be activated in an execution state* ω *if there exists another execution state* ω' *and context situation c such that* $\omega \xrightarrow{n,c} \omega'$.
A sequence of node executions $\omega_0 \xrightarrow{n_1, c_1} \omega_1 ... \omega_{k-1} \xrightarrow{n_k, c_k} \omega_k$ *is called an execution sequence.*

There can be more acceptable context situations that enables the activation of a node.

Definition 5.7 (**Acceptable context situations for an activated node**). *Let G=(N,E) be a workflow graph and* $C = \{c_1, c_2, ..., c_k\}$ *be a set of all possible context situations that can occur while G is executed. Let* $n \in N$ *be a node,* ω *and* ω' *be two execution states of G.*
Acceptable context situations for a node n are such context situations that enable the change of the execution state from ω *to* ω' *by the execution of the node n are*

$$\forall c_i \in C_n, C_n \subset C, \omega \xrightarrow{n, c_i} \omega'$$

This is also denoted as $\omega \xrightarrow{n, C_n} \omega'$.
If $C_n = C$*, then the change of the execution state is not dependent on the context situation and all context situations are acceptable. It means that the execution of the node n is context-independent and this is denoted as*

$$\omega \xrightarrow{n, \widetilde{c}} \omega'.$$

Definition 5.8 (**Reachable execution state**). *An execution state* ω' *is reachable from an execution state* ω*, denoted* $\omega \xrightarrow{*} \omega'$*, if there exists a finite execution sequence* $\omega_0 \xrightarrow{n_1, c_1} \omega_1 ... \omega_{k-1} \xrightarrow{n_k, c_k} \omega_k$ *such that* $\omega_0 = \omega$ *and* $\omega_k = \omega'$.

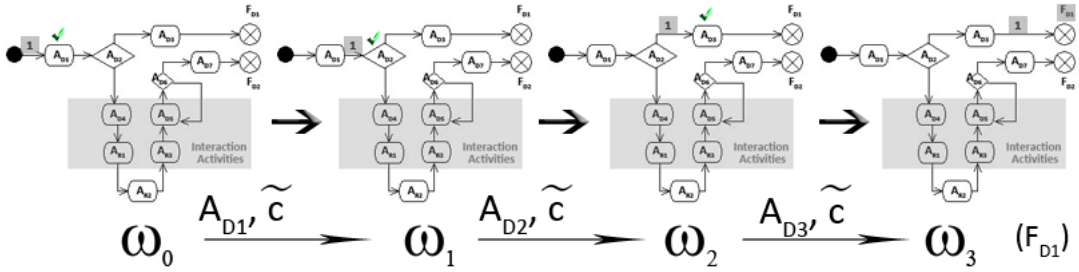

Figure 5.2: Example of MobWEL Control Flow Execution

To illustrate the application of the MobWEL process flow semantics, Figure 5.2 shows an example of the execution sequence in the workflow model, starting with ω_0, and reaching the ω_3 execution state and the final flow state F_{D1}. The nodes activated in the execution sequence are: A_{D1}, A_{D2}, A_{D3}, and the changes of execution states are context independent. In the workflow model, the tokens are associated with edges and highlighted in grey squares and the activated nodes, respectively activities are marked with a tick.

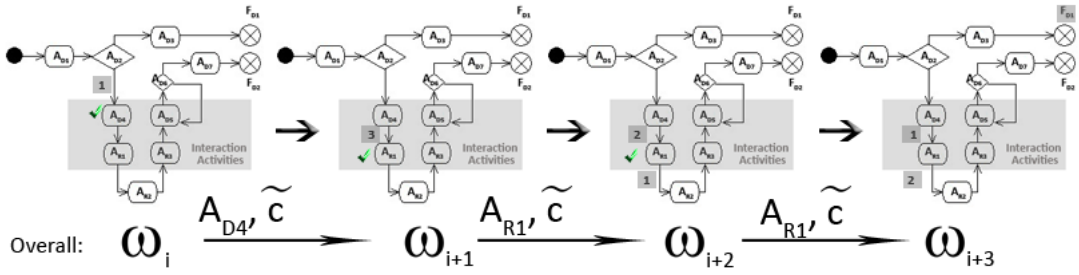

Figure 5.3: Example of Execution of an Interaction Activity - Whole Workflow

Figure 5.3 illustrates the change of the execution state by activating an interaction activity A_{D4} when a request to review the picture is sent to three different actors who play the role of reviewer. Therefore, 3 tokens are associated with the edge. In the next step, when one of the reviewers receives the request, the activity A_{R1} is executed which causes the change of the overall execution state. Then the activity is activated again on another actor's device. So overall, the execution sequence is triggered: $\omega_i \xrightarrow{*} \omega_{i+3}$.

However, as the workflow case is executed on multiple mobile devices and by various actors, the execution sequence is defined for all participating actors independently. From the point of view of the actor with the designer role D_1, after the request to review the picture has been sent to three reviewers, the flow continues towards the interaction activity A_{D5} (Figure 5.4a). The change of the execution state triggered by activating the interaction activity A_{R1} on the device of the reviewer R_1 and the reviewer R_2 is shown in Figure 5.4b. In the case of the reviewer R_3, the

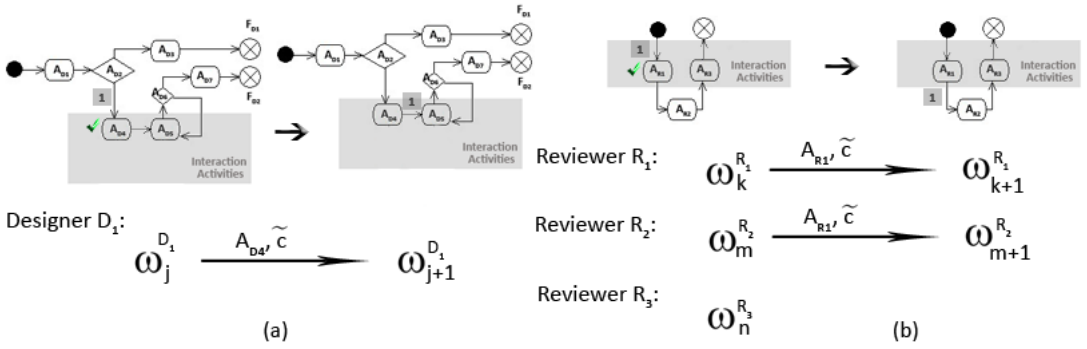

Figure 5.4: Example of Execution of an Interaction Activity - Individual Actors

activity has not been activated yet.

As demonstrated, the control flow semantics is used to derive the behaviour of individual workflow partitions and the overall behavior of the whole workflow instance. However, to simplify the description of the execution sequence on the workflow model, the notation shown in Figure 5.5 is used. There are two execution paths highlighted. An execution path is highlighted by *a*) placing tokens on flows (number of tokens corresponds to the overall count of the particular edge invocation); and *b*) marking activated activities with ticks and using grey squares.

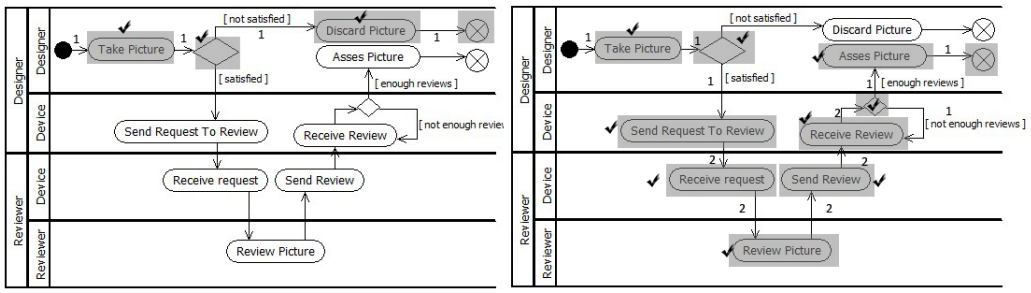

Figure 5.5: Examples of Executions Paths Depicted on Model

In the following section, the existing control-flow semantics presented in this chapter is extended by the data flow.

5.4 Extended Semantics with Data Flow

The MobWEL workflow language allows to visualise content flow explicitly, however, the content flow is also embedded within the control flow and content-related data are used as input or output data of several workflow activities. Explicit content flow is referred as content lifecycle. Implicit content-related data flow assumes the definition of workflow process variables which are accessible in the read/write mode by workflow activities.

Content-related data in a workflow model form a subset of data in the model. To monitor implicit content-related data flow in a given workflow graph, the approach described in the work of (Wahler, 2009) has been adapted, and input and output data are added to the representation of a node n and its edges. This is formalised in the following definition:

Definition 5.9 (**Implicit data flow**)**.** *Given a workflow graph G = (N,E) and a set of process variables V with a values set D, a data flow for G is defined using two functions dataIN, dataOUT:* $N \rightarrow \rho(D)$ *that map a node to sets of the values for data variables available before and after the execution of node n. A partial function type:* $E \rightarrow D$ *is defined to map an edge* $e \in E$ *to the set of the values D for data variables. Given a node* $n \in N$*, the following conditions hold:*

- *dataIN(n) and dataOUT(n) are empty if n is not an activity or a decision;*
- *all incoming and outcoming edges of* n *are mapped to the same set of the values for data variable, dataIN(n) = dataOUT(n), if* n *is a decision, a merge, a fork or a join.*

Based on the definition, the notation for the data flow is defined, as shown in Figure 5.6. In this example, $dataIN(A) = d_1$ and $dataOUT(A) = d_2$ if A is an activity, and $dataIN(D) = dataOUT(D) = d$ if D is a decision.

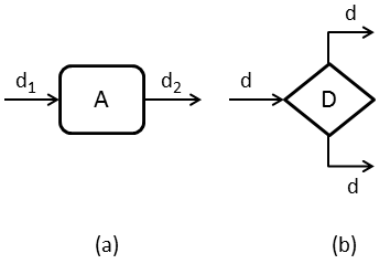

Figure 5.6: Notation for Data Flow

As the interest is focused particularly on decisions based on context information or content state information, the notation for these types of decisions is shown Figure 5.7. A decision based on context information is labeled as D_{Ctx}, whereas the label for a decision based on content state information is D_{CSt}.

5.5 Content Behaviour

Content lifecycle is defined explicitly and content states are the attributes of content objects that are managed independently from the process control flow. In this section, definitions for content behaviour management are introduced.

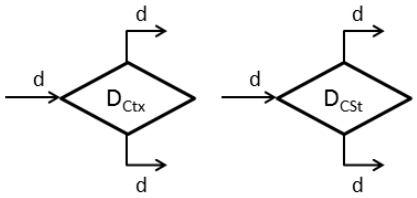

Figure 5.7: Notation for Decisions

Definition 5.10 (**Fired Transition, Transition Firing Sequence**)**.** *Let L be a context-aware content lifecycle for a content object O. A transition t with a source state s in the lifecycle L is said to be fired if condition(guard) g placed on the transition t is evaluated to true and there exists a content state s' such that* $s \xrightarrow{g} s'$*.*
A sequence of fired transitions $s_0 \xrightarrow{g_1} s_1 .....s_{k-1} \xrightarrow{g_k} s_k$ *is called a transition firing sequence.*

Definition 5.11 (**Reachable Content State**)**.** *A content state* s' *is reachable from a content state* s*, denoted* $s \xrightarrow{*} s'$*, if there exists a finite transition firing sequence* $s_0 \xrightarrow{g_1} s_1 ... s_{k-1} \xrightarrow{g_k} s_k$ *such that* $s_0 = s$ *and* $s_k = s'$*.*

5.6 Consistency of Process Flow and Content Lifecycle

This section is dedicated to the consistency between process flow and content lifecycles. Content lifecycles are managed independently from the main process flow logic. Thus as the process flow is executed and the content object is manipulated through various activities, the associated content object lifecycle should evolve accordingly. To achieve that, MobWEL workflows should be well-formed. Workflows are well-formed if there are no dangling activities or tasks. In well-formed workflows, every part of workflow contributes to the result of a run and all workflow roles and declared input data contribute to achieve the expected workflow products.

The content object is in a content state at any given time of the process execution.

Definition 5.12 (**Process execution state with corresponding content state**)**.** *Given a workflow graph G = (N,E) and an integrated context-aware content lifecycle L for a content object O, an execution state of G is represented by* (ω, s, O)*, where* ω *is the mapping defined in Definition 7.3, O is the manipulated object, and* s *is the current content state of the content object* O*. Thus* (ω, s, O) *represents a process execution state with corresponding content state* s *for an object O.*

Therefore, the reachability of a certain execution state is defined as follows:

Definition 5.13 (**Reachable process execution state with corresponding content state**). *A process execution state ω' with corresponding content state s' for an object O is reachable from an execution state ω with corresponding content state s, denoted $(\omega, s, O) \overset{*}{\mapsto} (\omega', s', O)$, if and only if there exists a finite transition firing sequence $s_0 \xrightarrow{g_1} s_1 .....s_{j-1} \xrightarrow{g_j} s_j$ such that $s_0 = s$ and $s_j = s'$, and also if and only if there exists a finite execution sequence $\omega_0 \xrightarrow{n_1, c_1} \omega_1 ... \omega_{k-1} \xrightarrow{n_k, c_k} \omega_k$ such that $\omega_0 = \omega$ and $\omega_k = \omega'$.*

Finally, the definition for the consistency between these two workflow assets is introduced.

Definition 5.14 (**Consistency of Process Flow and Content Lifeycle**). *Let Q = $\{ < \omega_{11} \overset{*}{\rightarrow} \omega_{1p} >,...,< \omega_{n1} \overset{*}{\rightarrow} \omega_{nq} >\}$ be a set of execution sequences generated by all possible executions of a well-formed MobWEL process flow F, let X = $\{ < s_{11} \overset{*}{\rightarrow} s_{1m} >,...,< s_{p1} \overset{*}{\rightarrow} s_{pn} >\}$ be a set of all possible transition firing sequences within a context-aware content lifecycle L for a content object O, and let Z = $\{(\omega_{12}, s_{p2}, O)$, (ω_{1p}, s_{p2}, O),...,$(\omega_{nq}, s_{1m}, O)\}$ be a set of all possible process execution states with corresponding content states. The process flow F and the lifecycle L are consistent if and only if the following conditions are satisfied:*

- *for every execution sequence q = $< \omega \overset{*}{\rightarrow} \omega' >$ in Q, there exist a transition firing sequence z = $< S \overset{*}{\rightarrow} S' >$ in Z such that $(\omega, S, O) \overset{*}{\rightarrow} (\omega', S', O)$;*
- $\forall z_1, z_2 \in Z, z_1 = (\omega, s, O), z_2 = (\omega', s', O)$ *(for which $\exists q \in Q, q =< \omega \overset{*}{\rightarrow} \omega' >$, and $\exists x \in X, x =< s \overset{*}{\rightarrow} s' >$) the following implication must be met:*
  If q is executed, then x is fired and the content state s' is reached.

5.7 Summary

This chapter has introduced the semantics of the MobWEL workflow language. With this, the definition of the MobWEL workflow language is completed. The workflow management system that is capable of managing and executing such MobWEL workflows is described in next chapter.

Chapter 6

MobWEL Workflow Management and Execution

Contents

6.1 Introduction

The previous two chapters defined the syntax and semantics of the MobWEL workflow language. The MobWEL language can be used to define MobWEL workflows. Management and execution of MobWEL workflows on mobile devices need to be supported by a suitable workflow management software that also has capabilities to support context and content management, and operates in a distributed manner. This chapter presents the MobWEL workflow management system, a system capable of managing and executing MobWEL workflows.

The rest of this chapter is organised as follows. The logical architecture of the MobWEL workflow management system and the parsing process of MobWEL work-

flows are presented in Section 6.2. After that, individual system components are presented. Section 6.3 presents the Context Provider, a component designed for context acquisition. Context routing and consumption is controlled by the Context Manager component, presented in Section 6.4. The advanced content management functionalities are provided by the Content Manager component, described in Section 6.5. The MobWEL engine, the main execution unit, is presented in Section 6.6. Finally, Peer-to-peer Interaction Manager is described in Section 6.7. Components interactions are outlined in Section 6.8. This chapter is concluded in Section 6.9.

6.2 Architecture of MobWEL Workflow Management System

This section describes the high-level architecture of the MobWEL workflow management system.

MobWEL workflows are based on BPEL, therefore, in the center of the MobWEL workflow management system, a BPEL engine is operating. Because of the complexity of the workflow management solution, the semantically related data and functions have been grouped together into smaller, more manageable functional parts called components. Each internal component has a specific functionality designed for management of one particular MobWEL workflow part. The high-level architecture of the MobWEL workflow management and execution system is shown in Figure 6.1.

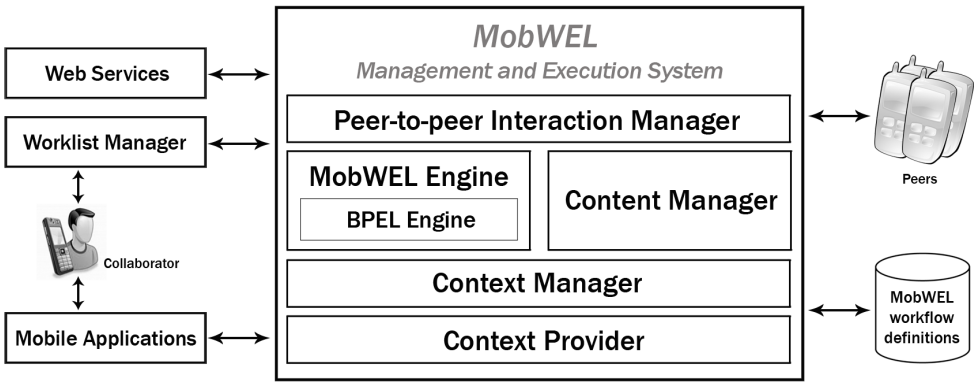

Figure 6.1: High-Level Architecture of the MobWEL Workflow Management System

External interactions are outlined between the MobWEL workflow management system and other external components. Interaction with *Web Services* is enabled by the BPEL engine which is capable of importing and exporting information by using

web services. *Worklist Manager* is an external component that provides task management. *Mobile Applications* use the workflow management services provided by the MobWEL workflow management system. Any mobile application can deploy own MobWEL workflow description to the MobWEL workflow management system and then request its instantiation. The deployed MobWEL workflow descriptions are stored in a *MobWEL workflow definitions repository*. In peer-to-peer mobile collaboration, each device behaves as an autonomous management unit and participates equally in workflow execution and management. The MobWEL workflow management system running on one device communicates and interacts with other MobWEL workflow management systems deployed on peer devices.

Internally, the system is composed of the following components:

Context Provider is a component responsible for context monitoring, acquiring, processing, aggregating and disseminating.

Context Manager manages the use and consumption of context information and drives context routing to other components at run-time.

Content Manager provides advanced content management functionalities and manages the evolution of content management lifecycles.

MobWEL Engine instantiates, manages and executes MobWEL workflow instances. The MobWEL engine extends a BPEL engine.

Peer-to-peer Interaction Manager manages communication and messages exchanges between mobile devices.

As mentioned, each component manages a particular aspect of MobWEL workflows. Each component has an integrated parser that enables interpretation and parsing of the corresponding MobWEL workflow part. The MobWEL workflow parsing process is described next.

MobWEL Parsing Process

The main parser is the **MobWEL Parser**, a part of *MobWEL Engine*. This parser is invoked when a new MobWEL workflow description is deployed. The input for the parser is a MobWEL workflow definition, deployed as a set of XML documents. Documents are persisted in the *MobWEL Workflow Definition Repository*. This parser firstly analyses the MobWEL workflow descriptions, then builds the corresponding internal data representations and structures. The parsing process is illustrated in Figure 6.2.

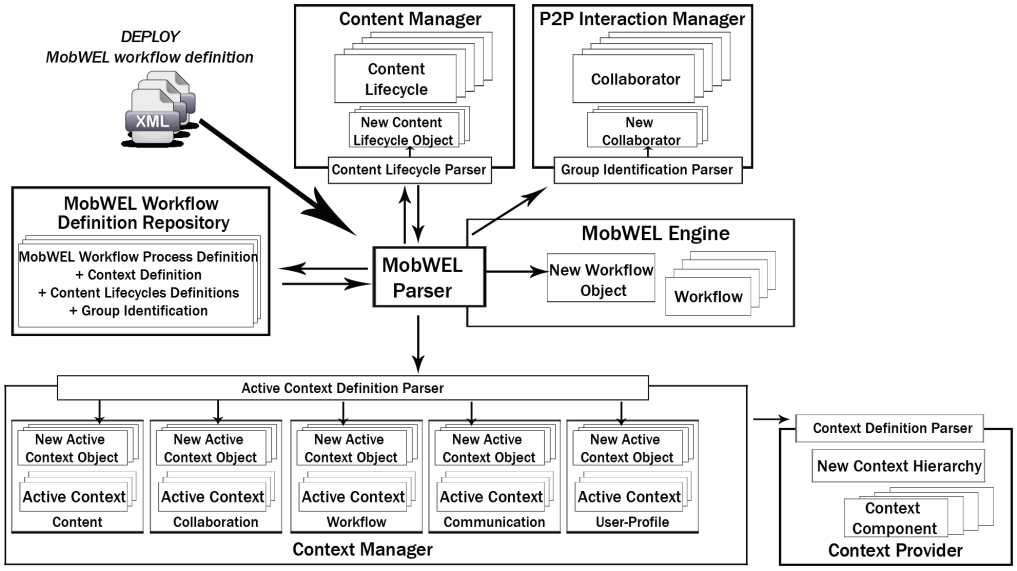

Figure 6.2: Deployment of MobWEL Workflow

Apart from the main control flow description, the MobWEL workflow definition contains descriptions of context-aware content lifecycles, context definition and collaborators group. Each workflow part is parsed by a parser in the corresponding system component as follows:

- **Content Lifecycle Parser**, a part of *Content Manager*, is called to parse the descriptions of context-aware content lifecycles and build internal lifecycle objects.
- To parse the group identification part, **Group Parser** in *Peer-to-peer Interaction Manager* is used.
- **Active Context Definition Parser** and **Context Definition Parser** interpret the context definition part of the MobWEL workflow.
  - **Active Context Definition Parser** as a part of *Context Manager* parses the context definition part and extracts information only about workflow-active context variables which will have a direct impact on the workflow execution.
  - **Context Definition Parser** in *Context Provider* is used to interpret context hierarchies and build internal structures in order to manage processing of context information from its acquisition to its dissemination.

This section has presented the high-level architecture of the MobWEL workflow management system and the overall parsing process. In next sections, functioning of each component, starting with the description of Context Provider, is outlined.

6.3 Context Provider

This section describes Context Provider, a component designed for context provisioning on a mobile device.

Context data is acquired locally on each mobile device and consumed by the workflow management system. However, there can be other mobile context-aware applications running on the same device which could use and consume the same context data. It would be inefficient to have a number of context provisioning services with same functionalities implemented on one mobile device. Therefore, *Context Provider* has been designed to be used twofold as:

1. an internal component of the workflow management system, which provides services solely to the workflow management system. In this case, this component can be fully designed in a workflow-specific way.

2. an external component, respectively a stand-alone engine, which provides context provisioning services to several context consumers and mobile applications.

Context Provider interprets the context definition part of the MobWEL workflow description, but is able to interpret any context model definition which conforms to the context definition XML schema. These context definitions can be provided by other mobile applications. Based on the context model definitions, the provider can monitor, acquire, process and aggregates the relevant raw context data, and disseminates meaningful and required context information back to applications.

Context Provider also supports context querying. Mobile applications can synchronously communicate with the provider and obtain any context information at real-time.

The infrastructure of *Context Provider* is shown in Figure 6.3.

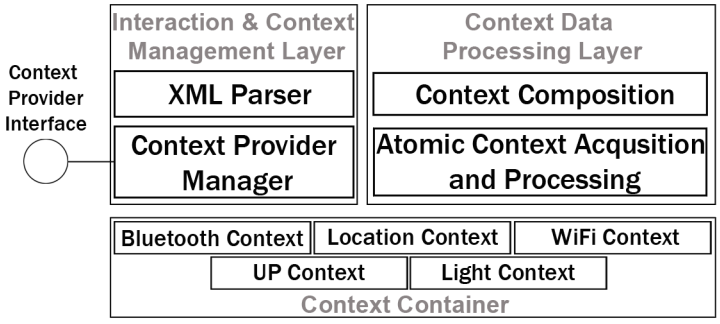

Figure 6.3: Context Provider Infrastructure

Context Provider has *Context Container* which contains self-contained context components. The container can contain various context components, each managing particular context concern. There are five context widget examples presented, Bluetooth Context, Location Context, WiFi Context, UP (User Preference) Context, and Light Context.

Context Provider has two main layers. The *Context Data Processing Layer* manages, processes and aggregates context data acquired by context widgets within the system. The *Interaction and Context Management Layer* manages interaction with other mobile applications, disseminates context information and manages lifecycles of context widgets. Interaction with other applications is enabled through *Context Provider Interface*.

The elements of Context Provider are in details described next.

6.3.1 Context Data Processing Layer

The *Context Data Processing Layer* represents the core of the context provider. In this layer, the internal context aggregations and hierarchies are built when context definition models are parsed. This layer supports context data monitoring, high-level context value derivation and context aggregation, as explored next.

Atomic Context Acquisition and Processing

Context Component is a construct designed to manage and process particular atomic context data that can be acquired from an external context source. By way of illustration, Bluetooth Manager can provide an information whether Bluetooth is switched on or off.

Although different context sources provide heterogeneous context data in various formats, the high-level principles for context acquisition and processing are same and can be generalised. Based on that fact, the blueprint of **Context Component**, shown in Figure 6.4, and the mechanism for context acquisition and processing have been designed.

Context Component is described by the *contextName* attribute which specifies the name of particular context. As various applications might define different high-level context values for the same context, **Context Component** can have one or more **ContextValuesSet**(s). Each set holds the information about the current context value (*lastContextValue*), and the date of last context change (*dateOfChange*). A **ContextValuesSet** can be defined once and used by multiple workflows or mobile applications. As each workflow or application is identified by its **AppKey**, the values set is associated with the particular app keys. For example, the Battery level can be

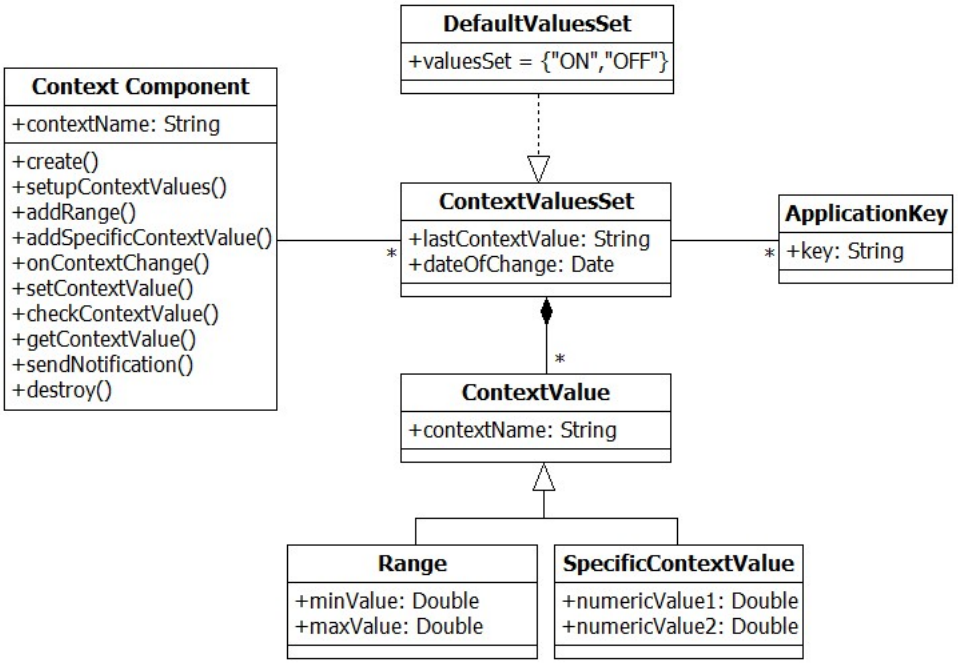

Figure 6.4: Context Component

associated with two different values sets defined: {LOW, MEDIUM, HIGH} or {LOW, FULL} depending on the specifications in the context definition model.

The values set {ON, OFF} is set as default set of values.

ContextValuesSet is a set of **ContextValue**(s). Often context values have to be derived from raw context data. **Range** and **SpecificContextValue** have been created to illustrate the use of the values descriptor designed for high-level context information derivation. **Range** supports the use of ranges, for example, a context value *LOW* can be derived for battery context if its raw numeric data is in a range between 0% and 5%. **SpecificContextValue** can be used to name locations defined by its coordinates, for example, the value of *"Museum"* can be set for a pair of coordinates. Another point in favour of using the values descriptor principle is that a finite set of context values can be built as shown in the example for battery level, two or three context values are defined instead of hundred.

The behaviour of ***Context Component*** is defined by using the following methods:

- *create and destroy* - the methods enable construction and destroying of the context component object;
- *setupContextValues* - the method is used to define a new context value set;
- *setContextValue* - this method sets a new context value in the values set;

- *addRange and addSpecificContextValue* - the methods are used to define a range or a specific context value;
- *getContextValue* - this method is a typical getter method;
- *checkContextValue* - in case that the context data needs to be obtained at real time, this method is used to query the context source. For example, some context sources can inform about context changes in certain intervals, so this method enables to query the context data at any time when required;
- *onContextChange* - if a new context raw data is obtained from a context source, the method is called. It tries to set the new context value in all context values sets. However, the new value is set only if two conditions are fulfilled:
  - (a) the new context value is an element of the context values set;
  - (b) the new context value differs from the previous context value.

  If the new value has been set, the sendNotification method is called. For example, if last context value for Bluetooth has been *OFF*, context information is broadcasted if and only if the new context value is *ON*. If context data obtained from Bluetooth adapter is *CONNECTING* then this information is not broadcasted further because it is not element of the context values set. Or if the battery level has been changed from 23% to 22%, the context value is still *MEDIUM*, therefore, there is no need to broadcast the same context value to listening parties;
- *sendNotification* - if a change of context value occurs, a notification is sent internally to all listening parties including composite components. The data structure of notifications is: *AppKey-ContextName-ContextValue-ContextDate*.

As shown, **Context Component** is an abstract class that captures the common context constructs and principles for context data acquisition and processing. However, context sources operate and provide context data in various ways. Thereby, subclasses of the **Context Component** class for managing specific atomic context data have to be defined in *Context Container*. These concrete components inherit most of the **Context Component** structure and behaviour. Only constructor and the *checkContextValue* method are context-specific constructs and have to be specified in each component.

Context Composition

Context composition (aggregation) is supported by introducing the **Composite Component** class, see Figure 6.5. Two or more types of **Context Component** can be loosely aggregated to form **Composite Component**.

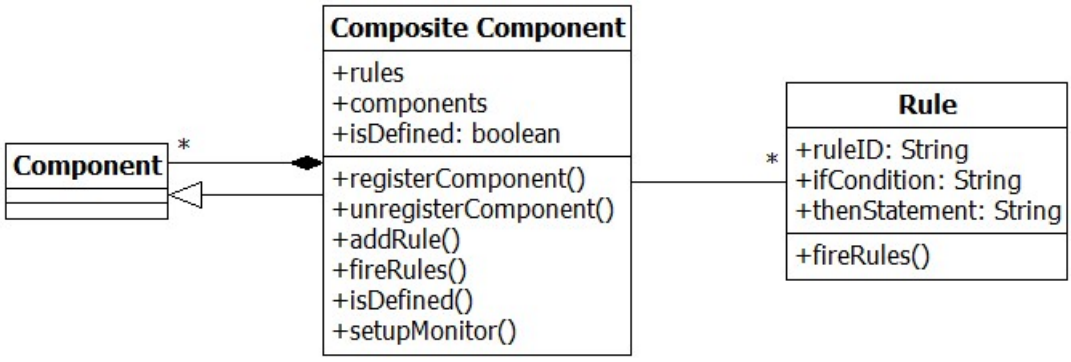

Figure 6.5: A Composite Component

The main role of the composite component is to aggregate context information as prescribed in the context definition model. Composite components inherit all constructs of context component as they also need to obtain context changes and act upon it. In addition, the composite component must be aware of context *components* it is composed of and *rules* which drive the derivation of aggregated context values. Composite components can be constructed, assembled and defined at runtime. The *isDefined* attribute is used to determine that the component has been fully defined, all child context components have been added and changes in child context values can be monitored.

The definition of context aggregated values is driven by rules. The rules are formalized by a function as follows:

Definition 6.1 (Rule for composite component)**.** *Let C_{comp} be composite component and $C_1, C_2, \ldots, C_m$ be its child context components. Let V_{comp} be the value set of C_{comp}. Let V_i be the value set of component $C_i, \forall i \in [0, m]$. Then rule function can be specified as*

$$f\text{: } V_1 \times V_2 \times \ldots \times V_n \rightarrow V_{comp}$$

and the rule R can be written as:

$$R(v_1, v_2, \ldots, v_n, f(v_1, v_2, \ldots, v_n)), \text{ where } v_i \in V_i \text{ and } f(v_1, v_2, \ldots, v_n) \in V_{comp}.$$

The values $v_1, v_2, \ldots, v_n$ are stored in the *ifCondition* attribute. The aggregated value: $f(v_1, v_2, \ldots, v_n)$ is stored in the *thenStatement* attribute.

The behaviour of **Composite Component** is defined by using the methods inherited from **Context Component**, and the following extra methods:

- *registerComponent and unregisterComponent* - the methods enable adding and removing of a child context component object.
- *addRule* - the method is used to add a rule.
- *fireRules* - when a child context value changed, the method is used to fire rules and determine new aggregated context value.
- *isDefined* - to set the value of the *isDefined* attribute.
- *setupMonitor* - once the composite component is fully defined, this method is used to launch a monitoring mechanism for notifications broadcasted by child context components.

Basically, a composite component object can be in the *Definition* or *Active* state as shown in Figure 6.6.

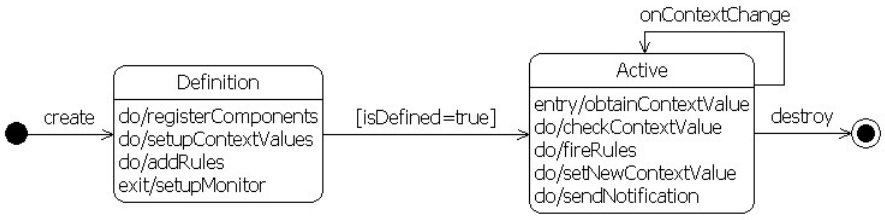

Figure 6.6: Composite Component Lifecycle

The state diagram shows methods that are performable in each state. The object is in the initial *Definition* state until its all child components and rules are defined. While in the *Definition* state, its child context components can be registered, its aggregated context value set can be defined, and rules can be specified. The fully defined composite component becomes active when the *isDefined* attribute is set to true. At the moment of leaving the *Definition* state, the *setupMonitor* method is called.

In the *Active* state, a situation is constantly monitored for any child's context change. If such a context change occurs, composite component is notified. Rules associated with the composite component are fired to determine the aggregated context value. Similarly as in the context component, the *sendNotification* method is called only when the new aggregated context value differs from the previous value, as shown in the following example.

Consider two context components - *Bluetooth* and *Wifi*, both with value sets: $\{ON,OFF\}$. A composite component object named *DATASYNC* is defined with its own values set $\{ON, OFF\}$, and with *Bluetooth* and *Wifi* registered as its children. Rules for *DATASYNC* are defined as follows (*R1*($\{ON,ON\}$, *ON)*, *R2*($\{OFF,ON\}$, *ON)*, *R3*($\{ON,OFF\}$, *ON)*, *R4*($\{OFF,OFF\}$, *OFF)*).

Let have the initial child context values of both, Bluetooth and Wifi context, be set to *ON*. So the aggregated context value of *DATASYNC* is *ON*. A context change in one child context component ($ON \mapsto OFF$) does not change the aggregated context value of *DATASYNC*. The aggregated context value of DataSync would remain (*ON*), thus no notification is sent and the *sendNotification* method is not invoked.

The relationships between **Context Component** and **Composite Component** enable building context hierarchies in which low level context components are loosely coupled to form high level composite contexts. These context hierarchies and data structures correspond to the context definitions specified in the parsed XML documents. As demonstrated, this layer enables monitoring, acquiring and processing of relevant context data.

Next, the layer with the role to manage lifecycle of context components, and disseminate context information to applications, is presented.

6.3.2 Interaction and Context Management Layer

The main component of the *Interaction and Context Management Layer* is **Context Provider Manager**. This layer has several characteristics:

1. When a new context definition is received, it invokes the **XML Parser** to parse XML description and builds corresponding internal context hierarchies.

2. It maintains **Context List**, a list of **Context Component**s which are in the active state. Keeping the list up-to-date ensures that each component runs only once.

3. It disseminates context information to external components or application. The messages are broadcasted in the *AppKey-ContextName-ContextValue-Context-Date* format.

4. This component implements **Context Provider Interface** and provides support for context querying.

6.3.3 Context Provider Interface

To standardise interaction between the context engine service and applications, it is necessary to have a well-defined interface. **Context Provider Interface** include specifications for context definition and context querying as follows:

- *setupContexts* The method is called by applications in order to deploy context definition described in an XML document.
- *registerContextPath* Particular context components for specific context have to be defined in *Context Container*. However, not all required context component can be known when context provider is built. Therefore, to support dynamic class loading and addition of other context components into container at run-time, this method can be used. The input parameter is the actual path to the context components.
- *getContextValue* The method is used for context querying at real-time. The input for the method is component name and output is the current context value.

6.3.4 Context Provider Usage

As outlined, there are two options how *Context Provider* can be used, either internally within the workflow management system, or externally. *Context Provider* has been designed in a generic way to support both ways of usage, see Figure 6.7.

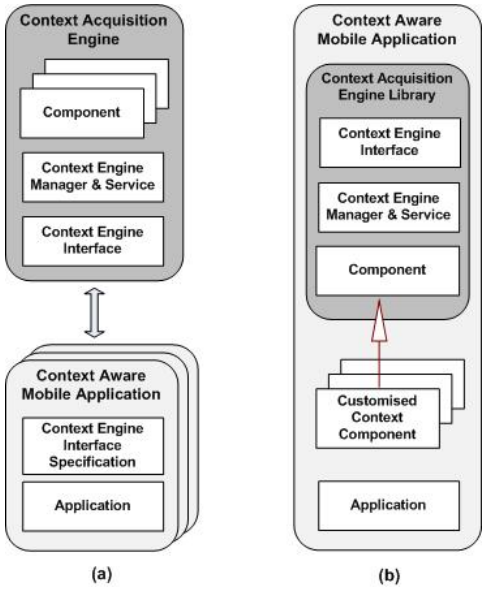

Figure 6.7: Usage Scenarios

Firstly, *Context Provider* can be deployed on a mobile device as a standalone background service providing support for all context-aware applications as shown in Figure 6.7(a). In this case, context-aware applications interact with it by using its **ContextEngine Interface**. Secondly, *Context Provider* can be used as a library within an application or workflow management system, as shown in Figure 6.7(b). In this case, the particular context-aware application can extend the functionality of the *Context Provider* by defining own Context Container and own context components which inherit the attributes and behaviour from **Context Component**.

This concludes the description of *Context Provider*. Next section describes Context Manager, a component designed to manage the use and consumption of context information within the workflow management system.

6.4 Context Manager

The use and consumption of context information within the MobWEL workflow management system is managed by the *Context Manager* component. The design of the component is described in this section.

Context Manager acts as intermediator between Context Provider and other internal workflow management components. Context Manager contains mechanisms to synchronously and asynchronously communicate with Context Provider. To communicate effectively, both components need to be aware about the same contexts. Because of this, the same context definition is deployed to both components. When a new context definition is deployed to Context Manager, Context Manager also passes the definition to Context Provider. This ensures that Context Provider broadcasts context messages that are related to the parsed workflows and Context Manager can process them further. Context Manager has implemented a mechanism to receive the broadcasted messages and also a mechanism to query context when required.

Context information is consumed within the workflow management system in several ways. Context Provider broadcast context notifications, however, the messages do not contain information how the data should be further used within the system or which component needs it. Therefore, the further use and routing of context information is managed by Context Manager. Context Manager persists the last known context values of workflow-active context components and provides context information to other internal system components which can query it at real time.

The structure of this component is shown in Figure 6.8.

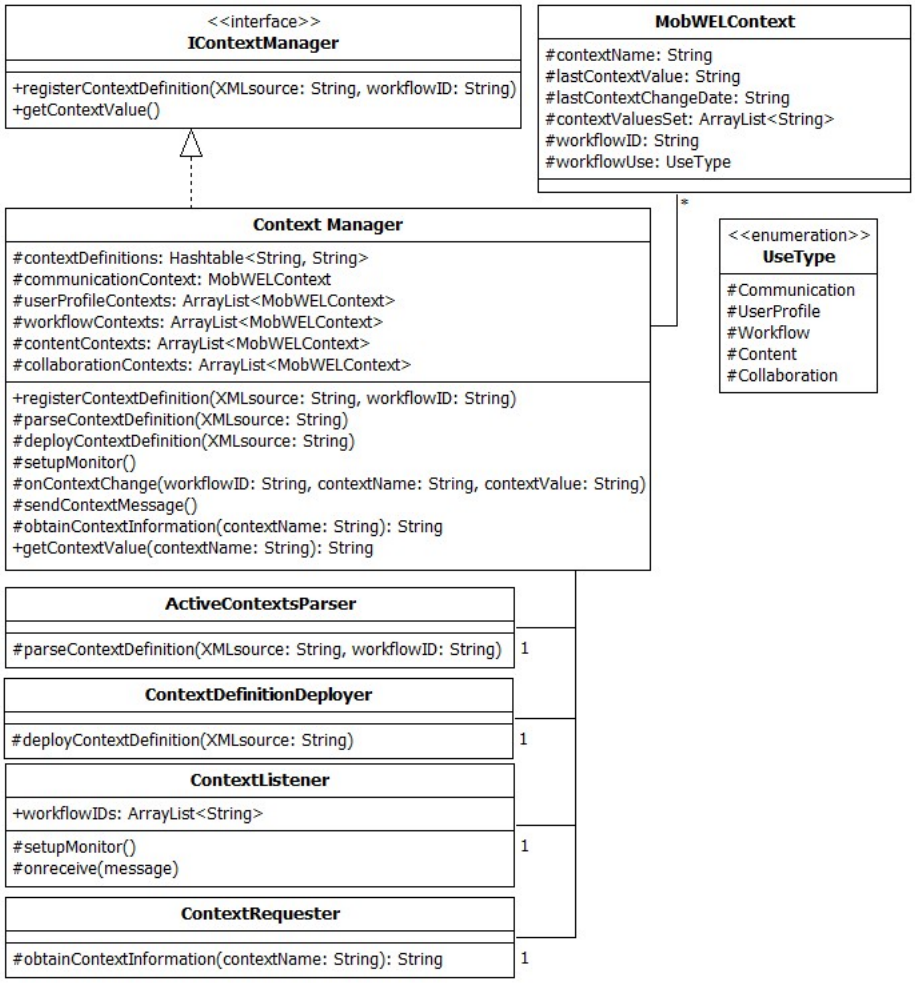

Figure 6.8: Context Manager

The **Context Manager** class is a main element in the package. There are four elements created to provide different functionalities: **ActiveContextsParser**, **ContextDefinitionDeployer**, **ContextListener**, and **ContextRequester**. Algorithm 6.1 outlines the sequence of all operations performed when a context definition document is deployed and the *registerContextDefinition(String XMLsource, workflowID: String)* method is invoked in Context Manager.

Context Manager is associated with **ActiveContextsParser**, an element that interprets and parses MobWEL context definitions. The parser extracts only information about workflow-active contexts. This ensures that only relevant and required context information is received and further managed. To support this, the **MobWELContext** class has been created. For each workflow-active context defined in the XML description, a MobWELContext object is constructed. The attributes specified in the MobWELContext class are *contextName, lastContextValue, lastContextChangeDate, contextValuesSet, workflowID* and *workflowUse*. Apart from

Algorithm 6.1 Registering of Context Definition

```
Input: A path to the XML description (XMLsource) and the ID of the workflow
(workflowID).
Output: A set of constructed MobWELContext objects.
Begin
if XMLsource not empty then
    //Step 1. parse workflow active contexts
    ActiveContextsParser.parseContextDefinition(XMLsource, workflowID)
    //Step 2. deploy the context definition to Context Provider
    ContextDefinitionDeployer.deployContextDefinition(XMLsource)
    //Step 3. add to the list of context definitions persisted in Context Manager
    contextDefinitions.add(XMLsource, workflowID)
    //Step 4. obtain initial context values
    while exist O.workflowID = workflowID do
        ContextRequester.obtainContextInformation(O.contextName)
    end while
    //Step 5. if no listener, set up one via setupMonitor method
    if ContextListener.contextMonitor is not set then
        ContextListener.setupMonitor()
    end if
end if
End
```

the *lastContextChangeDate* and *lastContextValue* attributes, the values of the MobWELContext attributes are extracted from the context definition document.

Algorithm 6.2 shows details of Step 1 and Step 2. In Step 1, the actual parsing process is performed. When active contexts are parsed and corresponding internal MobWELContext objects constructed, **ContextDefinitionDeployer** is used and the definition is passed and deployed in Context Provider, as shown in Step2.

The context definition is added to the list of definitions in Context Manager (STEP 3). After that, Context Provider creates corresponding context hierarchies and broadcast relevant messages about context changes. To ensure that Context Manager has current context values of all active contexts, **ContextRequester** is used and the initial context values are obtained by invoking this method (STEP 4):

ContextRequestuer.obtainContextInformation(O.contextName) .

This method is invoked for each constructed MobWELContext object. **ContextRequester** binds synchronously to Context Provider and query particular context information by invoking the *ContextProvider.getContextValue(contextName)* method through the Context Provider's Interface2.

To receive context information from Context Provider, **ContextListener** is used and a listening mechanism in this element is established. This mechanism is set up by calling the *setupMonitor* method. By setting the context monitor, the broadcasted

Algorithm 6.2 Registering of Context Definition - STEP 1 and STEP 2

Input: A path to the XML description (XMLsource) and the ID of the workflow (workflowID).
Output: A set of constructed MobWELContext objects.
Begin (STEP1)
for context C defined in XMLsource **do**
  if C.getAttributeValue(workflowActive) = 'YES' **then**
    construct a new MobWELContext object O
    $O : contextName \leftarrow C : getAttributeValue(contextName)$
    $O : workflowUse \leftarrow C : getAttributeValue(consumptionType)$
    $O : workflowID \leftarrow workflowID$
    while exist C.getAttributeValue(contextValue) **do**
      O.contextValuesSet.add(C.getAttributeValue(contextValue))
    end while
    if O.workflowUse = 'Content' **then**
      ContextManager.contentContexts.add(O)
    else if O.workflowUse = 'Communication' **then**
      ContextManager.conmmunicationContext = O
    else if O.workflowUse = 'Collaboration' **then**
      ContextManager.collaborationContexts.add(O)
    else if O.workflowUse = 'Worfklow' **then**
      ContextManager.workflowContexts.add(O)
    else if O.workflowUse = 'UserProfile' **then**
      ContextManager.userProfileContexts.add(O)
    end if
  end if
end for
End (STEP1)
Begin (STEP2)
if Context Provider is external component **then**
  bind to context provider service (ContextService)
  ContextService.setupContexts(XMLsource)
  unbind
end if
if Context Provider is internal component **then**
  ContextProvider.setupContexts(XMLsource)
end if
End (STEP2)

messages can be received. When a message from Context Provider is received, the *ContextListener.onReceive(Message m)* method in ContextListener is invoked and operations listed in Algorithm 6.3 are performed. The context message specifies context name, thereby, the corresponding MobWELContext object is found, and its *lastContextValue* and *lastContextChangeDate* values are changed to the values obtained from the message.

Algorithm 6.3 Obtaining context notification from Context Provider

Input: A message M received from ContextProvider.
Output: New context value persisted in Context Manager.
Begin
//obtain data from the message
$workflowID \leftarrow M.getString(\text{"}contextApplicationKey\text{"})$
$contextName \leftarrow M.getString(\text{"}contextName\text{"})$
if workflowID exist in ContextListener.workflowIDs **then**
$MobWELContextO \leftarrow ContextManager.findObject(workflowID; contextName)$
$O : lastContextValue \leftarrow M : getString(\text{"}contextInformation\text{"})$
$O : lastContextChangeDate \leftarrow M : getString(\text{"}contextDate\text{"})$
ContextManager.onContextChange(O)
end if
End

Finally, Context Manager is informed about the context change through the *ContextManager.onContextChange(MobWELContext O)* method and based on the *workflowUse* attribute, the context message is delivered to the right internal component.

6.4.1 Context Manager Interface

Context Manager provides support to other components of the workflow management system through the methods defined in **Context Manager Interface**:

- *registerContextDefinition and removeContextDefinition* - The methods are used to register or remove workflow-specific context definitions. The path to the XML description is an input. Also *workflowID* has to be specified to keep association between the workflow case and active contexts.
- *getContextValue* - Context Manager holds the last context values which can be provided upon a request at real time.

This section described the *Context Manager* component of the MobWEL workflow management system. In next section, the component for content management, *Content Manager* is presented.

6.5 Content Manager

This section presents *Content Manager*. *Content Manager* provides advanced content management functionalities and its role is to control and manage lifecycles of content items manipulated in workflows. This component parses *Context-aware content lifecycle*, the MobWEL workflow part that describes: *a*) content-related metadata; and *b*) content lifecycle. One lifecycle definition describes the behaviour of corresponding content objects processed in all running workflow instances of the particular workflow. Content Manager monitors the evolution of the content objects in running workflow instances and maintains information about their current states according to the lifecycle description. If a content state of a content object is changed, it informs the particular workflow instance about the change, and the instance can act upon that change. The structure of *Content Manager* is shown in Figure 6.9.

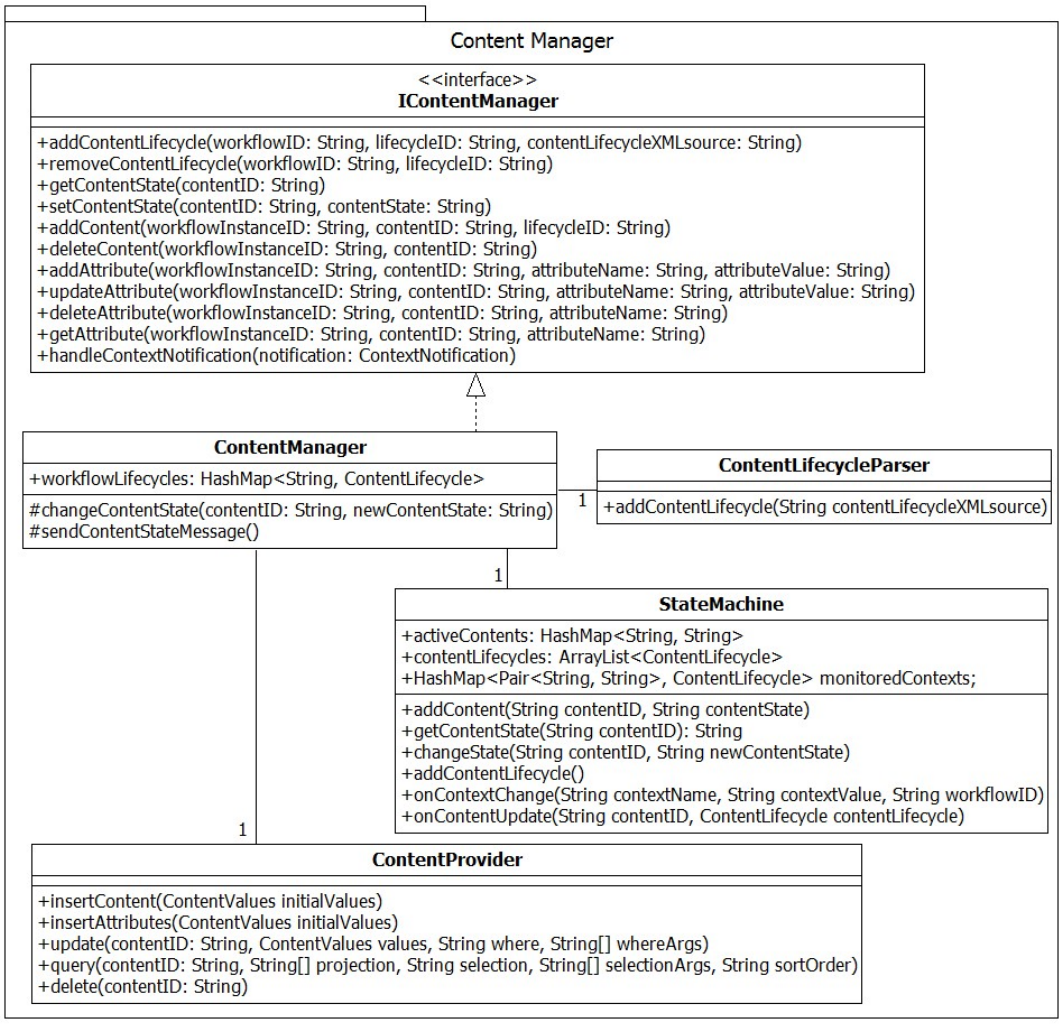

Figure 6.9: Content Manager

Three elements have been constructed, each focusing on a different functional aspect. Firstly, **ContentLifecycleParser** is an XML parser that interprets the context-aware content lifecycles and build internal data structures. Secondly, content-related metadata is persisted and managed by using **Content Provider**. Finally, *Content State Transition System* is used to manage content lifecycles and maintain content states. The **Content Manager** class, at the top of the hierarchy, manages these three components. The manager implements methods specified in the **Content Manager Interface**.

Next sections describe the elements in details.

6.5.1 Content Provider

Access to a structured set of content-related metadata is managed by the element called *Content Provider*. The structure of data in *Content Provider* is described in the data model depicted in Figure 6.10.

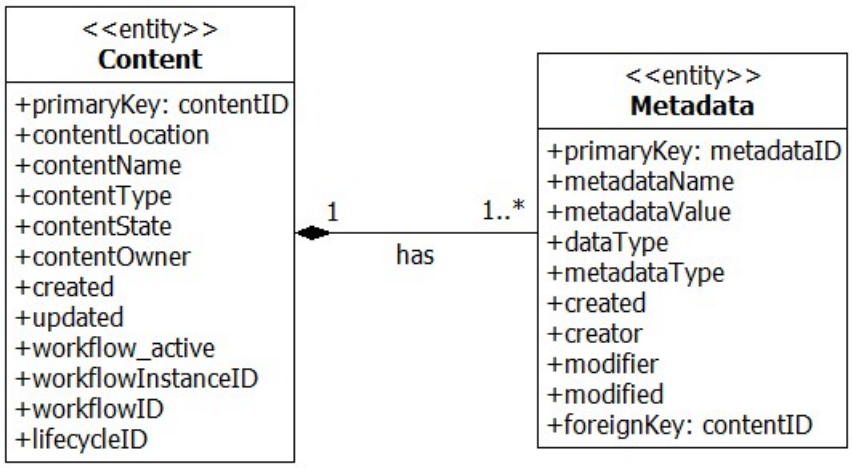

Figure 6.10: Data Model of Content Provider

The data model shows two entities: **Content** and **Metadata**. As the names suggest, the first entity is created for content objects and the other entity for their associated metadata. The *Content* entity contains properties that define the characteristics of each content object processed in a workflow instance. The *contentID* attribute identifies uniquely a particular content object. The identifier of each object is formed by the workflowInstanceID concatenated with the name of the corresponding content variable in the process flow. Content objects are stored in the mobile file system and the *contentLocation* attribute specifies the actual path to the given content object. The *contentName* and *contentType* attributes correspond to name and type of the content object. The attributes *created* and *updated* refer to the creation

date and date of last update of the record in this entity. The *contentState* and *contentOwner* attributes define the recent content state, and the owner or creator of the content object.

Other fields in this entity are workflow-specific. To determine whether the content object is or is not processed in a running workflow instance, the *workflowActive* attribute is used to describe the information. The *workflowID* and *workflowInstanceID* attributes correspond to the identifiers of the workflow and the workflow instance that processes the given content object. The last field, *lifecycleID*, refers to the identifier of the relevant context-aware content lifecycle.

A content object can be associated with one or more metadata. The *Metadata* entity has attributes corresponding to the associated metadata. The *metadataName, dataType* and *metadataType* fields in this entity is populated with the metadata details described in the context-aware content lifecycle description. The actual value of metadata is stored in the *metadataValue* field. The *created, creator, modifier* and *modified* fields are used to keep track when was the metadata record created or last modified and by whom.

6.5.2 Content Lifecycle Parser

Content Lifecycle XML Parser converts content lifecycles described in XML documents into internal data structures illustrated in Figure 6.11.

Content Lifecycle Parser constructs corresponding *ContentLifecycle* objects. Each *ContentLifecycle* object is associated with a set of *Metadata* objects, *ContentState* and *ContentTransition* objects. The source and target states, called *preState* and *postState*, of each *ContentTransition* object refer to the corresponding *ContentState* objects. After that, guards of transitions are mapped into expression objects which are further parsed into *Condition* objects. To enable condition evaluation, in each *Condition* object, the left and the right side of the expression are separated, and the relation operator is exempted. To enable context monitoring, the context name and expected context value are extracted and set in the *ContentAwareCondition* and *ContextDrivenCondition*. For the *AttributeCondition* objects, the attribute (metadata name) and expected attribute value are extracted from the expression. When a condition is evaluated, the *evaluateCondition()* method is invoked. In this evaluation process, the actual values of left side and right side are obtained by calling the *obtainValues()* method and persisted in the *leftSideValue* and *rightSideValue* fields. Finally, the relational operator for comparison of these two values and either true or false is returned for the condition evaluation.

However, there have been some issues identified in the parsing process of the

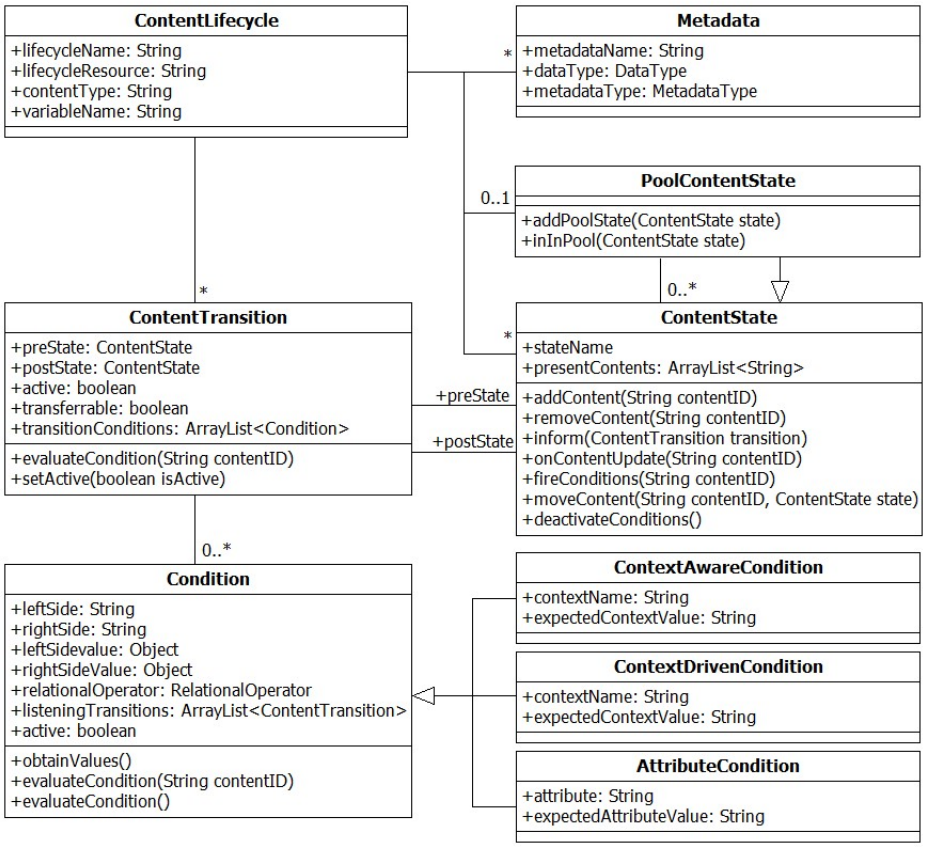

Figure 6.11: Internal Structures of Content Lifecycles

same content lifecycle on devices with different roles. Firstly, distributed workflow execution imposes lack of consistency in the evolution of the content lifecycle across multiple devices because a content object can be in two different content states on two different devices. Secondly, context transitions requires context monitoring which is role-specific. Attempts to fire the transitions between two states on all devices is inefficient as certain transitions are associated only with certain workflow partitions, and relevant context should be monitored only on the devices which have the partitions deployed. Thereby, to overcome these issues, the concept of *PoolContentState* has been introduced. *PoolContentState* is basically a specific content state that is also a pool of other content states. In the parsing process, all content states, which are not associated with the particular role and parsed workflow partition, are placed into the pool and the transitions between these states are neglected in the parsing process and not mapped into *ContentTransition* objects. The exception is for the transitions between a role-specific content state and a content state within the pool state. In this case, the transition is mapped into the *ContentTransition* object that connects the role-specific content state and the pool state. The concept of the pool state is illustrated in Figure 6.12.

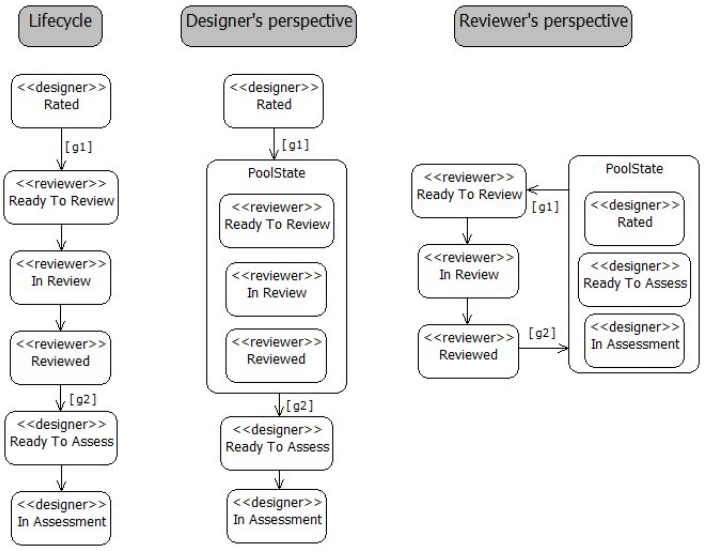

Figure 6.12: Lifecycle Example in Different Representations

There is a part of the picture lifecycle illustrated. The lifecycle contains six content states, three content states are associated with the role of designer, and three of them with the role of reviewer. Consider that there are two guards, $g1$ and $g2$, placed on the transitions between the *Rated* and *Ready To Review* content states, and between the *Reviewed* and *Ready To Assess* content states. When this lifecycle is parsed with the workflow partition defined for the role of designer, all states not owned by designer are placed in the pool state, and corresponding transitions ($g1$ and $g2$) become transitions between the actual content states and the pool state. Similarly, in the parsing process for the role of reviewer, the irrelevant content states for this role are placed in the pool state. These two perspectives, the *Designer's perspective* and *Reviewer's perspective*, are illustrated in the same figure.

The benefits of using pool states are twofold:

1. Context-related guards are evaluated only on relevant devices. Thus context is monitored only on those devices that need it.

2. There is no need for full content state synchronisation over devices. Each device can act upon local information. If content is in a state that belongs to the pool state, it means that the content is processed on another device.

The parsing process of a context-aware content lifecycle description is described in Algorithm 6.4.

The parsing process constructs a corresponding *ContentLifecycle* object. The constructed objects become a part of the *Content State Transition System* which is described next.

Algorithm 6.4 Adding of Context-Aware Content Lifecycle

Input: A path to the XML description, *XMLsource*, role R and the ID of the workflow, *workflowID*.
Output: ContentLifecycle object named L_{object}.
Begin
get XML with the XMLsource path
construct pool content state S_{pool}
if XMLsource not empty **then**
  construct a new ContentLifecycle object L_{object}
  while exist metadata M in XML **do**
    construct a new Metadata object M_{object} with attributes of M
    L_{object}.add(M_{object})
  end while
  while exist state S in XML **do**
    construct a new ContentState object S_{object} with attributes of S
    L_{object}.add(S_{object})
    if $S_{object}.roleName$ equals R **then**
      S_{pool}.add(S_{object})
    end if
  end while
  while exist transition T in XML **do**
    if $T.sourceState$ is not in S_{pool} **then**
      construct a new ContentTransition object T_{object} with attributes of T
      $sourceState \leftarrow T.sourceState$
      $sourceState_{object} \leftarrow L.findState(sourceState)$
      if $T.destinationState$ is not in S_{pool} **then**
        $targetState \leftarrow T.targetState$
        $targetState_{object} \leftarrow L.findState(targetState)$
      else
        $targetState_{object} \leftarrow S_{pool}$
      end if
      T_{object}.setPreState($sourceState_{object}$)
      T_{object}.setPostState($targetState_{object}$)
      if exist guard G for T in XML **then**
        construct Expression E corresponding to G
        construct condition C
        $C \leftarrow E.parseExpression$
        $T_{object}.add(C)$
      end if
      L_{object}.add(T_{object})
    end if
  end while
end if
  return L_{object}

6.5.3 Content State Transition System

The last element of *Content Manager* is the **Context-Aware Content State Transition System** which is a finite state transition system that has been designed and developed to support management of context-aware content lifecycles. The system is defined as follows:

Definition 6.2 (Context-Aware Content State Transition System)**.** *Let L be a context-aware content lifecycle. Then Context-Aware Content State Transition System for the lifecycle L is described as a 4-tuple* $<S,s_0,C,\mapsto>$*, where*

- $S \stackrel{def}{=} \{\mathrm{s_i} : \mathrm{i} \in [0, \mathrm{n}]\}$ *is a finite, non-empty set of content states;*
- s_0 *is an initial state;*
- $C \stackrel{def}{=} \{\mathrm{c_i} : \mathrm{i} \in [0, \mathrm{p}]\}$ *is a set of conditions;*
- $C_{CA} \subset C$ *are context-aware conditions,* $C_{CD} \subset C$ *are context-driven conditions, and* $C_{AT} \subset C$ *are attributes-related conditions,* $C_{CA} \cap C_{CD} \cap C_{AT} = \varnothing$*;*
- $\mapsto \subset \ S \times C \times S$ *is a transition function from a source content state to the destination content state under a certain condition.*
  If $s_i, s_j \in S, c_k \in C, (s_i, c_k, s_j) \in \mapsto$*, then the transition can be written as:*
  $$s_i \overset{c_k}{\mapsto} s_j$$

There are three important aspects of the system explored next.

1. A number of content items processed in multiple workflow instances will have the same content lifecycle. Therefore, the idea of content spaces used in Alfresco, as described in Chapter 3, has been adapted in the *Context-Aware Content State Transition System*. Smart folders, as the content spaces are called in Alfresco, have been mapped into our system content states. It means that each content state in the content lifecycle behaves as a folder that can contains the references to the content items which are in this particular state at the given point of time. As has been shown in Figure 6.11, each state has an array list of *presentContents* that contains the content items IDs. This approach also enables to add rules when a content item enters the state, is in the state, or leaves the state.

2. Functioning of the *Context-Aware Content State Transition System* is associated with the following set of possible input events:

   - **Context information** change when a notification is received from Context Manager;

- **Content update** when a content item has been manipulated by an invoked and executed activity in the workflow process;
- **External content state change** when the state of a content item has been changed on another device and notification has been received.

3. At any time, a content item is in a certain state of its lifecycle. As already outlined, movement to a new state is triggered when an associated condition is evaluated to true. At the basic level, the system is a variant of state transition system, therefore, it inherits its behaviour. By integrating context awareness, conditions are evaluated in a different manner and content flow between two states becomes more specific. Context-driven and context-aware conditions must be well placed and designed as a context-driven condition is evaluated only once when a content item enters the content state with such transition, whereas a context-aware condition is evaluated when a context change occurs.

Operating of the state transition system follows Algorithm 6.5.

6.5.4 Content Manager Interface

Content Manager provides content management support to other components of the workflow management system. Other components communicates with *Content Manager* via its interface. The following functions are made accessible through its interface:

addContentLifecycle - This method is called when a new workflow definition is added and it includes a context-aware content lifecycle that needs to be parsed.

getContentLifecycle - It returns a content lifecycle object identified by its name.

removeContentLifecycle - Tho remove the lifecycle from Content Manager.

createContent - The method is used to create a content object in Content Provider.

addContent - If a content item is obtained from another device, it can be added to Content Manager by using this method.

onContextChange A context notification can be delivered to this component through this method.

addAttribute/updateAttribute/deleteAttribute Methods used to add, update or delete an attribute value.

Algorithm 6.5 Transition of content object O: $s_i \overset{c}{\mapsto} s_j$

Input: A content object O in a content state S_i. A set of outgoing transitions from the state S_i. Conditions placed on the transitions belong to one of the three disjoint sets C_{CA}, or C_{CD}, or C_{AT}.
Output: A content object O in a new content state S_j.
Require: A context change listener is set up for any condition $c \in C_{CA}$

```
if onContentUpdate operation for object O has been performed then
    for each transition t with condition c ∈ C_AT do
        if evaluate(c) then
            fire transition t and move O into the target state S_j of t
            inform state machine that O in a new content state S_j
            exit
        end if
    end for
    for each transition t with condition c ∈ C_CD do
        contextValue ← c.obtainContextInformation(contextName)
        if c.expectedContextValue = contextValue then
            fire transition t and move O into the target state S_j of t
            inform state machine that O in a new content state S_j
            exit
        end if
    end for
end if
if onContextChange notification received then
    contextName ← notification.obtainContextName
    contextValue ← notification.obtainContextValue
    for each transition t with condition c ∈ C_CA do
        if c.expectedContextName = contextName AND
c.expectedContextValue = contextValue then
            fire transition t and move O into the target state S_j of t
            inform state machine that O in a new content state S_j
            exit
        end if
    end for
end if
```

This section has described Content Manager. Next section outlines the MobWEL engine.

6.6 MobWEL Engine

The *MobWEL engine* is the heart of the system that interprets MobWEL workflow definitions, instantiate and executes MobWEL workflows. Thereby, the engine needs to contain a parser and the execution unit. The MobWEL Parser and the parsing process has been already described in Section 5.4.

As the engine extends a BPEL engine, the execution unit is already capable of managing BPEL processes, thereby, only the support of execution of the constructs introduced in the MobWEL language, such as *interactionActivity* and *contentActivity*, has to be added. The semantics for *interactionActivity* is inherited from $BPEL^{light}$. The *contentActivity* is an extended BPEL activity which has a simple role to inform Content Manager. Therefore, it is handled in the same manner as other BPEL activities and its internal execution is shown in Algorithm 6.6.

Algorithm 6.6 The internal execution of *contentActivity*

```
Input: A set of data: action A, content variable C, element E and value V.
if A = ”add” then
    ContentManager.addAttribute(workflowInstanceID, C, E, V)
    exit
end if
if A = ”update” then
    ContentManager.updateAttribute(workflowInstanceID, C, E, V)
    exit
end if
if A = ”remove” then
    ContentManager.deleteAttribute(workflowInstanceID, C, E, V)
    exit
end if
```

Support for context and content awareness needs to be developed in the MobWEL engine. The MobWEL built-in functions presented in Section 4.3.5 are supported in the engine. When a condition containing a MobWEL function is evaluated, context information or content state is queried and obtained from Context Manager or Content Manager. The MobWEL engine handles the results of the condition evaluation in the same way as any other condition. Notifications from Context or Content Manager need to be handled by two methods are developed: *onContextChange* and *onContentStateChange*. These methods handle incoming events notifications and

route the information to the relevant workflow process. Event notifications are handled by event handlers.

The last component of the MobWEL workflow management system, Peer-to-Peer Interaction Manager, is presented next.

6.7 Peer-To-Peer Interaction Manager

The role of *Peer-to-peer Interaction Manager* is to manage transmissions of messages between mobile devices. This component cooperates with communication middleware which is responsible for the actual transmission and is not a part of the workflow management system. *Peer-to-peer Interaction Manager* is informed about successful or failed message delivery and acts upon the notification events by passing the message back to the workflow execution engine.

Peer-to-peer Interaction Manager contains the following elements:

Event Handlers: Event Handler handles incoming messages from other devices and requests coming from mobile applications or services. An event handler can be designed for each form of communication such as for incoming data, SMS, MMS messages or messages coming via Bluetooth connection. Event handlers are platform-specific.

Message Parser: Structured information and messages are sent between devices as a sequence of bytes. *Message Parser* is a component used to convert workflow-related data and objects into such message format that can be transmitted across the network. The parser also extracts workflow objects from incoming messages.

Identity Manager: In peer-to-peer workflow execution, the devices are aware of other fellow workflow participants and their devices. The identifiable elements of the participating mobile devices are phone number, Wi-Fi and Bluetooth MAC addresses. *Identity Manager* is used to store and manage the details about participating collaborators. The details include collaborators' names, their roles, and other data needed for device-to-device communication. In addition, a record of collaborator's availability based on the last known collaboration-related context value, is managed in this element.

6.7.1 Interaction and Message Handling

Once the MobWEL workflow was deployed, its workflow instances can be created. A mobile application creates a request to instantiate the workflow case. The work-

flow management system handles the request and instantiates the corresponding workflow. The workflow instance is created on a single mobile device. However, to complete the workflow instance, other collaborators and their devices must be involved in its execution. There are several ways how this can be achieved. For example:

- Static group of collaborators (defined for each instance): Each device holds collaboration-related context information about collaborators' availability. Based on this information, the group of collaborators can be defined when the instance is created. All participating devices are informed about the new instance so they can prepare resources for the instance execution. However, the disadvantage of this method is that by fixing the group of collaborators at the time of instance creation, the flexibility to choose collaborators anytime during the workflow instance execution is impossible.

- Static group of collaborators (shared for a number of instances): Another similar option is to use the same group of collaborators for a number of workflow instances if they are created within certain amount of time. This option is practical when a large number of instances is created within short period of time but in terms of flexibility has more disadvantages than the previous option.

- Dynamic group of collaborators (defined on-the-fly): The best option is to choose fellow workflow participants at any time during the instance execution.

- Collaborator's ability to participate is considered: Collaborator can set own preferences, for example, whether is currently available, or how many request wishes to handle in a certain period of time.

The options are listed to show the possible interactions and how knowing the collaborators' context situation can add value to the MobWEL workflow execution. However, it is out of the scope of this thesis to optimise the interaction model and this will be addressed in the future. To achieve the objectives of this work, the first option has been chosen as the most convenient one. Thus a group of collaborators is fixed when an instance is created and all messages related to the workflow instance are shared only among the group. All participants are informed about creation of a new workflow instance. Similarly, when the workflow instance is completed or terminated, all participants are informed. While the workflow instance is running, messages are exchanged between workflow participants.

Basically, from high-level perspective there are only three types of messages that can be shared between two mobile devices as shown in Figure 6.13.

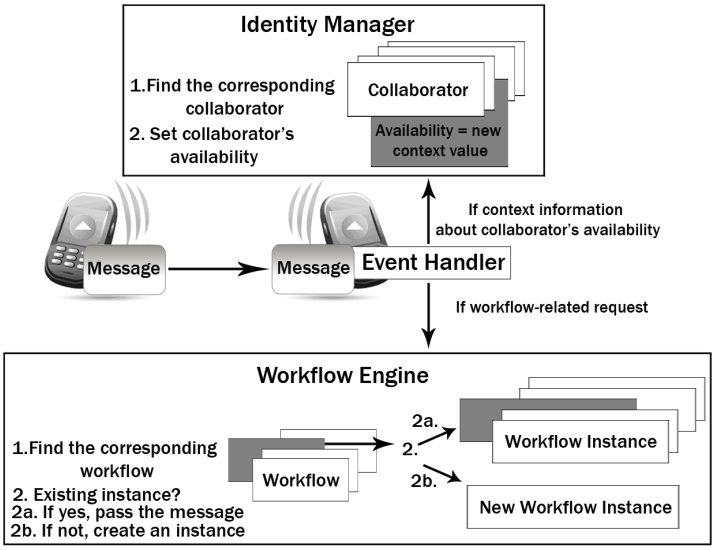

Figure 6.13: Incoming Message from Another Mobile Device

The first message type informs about collaborator's context, the second type informs about the state of a workflow instance and the last one contains data or content related to already running instance. If the message contains context information, the message is passed to Identity Manager which finds the collaborator and sets collaborator's availability to the context information included in the message. If the message contains workflow-related data, the particular workflow type identified by workflowID is found. Next, if the message is related to a running workflow instance then the message is passed to it. If not, the message contains information about new workflow instance that has been created on the other mobile device and the corresponding instance is created also on this device and message is passed to the created instance.

Based on the high-level analysis, the structure and format of messages has been designed and is described in next section.

6.7.2 MobWEL Communication Protocol

This section describes a protocol specification for exchanging structured information among peers who participate in the execution of MobWEL workflows. The message protocol simplifies the exchange and handling of message between peers. Firstly, the structure of messages is defined. Next, processing of messages is examined.

6.7.3 Message Structure

At high-level, there have been three types of messages identified. To design a uniformed message format suitable for all types, each message is examined and analysed in details:

1. Context-related message informing about collaborator's current availability: This message evidently needs to contain information about the workflow type that it belongs to, identified by *workflowID*, and the context information itself. The context information must be specified by the *context name* and *context value*. It needs to be clearly stated that the data in the message is context-related.

2. Message informing about the state of a workflow instance: To identify the workflow type which has been instantiated, *workflowID* has to be given. Additionally, the id of the workflow instance needs to be given in order to keep the execution of the workflow instance consistent across devices. There are three possible states the instances can be in: New, Running, and Terminated. The state needs to be indicated in the message. If the state is new or terminated, either new instance is created or running instance is terminated. If the state is set to running, it means that the message includes other, the workflow instance-related data.

3. Message includes workflow-related data: As outlined, if the state of instance is defined as running, the message contains some workflow instance-related data. There are three options regarding the data type considered in this work:

   - Text-based data that represent inputs or outputs of workflow activities;
   - Content and its metadata that is sent to other device;
   - Content-state to communicate progress.

Based on the analysis, the MobWEL message has been built by using the following constructs:

Workflow State - identifies the workflow state:
NEW - informing about a new workflow instance;
RUNNING - indicating that the message is related to an existing instance;
END - informing about termination of an instance;

Workflow ID and Workflow Instance ID - unique identifiers of the particular workflow case and workflow instance;

Sender's Phone ID - identifies the sender of the message (using the phone number);

Data Type - specifies the type of data that are included in the message:
CONTEXT - context-related data;
CONTENTSTATE - data that drives the correct execution of workflow;
VARIABLE - text-based data (inputs or outputs of the performed tasks);
CONTENT - information about content and its metadata;

Content ID and Content Object - if a piece of content is shared between devices, these fields are filled with corresponding information about content id and content object itself;

Action - specifies an action, task or activity that needs to be performed by other device;

Workflow Data[(name,value)] - actual workflow data;

Timestamp - a point of time at which the message has been created.

6.7.4 Message Processing

When a message from another device is received, the message is processed by following Algorithm 6.7.

6.8 Internal Cooperations

In previous section, the logical architecture of the MobWEL workflow management system has been presented. This section focuses on interactions between the individual components of the system. The goal is to emphasise context and content awareness in the MobWEL approach.

Internal interactions which are related to context awareness are illustrated in Figure 6.14.

As already described, MobWEL Parser registers the Context Definition part of a MobWEL workflow description in Context Manager. The *appKey* attribute in the XML context definition refers to workflowID. Context Manager extracts and manages only active contexts for the workflow type. After that, Context Manager registers the context definition in Context Provider.

Synchronous and asynchronous communication between the components is supported. Context Manager can query context information by specifying context name and workflowID. Based on this information, Context Provider responds and provides the most recent context value. On the other hand, when a context change occurs, Context Provider sends a notification about the event to Context Manager. Context Manager contains a listener that captures and handles the notification. Based on

Algorithm 6.7 Message Processing by Event Handler

Input: Incoming message m.
(Note: WE = Workflow Engine, IM = Identity Manager)
$workflowState \leftarrow m.WorkflowState$
$workflowID \leftarrow m.WorkflowID$
$workflowInstanceID \leftarrow m.WorkflowInstanceID$
if workflowState = ”NEW” **then**
  we.create newInstance{workflowID, workflowInstanceID}
  exit
end if
if workflowState = ”END” **then**
  we.remove workflowInstance{workflowID, workflowInstanceID}
  exit
end if
if workflowState = ”RUNNING” **then**
  $dataType \leftarrow m.DataType$
  if dataType = ”CONTEXT” **then**
    $contextName \leftarrow m.WorkflowData.name$
    $contextValue \leftarrow m.WorkflowData.value$
    im.setCollaboratorContext{m.SenderPhone ID), workflowID,
    contextName, contextValue}
    exit
  end if
  if dataType = ”CONTENTSTATE” **then**
    we.informContentState{workflowID, workflowInstanceID,
    m.ContentName,m.WorkflowData.value}
    exit
  end if
  if dataType = ”VARIABLE” **then**
    we.handleRequest{workflowID, workflowInstanceID, m.Action,
    m.WorkflowData.name, m.WorkflowData.value}
    exit
  end if
  if dataType = ”CONTENT” **then**
    $i \leftarrow 0$
    while exist m(Workflow Data) **do**
      $variableData(name, value)[i++] \leftarrow m.WorkflowData$
      next
    end while
    we.handleRequest{workflowID, workflowInstanceID, m.Action,
    variableData[], m.ContentName, m.ContentItem}
    exit
  end if
end if

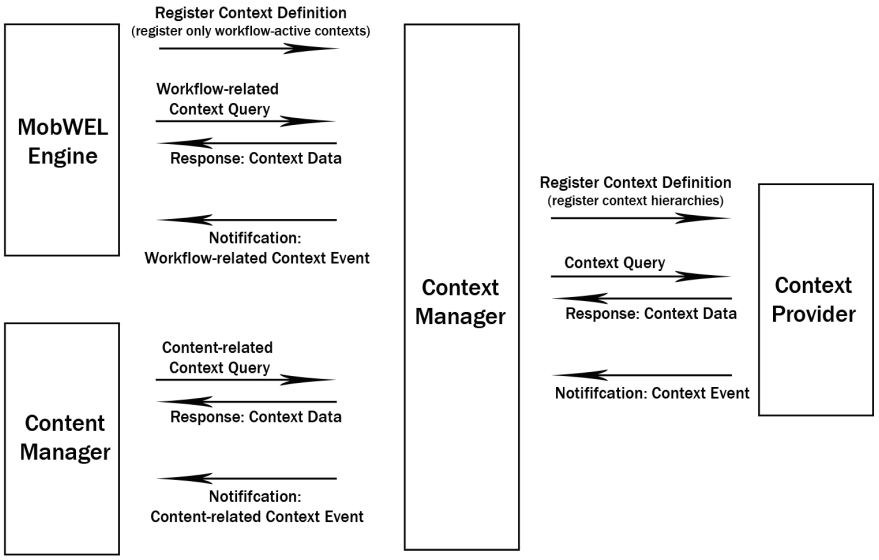

Figure 6.14: Context-Related Interactions

the consumption type, the notification is sent to either MobWEL Engine or Content Manager.

Context Manager also keeps the latest context values of all active contexts. Hence Workflow Engine or Content Manager can query context values by specifying context name and workflowID at any point of time. The important fact to notice is that all running workflow instances share the same context definition, therefore there is no need to keep different context information for each workflow instance. Basically Context Manager is workflow instance-agnostic. One context event notification is disseminated to all running instances. However, each instance might be in a different point of its execution, thus the same context event might be handled differently.

The second interaction model between MobWEL Engine and Content Manager, illustrated in Figure 6.15, relates to the exchange of content state-related information.

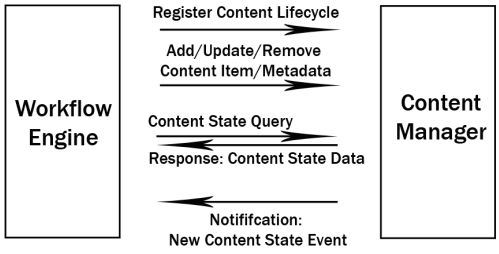

Figure 6.15: Content-Related Interactions

Similarly, when a MobWEL workflow description is parsed, the integrated content

lifecycles are registered within Content Manager. In contrast to Context Manager, Content Manager needs to know about workflow instances as it manages content objects processed in each instance. Content Manager processes content objects through their lifecycles and notifies MobWEL Engine when a content object has reached a new content state. MobWEL engine handles the notification related to the content state change accordingly. Content Manager also can respond to content state-related queries at real time.

With this section, the chapter is concluded and its summary is given next.

6.9 Summary

This chapter has described the logical architecture of the mobile MobWEL workflow management system, and outlined functioning of all its components. The MobWEL workflow management system is composed of five components: Context Provider, Context Manager, Content Manager, MobWEL Engine, and Peer-to-peer Interaction Manager.

With this chapter, the main part of this thesis in which the MobWEL workflow approach has been introduced is concluded. The next part of this thesis is dedicated to validation of the MobWEL workflow approach.

Part III

Validation

Chapter 7

From Design To Implementation

Contents

7.1 Introduction

In Chapter 5, the components of the MobWEL workflow management system have been described. To prove that the theory is functional, a prototype of the MobWEL workflow management system has been implemented on the Android platform. The Android platform has been chosen for the implementation because it is an open source platform written in a customised version of Java, and thereby, it is possible directly map UML classes from the system design into corresponding Java classes and structures. Furthermore, the platform provides powerful and rich communication facilities for implementation of peer-to-peer interaction model.

The MobWEL workflow management system is composed of a number of components. As outlined in Chapter 5, *Context Provider* can be implemented as an internal or external component. Because of the need to use *Context Provider* in two different projects, *Context Provider* has been implemented as an external component called **ContextEngine**. Thereby, the implementation has been driven by two different sets of requirements. The general design and implementation details of ContextEngine have been described in the work of Kramer et al. (2011).

Secondly, a prototype of the MobWEL workflow management system has been built and named **CAWEFA** (Context-Aware Workflow Engine For Android). The initial

version of CAWEFA has used ContextEngine as a context provider. However, for the validation purposes and easier workflow behaviour monitoring, Context Provider has been also added as an internal component to CAWEFA.

This chapter outlines some implementation details of the software prototypes. The remainder of this chapter is organised as follows. Section 7.2 describes the implementation details of the externalised Context Provider. CAWEFA, the prototype of the MobWEL workflow management system is explored in Section 7.3. Finally, this chapter is summarised in Section 7.4.

7.2 ContextEngine

This section describes the implementation details of ContextEngine.

7.2.1 Context Data Processing Layer

The self-contained **Context Component** has been implemented as an abstract class that provides the base for each context. However, there are different platform-specific context source types and ways for gathering raw context data. Based on the context source type and context acquisition principle, there have been four different subtypes of Context Component defined for the Android platform: **ListenerComponent**, **MonitorComponent**, **PreferenceChangeComponent** and **LocationContext**, see Figure 7.1.

A variety of built-in sensors such as a light sensor or accelerometer in mobile phones act as primary sources of environmental conditions. The **ListenerComponent** type has been designed to process raw context data captured by sensors. This component type implements SensorEventListener interface and has the following attributes: *sensorManager* needs to be specified in order to access the device's sensors, *sensorType* is used to specify the particular type of sensor, and *delayType* defines the rate sensor events are delivered at. To capture incoming notifications from sensors, the *onSensorChanged()* method has been implemented and this method is called when a change of context data occurs. *LightContext* for monitoring the light sensor values has been implemented as the concrete realisation of the listener type.

Another type of context component is **MonitorComponent**. There are context sources and services running on the Android platform which send broadcasts if a change occurs. To be able to receive the broadcasts, an Android-specific element called Broadcast Receiver needs to be registered within the context component. Thereby, MonitorComponent has been created for context data that are obtained by receiving broadcasts. Context broadcasters are, for example, bluetooth manager

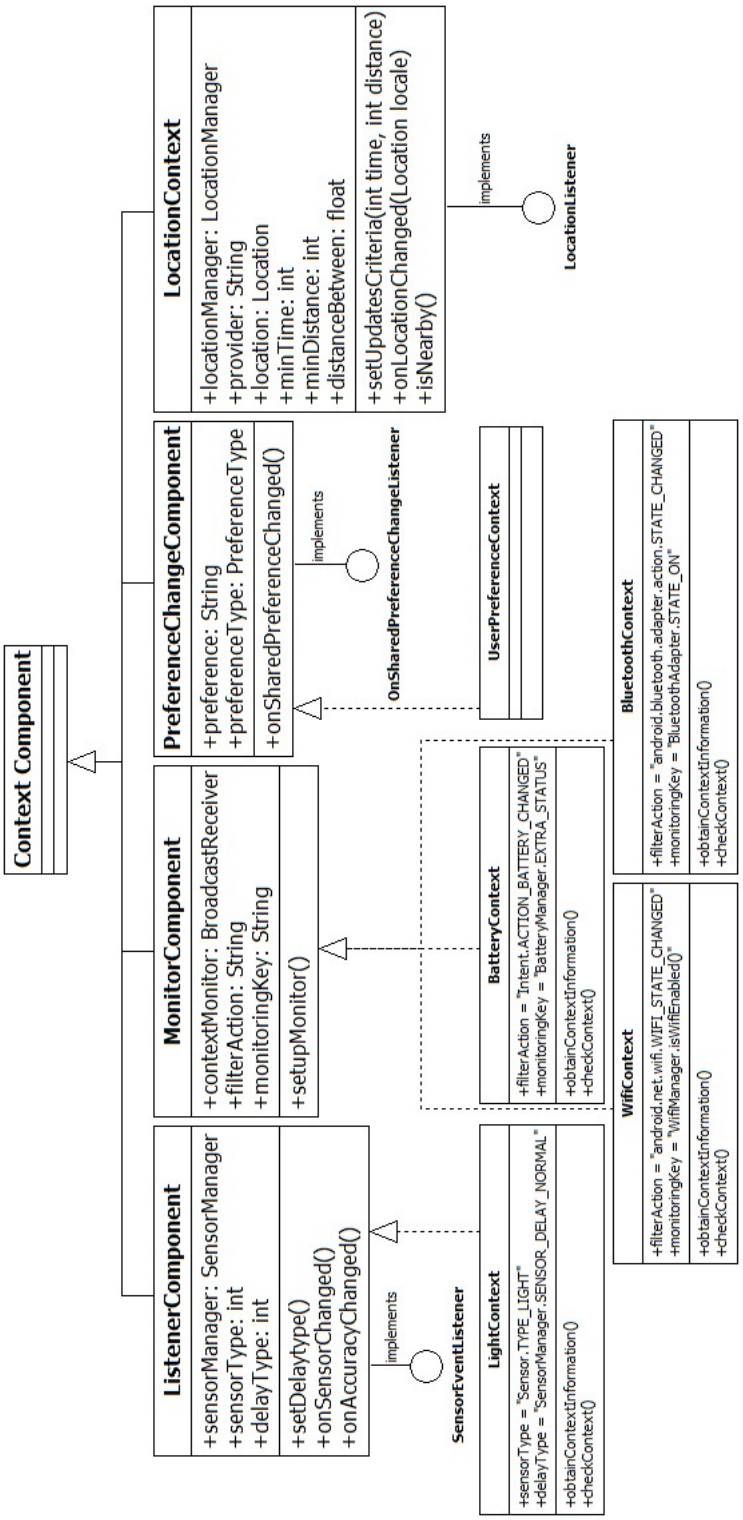

Figure 7.1: Component Types

that broadcasts bluetooth on/off status, wifi manager that informs about wifi status, or battery manager that informs about battery level. To demonstrate its use, there have been three corresponding realisations of this MonitorComponent type implemented: *WifiContext*, *BatteryContext* and *BluetoothContext*. Each component has the *contextMonitor* attribute which specifies the name of the Broadcast Receiver object, and the additional attributes, *filterAction* and *monitoringKey*, which are needed to filter out the relevant broadcasts. To support context data processing, the *obtainContextInformation* and *checkContext* are the only context-specific methods which have to be implemented for each realisation. The *obtainContextInformation* method has been designed to support derivation of high-level context information, for instance, if battery level value is 50, then the context information is MEDIUM. Once the high-level information is obtained, the *checkContext* method is called to determine whether there has been change in the context information. For example, if the previous value of battery level was 51 and the derived context information was as MEDIUM, then there is no need to broadcast this is context change as the derived context information is MEDIUM again.

The third type called ***PreferenceChangeComponent*** is defined for user preferences. On Android, the component type implements the *OnPreferenceChangeListener* interface. Changes in user preferences are obtained by invoking the *onSharedPreferenceChanged()* method. Each preference is described by the *preference* attribute which defines its name, identifier or key. The data type such as string, int or boolean is provided by the *preferenceType* attribute. This type of ContextComponent is used to monitor user preferences, thus the *UserPreferenceContext* represents its concrete realisation and can be used to obtain the user preference for number of reviews or rating.

Finally, the ***LocationContext*** component type is implemented to deal with data obtained from GPS. The following attributes has been added to the type: *locationManager* to provide access to the system location services, *provider* to specify name of the GPS location provider, *location* to indicate the last known location, *minTime* to specify minimum time interval between location updates in milliseconds, *minDistance* to specify the minimum distance between location updates in meters and *distanceBetween* to define the area of the nearby locations from current location in meters. To control the frequency of location updates, the *setUpdatesCriteria(int time,int distance)* method is used. The *onLocationChanged* method is implemented to receive notifications from the system location services. The context-aware application can list a number of locations in the XML context definition, the purpose of the *isNearby* method is to return those locations that can be found within the distance determined by *distanceBetween* parameter in the nearest-first order.

This section has demonstrated the implementation of the particular context components to obtain context data. These context component realisations are packed with the ContextEngine package, however, there might be other context variables defined later when ContextEngine is running. ContextEngine supports dynamic class loading which gives the programmer *the ability to install software components at runtime* (Liang & Bracha, 1998). Wissen et al. (2010) pointed out that the benefits of using DCL include the capability to lazy load classes, reducing memory usage; the ability to instantiate a component without explicit referencing, allowing more generic code; and finally allow the programmer to add additional context components to the engine without needing to alter or access the engine source code.

Broadcast Engineering

Apart from the atomic context components, the **Composite Component** has been implemented to support context aggregation. This component has to interact with its child contexts and be regularly informed about child context changes. The interaction is supported by using the *sendNotification* and *onReceive* methods shown in Listing 7.1.

Listing 7.1: Implementation of the *sendNotification* and *onReceive* methods

```
//specify the name of the intent
public static final String CONTEXT_INTENT = "uk.ac.uwl.mdse.contextengine.CONTEXT_CHANGED";

//method to send notification
public void sendNotification() {
        Intent intent = new Intent();
        intent.setAction(CONTEXT_INTENT);
        intent.putExtra(CONTEXT_NAME, name);
        intent.putExtra(CONTEXT_DATE, Calendar.getInstance().toString());
        intent.putExtra(CONTEXT_INFORMATION, contextInformation);
        sendBroadcast(intent);
}

//method to receive notification
IntentFilter filter= new IntentFilter(CONTEXT_INTENT);
private void setupMonitor() {
  contextMonitor = new BroadcastReceiver() {
  @Override
    public void onReceive(Context c, Intent in) {
        String context = in.getExtras().getString(CONTEXT_NAME);
        String value = in.getExtras().getBoolean(CONTEXT_INFORMATION);
        ...
           checkContext();
        }
         }
     }
  };
context.registerReceiver(contextMonitor, filter);
}
```

The *sendNotification()* method is implemented in each component and is invoked when a change of context information occurs. Generally, messages in Android applications which facilitate run-time binding between objects are called intents. A custom, ContextEngine-specific, intent for the internal context broadcasting purpose has been implemented in the *Context Component* class. Intents are characterised by an action name, and the intent in ContextEngine has been named as

$$"uk.ac.uwl.mdse.contextengine.CONTEXT\_CHANGED" \quad .$$

The listing also shows the implementation of the *sendNotification()* method, and how various data such as context name, context information, and date are associated with the intent, and then broadcasted.

Once the messages are broadcasted, a broadcast receiver must be registered within the components that are interested in receiving the messages. The *setupMonitor* method is used to register the broadcast receiver. The receiver is implemented with a filter that specifies which types of intents should be received. The filter is set for the same intent, defined with the action:

$$"uk.ac.uwl.mdse.contextengine.CONTEXT\_CHANGED" \quad .$$

The receiver implements the *onReceive* method in which the data is extracted and further processed as needed.

This section has explored messages exchange internally within ContextEngine. However, ContextEngine has to broadcast relevant context information to mobile applications. A supportive mechanism for external message broadcasting is explored next.

7.2.2 ContextEngine Manager

The ContextEngine provides context provisioning services to third-party context-aware applications, therefore, it needs to be capable of performing long running operations in background. The **ContextEngine Manager** element has been implemented as an Android-specific element called Service.

ContextEngine Manager has two roles: *a*) to manage lifecycles of context components; and *b*) to manage communication and interaction with external applications.

ContextEngine publishes its services through two interfaces declared by using Android Interface Definition Language (AIDL), see Listing 7.2. The first interface enables any mobile application running on the same mobile device to deploy their context definitions by calling the *setupContexts(String path)* method. The entry parameter of the method is the actual path to the context definition XML document. The referred XML documents must conform to the XML schema for context definition defined in Chapter 4. The other method in the first interface, the *register-*

ContextPath(String path) method, is used for dynamic class loading. The second interface supports synchronous communication and context querying. An application can query ContextEngine at any time and obtain a context value of any context specified by its name.

Listing 7.2: Interfaces

```
Interface 1:
interface IContextsDefinition {
  //to add context definition in XML document:
  void setupContexts(String path);
  //to add external context components:
  boolean registerContextPath(String path);
}
Interface 2:
interface ISynchronousCommunication {
  String getContextValue(in String componentName);
}
```

Asynchronous communication is supported through messages broadcasting. ContextEngine Manager has implemented a broadcast receiver that receives internally broadcasted messages with the action name:

$$uk.ac.uwl.mdse.contextengine.CONTEXT\_CHANGED$$

The messages are further broadcasted to external applications by using the *sendBroadcastToApps()* method. The body of the method is the same as the body of the *sendNotification* method, however, an external intent is specified for it. The action name for the external intent is

$$"uk.ac.uwl.mdse.contextengine.REMOTE\_SERVICE"$$

Mobile application can receive the broadcasted context messages if they set up a listener and a filter for this external action.

In this section the implementation details of ContextEngine have been described. Next section gives the implementation details of CAWEFA.

7.3 CAWEFA

This section describes the distributed mobile workflow management system called CAWEFA. The system has not been built from scratch. An open source BPEL workflow engine named Sliver[1] has been adapted and extended to run on the Android platform. The architecture of Sliver execution engine has been presented in Chapter 2. The separation of communication and processing concerns as initially outlined in Sliver's architecture remains same in CAWEFA. However, the communication layer has been extended to exchange direct messages between peers and the processing layer has been extended to support the execution of MobWEL workflows. In addition, components for context management and content management have been built. Components are encapsulated, and each component exists autonomously from other components which means that the internal functioning of any component is not visible to other components and they communicate with each other through interfaces. The singleton pattern has been used to restrict the instantiation of each component to one object.

A CAWEFA service used for the interaction with mobile applications is explored in next section.

7.3.1 CAWEFA Service

Sliver's BPEL Server has been replaced by two components: *CAWEFA Service* and *Workflow Manager*. The *CAWEFA Service* class extends an Android base component called *Service*, implements its abstract methods and overrides its lifecycle methods. This service runs as a background ongoing process and serves as entry point for communication with mobile applications. Mobile applications do not directly call methods in *CAWEFAService* but connect to remote interfaces specified by using AIDL (Listing 7.3). Mobile applications can add or remove their defined MobWEL workflow processes by connecting through the *IBusinessProcessDefinition* interface. The *IWorkflowClient* interface has been implemented to handle other workflow-related requests, for example, for workflow instantiation.

Listing 7.3: CAWEFA Interfaces

```
interface IBusinessProcessDefinition {
  void addBusinessProcess(in String operationNamespace, String processResource);
  void removeBusinessProcess(in String operationNamespace);
}
interface IWorkflowClient {
  int handleRequest(in String operationNamespace, in Object message);
}
```

[1] http://mobilab.cse.wustl.edu/projects/sliver/

When the *addBusinessProcess* method is invoked, the MobWEL workflow definition is parsed and converted into internal data structures as described next.

7.3.2 MobWEL Process Parsing

The BPEL parser of the Sliver engine has been adapted and extended to parse MobWEL XML documents. The *Process*[1] class in Sliver has been adapted to handle the additional tags and parse the whole MobWEL specification correctly, see Listing 7.4.

Listing 7.4: The adapted and extended constructor of the Sliver's *Process* class

```
public Process(XmlPullParser parser)
  throws XmlPullParserException, IOException, MalformedDocumentException
  {
    parser.require(XmlPullParser.START_TAG, namespace, "process");
    name = parser.getAttributeValue(null, "name");
    if(name == null)
       throw new MalformedBPELException(parser,
         "<process> must specify name");
     ...
    anyAttribute = parser.getAttributeValue(null, "anyAttribute");
    if(anyAttribute == null)
      throw new MalformedBPELException(parser,
        "<process> must specify anyAttribute");
    ...
    // Read and validate the process's name and attributes

    parser.nextTag();

    partnerLinks = new PartnerLinks(parser);
    // Read the <partnerLinks> child
    partners = new Partners(parser);
    // Read the <partners> child
    collaboratorsGroups = new CollaboratorsGroups(parser);
    //Read the <CollaboratorsGroups> child
    parseCollaboratorsGroups(collaboratorsGroups);
    //parse collaboratorsGroups to Identity Manager
    variables = new Variables(parser);
    // Read the <variables> child
    lifecycles = new Lifecycles(parser, variables);
    //Read the <lifecycles> child
    parseContentLifecycles(lifecycles);
    //parse contentLifecycles to Content Manager
    contextDefinitions = new ContextDefinitions(parser);
    //Read the <contextDefinitions> child
    parseContextDefinitions(contextDefinitions);
    //parse contextDefinitions to Context Manager

    correlationSets = new CorrelationSets(parser);
    // Read the <correlationSets> child
    ScopeData scopeData = new ScopeData(name, partnerLinks,
              variables, correlationSets, suppressJoinFailure);
```

[1]The name of the *Process* class has not been renamed to *'Workflow'* in CAWEFA. The name is preserved due to backward compatibility with Sliver engine.

```
    scopeData.mobwelPartners = partners;
    Enumeration e = lifecycles.getLifecycles();
    while(e.hasMoreElements()){
        Lifecycle lifecycle = (Lifecycle)e.nextElement();
        scopeData.contentVariables.put(lifecycle.getVariable(), lifecycle.contentLifecycle)
            ;
    }
    // Create a new scope with the scope data we've read so far

    faultHandlers = new FaultHandlers(parser, scopeData);
    // Read the <faultHandlers> child

    eventHandlers = new EventHandlers(parser, scopeData);
    // Read the <eventHandlers> child

    activity = Activity.parse(parser, scopeData);
    startActivities = activity.getStartActivities();
    interactionActivities = activity.getInteractionActivities();
    // Read the root activity

    parser.require(XmlPullParser.END_TAG, namespace, "process");
    // Read the closing tag
        }
```

MobWEL workflows encoded in XML documents are converted into executable Java workflow objects. Each tag in the BPEL specification is represented in Sliver by a corresponding Java class, e.g., the $<$*variable* $>$ tag is mapped to the *Variable* class or the $<$*variables* $>$ tag is mapped to the *Variables* class. There are two different categories of tags. The first category is formed by tags for global declarations, and the second category for the process control flow activities. As the set of global declarations has been extended in MobWEL, parsing of content lifecycles, group identification and context definition tags has been added to the *Process* constructor. The global declaration tag parsing is shown next.

Global Declaration Tag Parsing

To demonstrate parsing of a global declaration tag, a simplified version of the *Lifecycles* class is shown in Listing 7.5.

Listing 7.5: The *Lifecycles* class

```
public class Lifecycles {
  public final Hashtable lifecycles = new Hashtable();
  Lifecycles(XmlPullParser parser, Variables variables) throws
    XmlPullParserException, IOException, MalformedDocumentException
  {
    if(!parser.getName().equals("lifecycles"))
        return;
    // If there's no <lifecycles> block, then don't parse anything

    parser.require(XmlPullParser.START_TAG, namespace, "lifecycles");
    //requires <lifecycles> opening tag
```

```
    parser.nextTag();
    //move cursor to start of first child

    Vector links = new Vector();
    while(parser.getName().equals("lifecycle"))
    links.addElement(new Lifecycle(parser, variables));
    // parse every child element, the <lifecycle> tag

    if(links.isEmpty())
      throw new MalformedBPELException(parser,
       "<lifecycles> must specify at least one <lifecycle>");
    // Validate that there's at least one lifecycle

    Enumeration e = links.elements();
    while(e.hasMoreElements())
    {
       Lifecycle next = (Lifecycle)e.nextElement();
       lifecycles.put(next.getName(), next);
     }
     //all lifecycles are stored in hashtable

     parser.require(XmlPullParser.END_TAG, namespace,
                        "lifecycles");
     //requires </lifecycles> closing tag
     parser.nextTag();
     //moves cursor to the next tag
  }
  public Lifecycle getLifecycle(String name)
  {
     return (Lifecycle)lifecycles.get(name);
  }
  public Enumeration getLifecycles()
  {
     return lifecycles.elements();
  }
}
```

So all global declarations tags are similarly parsed into corresponding Java classes. Parsing of activity tags is slightly different, as is explored next.

MobWEL Activity Parsing and Implementation

Activity tags such as contentActivity, sequence, assign etc. represent actual executable actions, thereby their parsing and implementation differs from the global declaration tags. In Sliver, the abstract *Activity* class is implemented that provides *parse* method. The *parse* method is called when an activity tag is parsed. Each control flow activity extends the *Activity* class, and is mapped into a corresponding Java class, for example, the *sequence* tag is mapped into the *Sequence* class which extends the *Activity* class.

To support the runtime execution of a control flow activity, the *ActivityInstance* class is created in Sliver which implements the *execute* method. this method returns

a signal. The signal (COMPLETED, EXITED, CANCELED, CLOSED, COMPENSATED) is sent among activity instances. A typical case to illustrate the use of the signal pattern is a situation where a parent activity instance can only proceed when its child activity instance(s) have been completed successfully, or at least achieved a certain state. The *execute* method is implemented with respect to the global workflow instance execution which means that it does the bookkeeping regarding the instance's state, and the actual activity execution is performed by invoking the *executeImpl()* method. The *execute* method in a given activity instance is invoked when all activity's precedors have been completed. Consequently the *executeImpl* method is called from the inside of the *execute* method. All activity instance classes override the *executeImpl* method so the actual action specified for the given activity can be performed.

In Sliver, Java classes for all BPEL activities have been created, and Sliver supports 14 workflow patterns. In CAWEFA, only parsing of *contentActivity* and *interactionActivity* tags has had to be added and supported. To extend the set of BPEL activities, the *MobWELActivityExtension* class has been created in which the additional activities are specified. To illustrate the implementation of a MobWEL activity, the implemented *ContentActivity* class is presented in Listing 7.6.

Listing 7.6: The ContentActivity class

```
public class ContentActivity extends CawefaActivity {
  private final String name;
  private final String action;
  private final String element;
  private final String value;
  String var;
  VariableSpecification contentVariable;

  public ContentActivity(XmlPullParser parser, ScopeData scopeData) throws IOException,
      XmlPullParserException, MalformedDocumentException
  {
    super(parser, scopeData, "contentActivity");
    parseStartTag();
    name = parser.getAttributeValue(null, "name");
    action = parser.getAttributeValue(null, "action");

    if (!(action.equals("add") || action.equals("update") || action.equals("remove")))
        throw new MalformedBPELException(parser,
            "<ContentActivity> must specify action:add,update,remove");

    var = parser.getAttributeValue(null, "variable");
    contentVariable = getVariableSpecification(var);
    element = parser.getAttributeValue(null, "element");
    value = parser.getAttributeValue(null, "value");

    parser.nextTag();
    parseStandardElements();
    parseEndTag();
    parser.nextTag();
```

```
  }

  @Override
  public ActivityInstance newInstance(ProcessInstance processInstance) {
              return new ActivityInstance(this, processInstance){
                      protected Signal executeImpl()
  {
    ContentManager cm = processInstance.getContentManager();
    Object varvalue = processInstance.getObjectVariable(value);
    ContentLifecycle lifecycle = processInstance.process.getContentLifecycle(var);
    if (action.equals("add")){
        cm.addContent(processInstance, var, lifecycle, String.valueOf(varvalue));
    }
        if (action.equals("update")){
    cm.updateAttribute(processInstance.processInstanceID, var, lifecycle, element, String.
        valueOf(varvalue));
    }
    if (action.equals("remove")){
        cm.removeContent(processInstance, var, lifecycle);
    }
    return Signal.COMPLETED;
   }

   protected synchronized void cancelImpl(){
      ...
    return Signal.CANCELED;
   }
   };
  }

  protected VariableSpecification getVariableSpecification(String variableName)
    throws MalformedBPELException
  {
    if(variableName == null)
        return null;
    VariableSpecification variableSpec =
        scopeData.getVariable(variableName);
    if(variableSpec == null)
        throw new MalformedBPELException(parser, "No variable named "
            + variableName + " in scope");
            return variableSpec;
    }
}
```

The parsing process also deploying of context definition to context provider. The interaction between CAWEFA and ContextEngine is explored next.

7.3.3 Interaction with ContextEngine

If Context Provider is implemented as an external component and its service is running, the *ContextDefinitionDeployer* class in Context Manager can be used to deploy the context definition model to it and use its features. The features are specified in AIDL files which have been provided by ContextEngine and imported in CAWEFA. The files generate a Java interface and an inner *Stub* class. The class is used to

create a remotely accessible object, and the deployer can use it to invoke a method. The *IBinder* interface is the base of the remoting protocol in Android and *IBinder* is returned to any caller that binds to the ContextEngine *Service*. The *ContextDefinitionDeployer* class binds to the ContextEngine service and invokes the *setupContexts* method in order to pass the path to the context definition file, as shown in Listing 7.7.

Listing 7.7: ContextDeployer

```
IContextDefinition contextService;
String contextDefSource = "data/mnt/contextDefinition.xml";
private ServiceConnection mConnection = new ServiceConnection() {
  public void onServiceConnected(ComponentName className, IBinder service) {
  try {
    contextService = IContextDefinition.Stub.asInterface(service);
    try {
        contextService.setupContexts(contextDefSource);
    } catch (RemoteException e) {
        e.printStackTrace();
    }
    } catch (NotFoundException e) {
        e.printStackTrace();
    }
}
```

The ContextListener class has been implemented to monitor messages broadcasted by ContextEngine. If the broadcasted application key matches any workflow key, the context value of corresponding context component is changed, and Context Manager is informed about the change in order to deal with the context information further. The implementation details of the broadcast receiver within the ContextListener class is shown in Listing 7.8.

Listing 7.8: ContextListener

```
private static final String CONTEXT_INFORMATION = "context_information";
private static final String CONTEXT_NAME = "context_name";
private static final String CONTEXT_DATE = "context_date";
private static final String CONTEXT_APPLICATION_KEY = "context_application_key";
private void setupContextMonitor() {
  contextMonitor = new BroadcastReceiver() {
  @Override
  public void onReceive(Context context,Intent intent) {
    if (intent.getAction().equals("uk.ac.tvu.mdse.contextengine.REMOTE_SERVICE")) {
      try{
        Bundle bundle = intent.getExtras();
    String workflowKey = bundle
        .getString(CONTEXT_APPLICATION_KEY);
    String contextName = bundle
        .getString(CONTEXT_NAME);
    String contextValue = bundle
        .getString(CONTEXT_INFORMATION);
    String contextDate = bundle
        .getString(CONTEXT_DATE);
    MobWELContext activeContextObject = null;
```

```
    if (ctxManager.workflowIDs.contains(workflowKey)) {
    ArrayList<MobWELContext> activeContexts = (ArrayList<MobWELContext>) ctxManager.
        activeContexts.get(workflowKey);
    for(MobWELContext mc: activeContexts){
        if (mc.contextName.equals(contextName)){
        //set new context value
            mc.lastContextChangeDate = contextDate;
            mc.lastContextValue = contextValue;
            activeContextObject = mc;
        }}
       //inform context manager that a context change occured
       ctxManager.onContextChange(activeContextObject);
    }}
    catch (Exception ex){
    ...
    }}}};
    appContext.registerReceiver(contextMonitor, filter);
}
```

This section explored the interaction with ContextEngine. CAWEFA is a system running on each mobile device, thus these systems have to interact. In next section, the implementation details of peer-to-peer interaction and message exchange are described.

7.3.4 Peer-to-Peer Message Exchange

The MobWEL message serialisation has been supported by using Protocol Buffers - Google's data interchange format. Using Protocol Buffers is a way of encoding structured data in an efficient, extensible, language-neutral and platform-neutral format. Its advantage over XML-based message format is that protocol buffers are smaller, faster and simpler [1], therefore more suitable for frequent message exchange between mobile devices. The message format, called *MobwelData*, is specified by defining protocol buffers message type in a .proto file as shown in Listing 7.9. The format is simple and contains all message constructs. Moreover, fields are specified as required, optional or repeated. The fields such as *workflowID*, *workflowInstanceID*, *senderPhoneID* and *msgtype* are required and values for the fields must be provided in all messages. Optional fields are, for instance, *contentID* and *workflowState*. The values for the fields are provided only when content is sent between devices. Workflow data is specified by its *dataName*, and *dataValue*. As a message can contain numerous data, the *workflowData* field of *Data* type is specified as repeated. In addition, a number of *MobwelData* messages can be packed and sent together by using the *MobwelDataCollection* message.

[1] https://developers.google.com/protocol-buffers/docs/overview

Listing 7.9: Protocol buffer data structure described by using .proto file syntax

```
message MobwelData {
 enum State {
    NEW = 0;
    RUNNING = 1;
    END = 2;
  }
  enum MessageType {
    CONTEXT = 0;
    CONTENTSTATE = 1;
    VARIABLE = 2;
    CONTENT = 3;
  }
  required MessageType msgtype = 1;
  required string processInstanceID = 2;
  required string processID = 3;
  required string senderPhoneID = 4;
  optional State workflowState = 5;
  optional string action = 6;
  optional string timestamp = 7;

  optional string contentID = 8;
  optional bytes contentItem = 9;

  message Data {
        optional string dataName = 1;
        optional string dataValue = 2;
  }
  repeated Data workflowData = 10;
}
message MobwelDataCollection {
  repeated MobwelData mobwelData = 1;
}
```

A data access class can be easily generated, thus protocol buffers are easier to use programmatically. *MobwelDataProtos*, a class generated by the compiler from the *mobweldata.proto* file, is used to automatically serialise and deserialise workflow messages. The creation of a message is demonstrated in Listing 7.10.

Listing 7.10: Create a message

```
public void createContextMessage(){
  MobwelData mobwelData;
  mobwelData =
    MobwelData.newBuilder()
      .setMsgtype(msgType)
      .setWorkflowID("workflowID")
      .setWorkflowInstanceID("workflowInstanceID")
      .setSenderPhoneID(phoneNo)
      .setWorkflowState(MobwelData.MessageType.CONTEXT)
      .setTimestamp("now")
      .addWorkflowData(
        MobwelData.Data.newBuilder()
          .setDataName("dataName")
          .setDataValue("dataValue"))
      .build();
}
```

There have been two assistants implemented to support the exchange of messages among collaborators: *BluetoothAssistant*, and *BinarySMSAssistant*. *BluetoothAssistant* supports the transfer of messages over Bluetooth. The bluetooth Mac address has to be known to connect two devices and enable data transfer. This data transfer method can be used if collaborators are situated within a particular distance, for example, located in a meeting room.

BinarySMSAssistant enables sending binary data messages to recipients identified by their phone numbers. Sending binary data messages is easier and more convenient approach as the messages can go to multiple collaborators in a short time.

These two assistants are concrete realisations of the message exchange assistant to demonstrate the message exchanges between peers, however, other protocols and interaction techniques are supported in CAWEFA.

This section concludes the description of the most significant concepts and details implemented in CAWEFA. A chapter summary is provided next.

7.4 Summary

This chapter presented the implementation details of two prototypes built for the Android platform. ContextEngine has been implemented as an external context provisioning platform that provides its services to other context-aware mobile applications running on the same device. CAWEFA is a workflow management system that interprets, manages and executes MobWEL workflows. CAWEFA has been built on top of the existing BPEL engine called Sliver, and uses context provisioning services provided by ContextEngine. Both prototypes will be released as open source software.

Chapter 8

Evaluation

Contents

8.1 Introduction

So far, the MobWEL workflow language has been presented and details about the designed and developed workflow management system have been given. This chapter focuses on the final evaluation of the produced research artifacts. The validation goal is to determine whether the MobWEL workflow language is a suitable technology to define mobile distributed context-aware content-centric workflows. The usage scenario described in Chapter 2 is used for the validation purposes and experimentation.

The remainder of this chapter is organised as follows. An overview of the experimentation strategy is outlined in Section 8.2. The construction of a MobWEL workflow process by using the work pattern described in the usage scenario is discussed in Section 8.3. The constructed MobWEL workflow definition is deployed to CAWEFA and the experiment is conducted by workflow instantiating and monitoring the behaviour of running workflow instances. The results of the experimentation are presented in Section 8.4 and findings are discussed in Section 8.5. This chapter is summarised in Section 8.6.

8.2 Experimental Design

This section presents the design of the experimentation process in which the workflow presented in the usage scenario is constructed and instantiated.

The overview of the experimentation strategy is depicted in Figure 8.1.

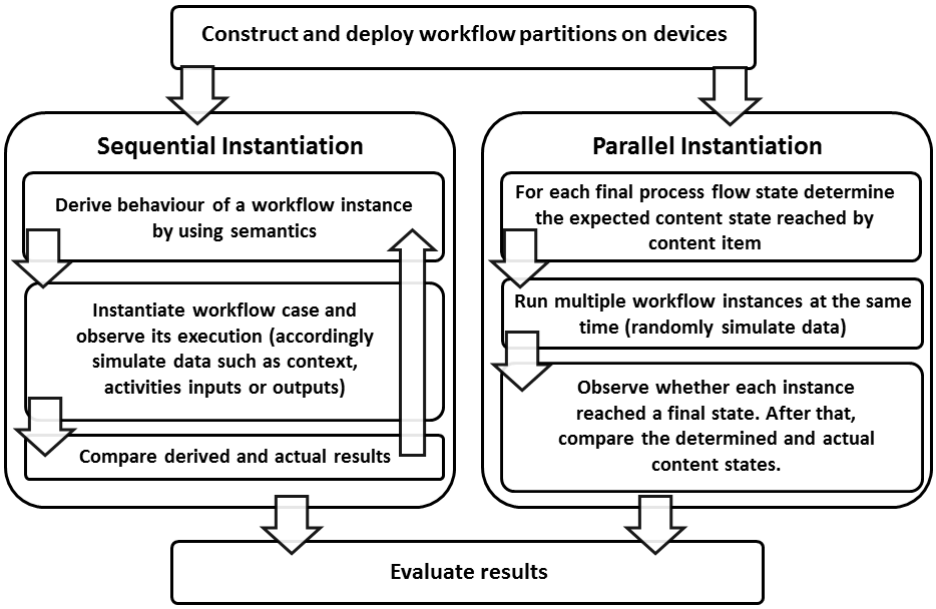

Figure 8.1: Experimentation Strategy

Firstly, the MobWEL workflow definition for the usage scenario is constructed by following the design methodology provided in Appendix A. Non-probability purposive sampling is used to select what context variables influence the workflow process.

Based on the workflow definition, quantitative data such as a number of execution paths, or a number of monitored contexts is collected. The data outlines how many different variants of the workflow can be built. After that, the MobWEL workflow definition is deployed to the MobWEL workflow management system. The workflow process is instantiated and the behaviour of each instance is monitored. Two strategies are used for workflow instantiation:

1. *Sequential Instantiation:* Workflow is instantiated sequentially, which means that only one instance runs at any given point of time. That enables to simulate various context data and monitor the behaviour of each instance from its start to the end. The expected behaviour of the instance is derived by using MobWEL semantics, and then the expected behaviour is compared with its actual behaviour.

2. *Parallel Instantiation:* In this strategy, workflow instances run in parallel. The workflow process can end at different final process flow states, and the picture

can reach different content object states. While the workflow instances are running, context situations are randomly generated. There are two aspects monitored: *a*) whether each workflow instance reached a final state; *b*) if a final process state has been reached, whether the reached content state corresponds to it.

To observe workflow instantiation and execution, the actual steps in the execution process have to be visualised. Data logging is a method that is used to print out the required steps on the screen. After the data are collected, the results are analysed and presented.

This section has described the experimentation strategy. Next section outlines the experimentation process.

8.3 Scenario-Based Evaluation

Experimentation is a validation method that involves two activities: construction of a MobWEL workflow process for the usage scenario, and running its workflow instances. By constructing the workflow process, completeness and language expressiveness are verified. By workflow instantiation, the workflow executability and change realisation are monitored and evaluated.

8.3.1 Construction of a MobWEL workflow process

This section describes the MobWEL workflow process defined for the usage scenario. The basic high-level work flow described in the scenario is shown in Figure 8.2.

This workflow has been used as a base for the construction of the MobWEL workflow process.

Workflow-Specific Context Model

To validate the workflow-specific context modelling approach, the several context situations have been modelled. The following factors have influenced the selection of context situations:

Group A - complexity of the context situation (A_1: atomic context, A_2: composite context);

Group B - derivation of context values (B_1: context with pre-defined values, B_2: context with derived high-level values, B_3: composite context with values derived from its children values);

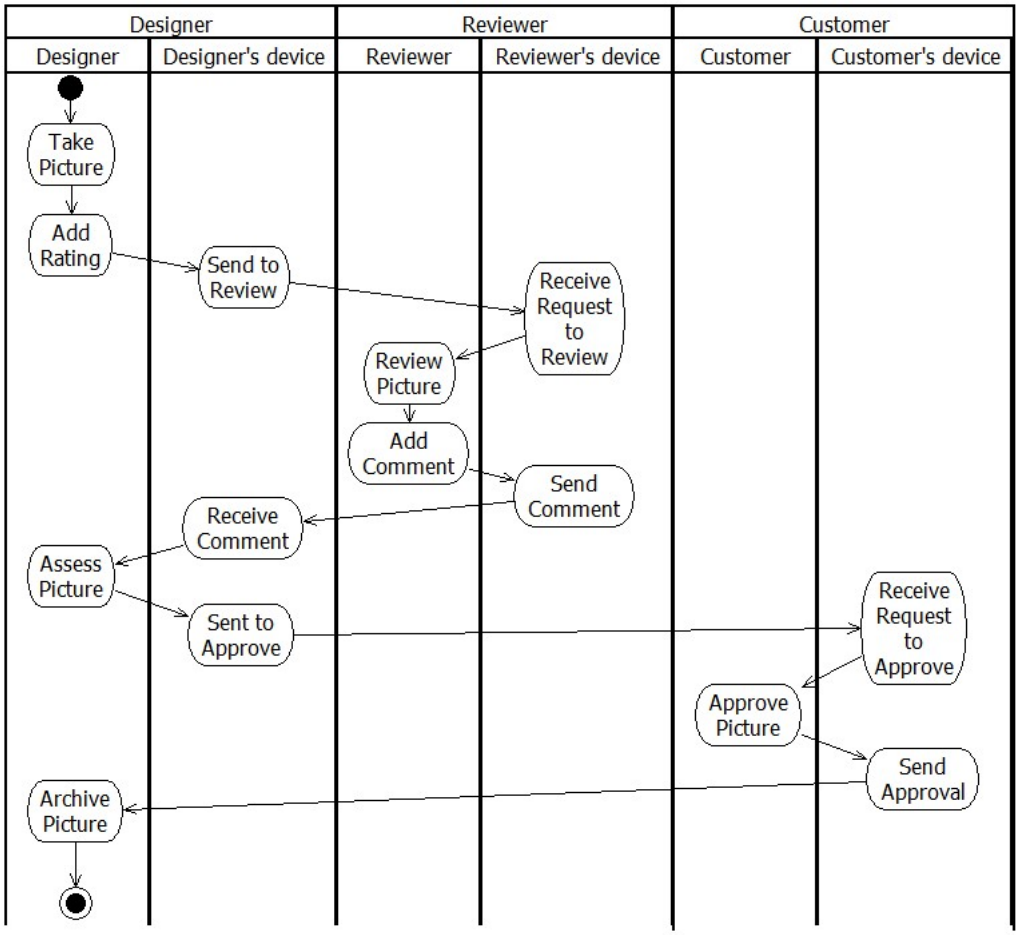

Figure 8.2: The Basic Work Flow in the Usage Scenario

Group C - workflow-active or auxiliary context (C_1: workflow-active context, C_2: not workflow-active context);

Group D - types of context information and source (D_1: user preference, D_2: location, D_3: device-related context);

Group E - context association type (E_1: communication, E_2: content (E_{21}: content metadata, E_{22}: content behaviour), E_3: workflow, E_4: collaboration);

Group F - collaborator's role (F_1: designer, F_2: reviewer, F_3: customer);

Based on the factors, the following context situations have been derived (the object models are provided in Appendix C):

- BatteryContext (A_1, B_2, C_2, D_3, E_1) is related to all roles;
- BluetoothContext (A_1, B_1, C_2, D_3, E_1) and WifiContext(A_1, B_1, C_2, D_3, E_1) are related to all roles;

- DataSync (A_2, B_3, C_2, D_3, E_1) and Connectivity (A_2, B_3, C_1, D_3, E_1) are related to all roles;
- NoOfReviewsUP (A_1, B_1, C_1, D_1, E_{22}) and RatingUP (A_1, B_1, C_1, D_1, E_{22}) are related to the role of designer;
- AtWork (A_1, B_1, C_2, D_1, E_4), Status (A_1, B_1, C_2, D_1, E_4), and Availability (A_2, B_3, C_1, D_1, E_4) are related to the role of designer and reviewer;
- AddCommentUP (A_1, B_1, C_1, D_1, E_3) is related to the role of reviewer;
- LocationContext (A_1, B_2, C_1, D_2, E_{21}) is related to the role of designer.

The selected sample of contexts ensures that there is at least one representant for each group. The sample has been chosen intentionally. Predominantly, most of them have been defined as a user preference context type because values of user preferences can be easily simulated and changed while workflow instances are running.

Group Identification Model

There are three roles involved in the workflow: designer, reviewer, and customer. In each running workflow instance, there can be only one actor performing the role of designer and one actor performing the role of customer. However, the role of reviewer can be performed by numerous actors. To keep the scope of the experiment manageable, three actors play the role of reviewer.

Context-Aware Content Lifecycle Model

There is one content object processed in the workflow process: picture. A context-aware content lifecycle is constructed for the picture, see Figure 8.3. The lifecycle outlines the states the picture moves through during its lifetime.

The actual content states are derived from the workflow process control flow, in particular, from workflow activities that modify the content object. The logical content states are created to enrich the lifecycle and enable integration of context conditions. Both types of context conditions are incorporated in the picture lifecycle. The conditions depend on context variables defined in previous section. The object models for the picture lifecycle are presented in Appendix C.

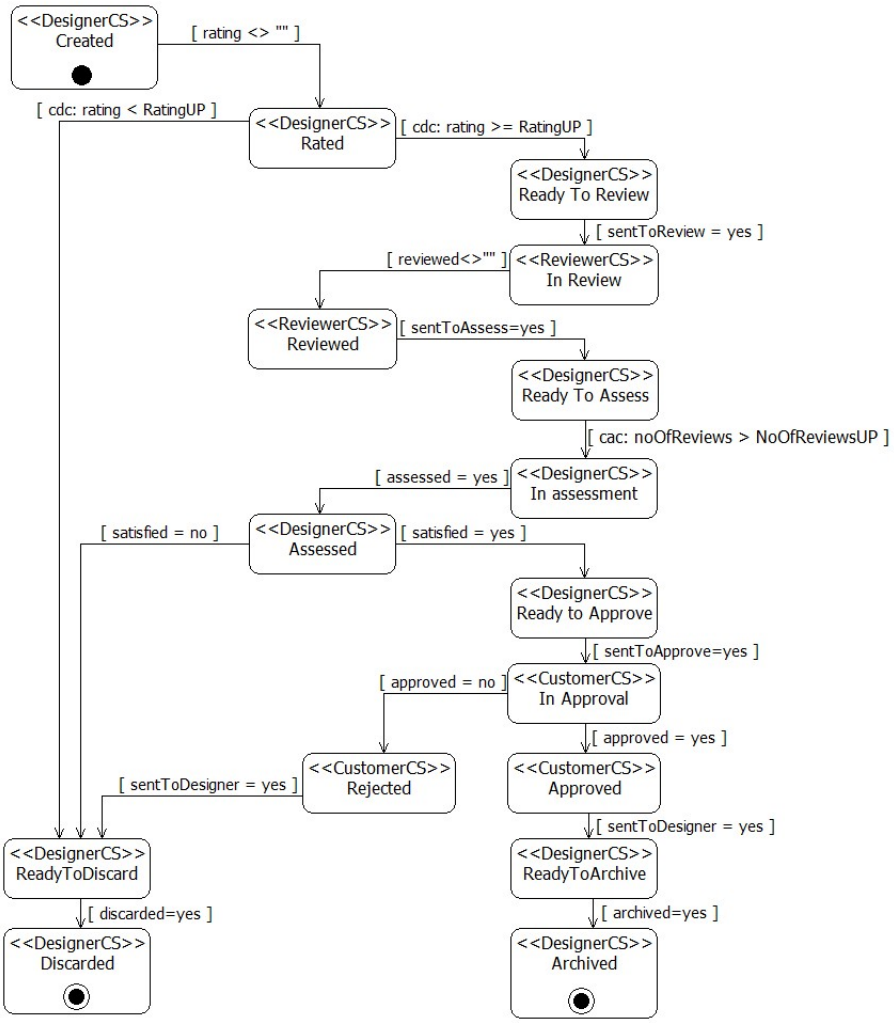

Figure 8.3: Picture Lifecycle

Process Flow Models

Process control flows for designer, reviewer, and customer have been defined. The models of process control flows are presented in Appendix C. Content activities, and decisions based on the use of the MobWEL extension functions: *getContentState* and *getContextValue*, have been appropriately integrated in the process control flows.

Anatomy of the Workflow Process

All workflow parts have been constructed, thus they could be assembled together to build the whole workflow process. The anatomy of the whole MobWEL workflow process defined for the usage scenario is depicted in Figure 8.4. The *Group Identification* and *Picture Lifecycle* parts are role-independent and are included in all workflow partitions. *Context Definition* and *Process Flow* are role-specific, thus

these elements have been defined for each role.

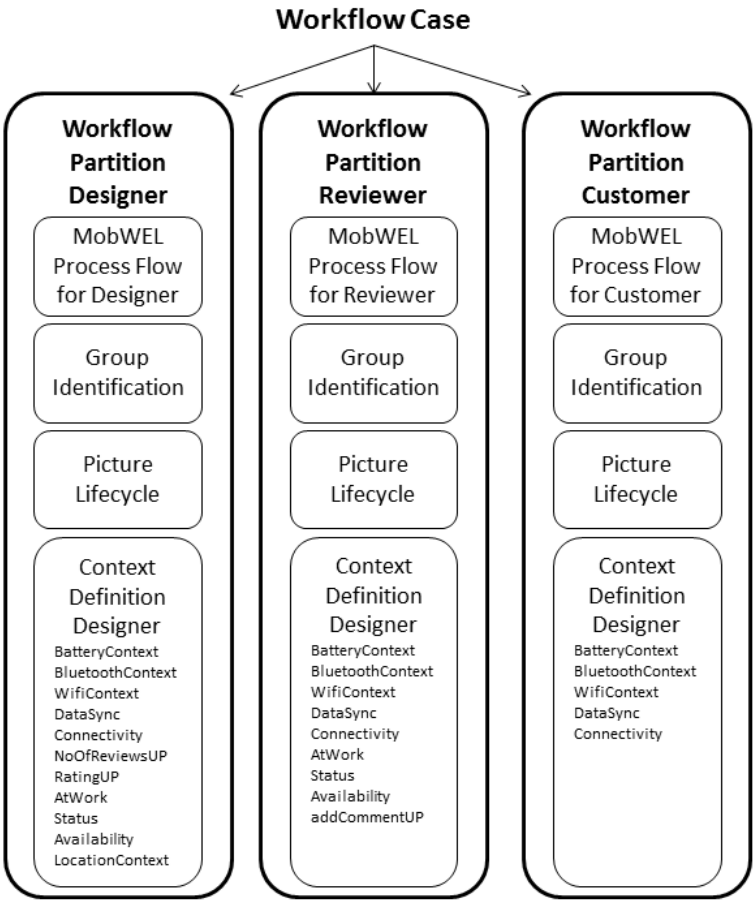

Figure 8.4: Anatomy of the Workflow Process

The workflow process has been mapped into a set of XML documents and deployed to the MobWEL workflow management system which runs on mobile devices. In addition, quantitative data has been collected from the workflow model as presented next.

8.3.2 Quantitative Data

Table 8.1 gives statistics for the MobWEL workflow process model developed in previous section.

As can be seen from the table, the data is divided into two groups. The first group provides details related to the particular workflow process, the second group gives details related to the content lifecycle. The table gives numbers of content activities that have been integrated in workflow process, outlines numbers of possible execution paths. The number of execution paths for the entire workflow depends on the number of participating actors. In this case, the number has been calculated for 5 actors. The table also shows how many context variables have direct impact on

Table 8.1: Workflow Model - Quantitative Data

	Number Of	Designer	Reviewer	Customer	Overall
CONTROL FLOW	Content Activities	10	3	2	15
	Execution Paths	10	2	1	28
	Monitored Contexts	4	3	1	8
	Content Objects	1	1	1	1
	Actors	1	3	1	5
CONTENT LIFECYCLE	States	11	2	3	16
	Transition Firing Sequences	4	3	2	4
	Monitored Contexts	2	0	0	2

the workflow execution. In addition, it shows that there is only one content object processed.

The second part of the table summarises content states which have been defined in the picture lifecycle. The table also outlines the number of possible transition firing sequences and content-related context variables. The next section describes the workflow instantiation process.

8.4 Workflow Instantiation

In this section, the workflow instantiation is described. The first set of experiments has been conducted by running individual workflow instances in a sequence, as presented next.

8.4.1 Sequential Workflow Instantiation

Firstly, all activities in the developed workflow process have been labeled as follows: activities performed by designer are labeled as $A_{D1} - A_{D22}$, activities performed by reviewer are labeled as $A_{R1} - A_{R9}$, and activities performed by designer are labeled

as $A_{C1} - A_{C6}$. Final process flow states are labeled as $F_{D1}, F_{D2}, and F_{D3}$ in the designer's process flow, F_{R1} in the reviewer's process flow, and F_{C1} in the customer's process flow. Let Content Object O be the picture. Content states in the picture lifecycle are labeled with $S_1 - S_{16}$.

To demonstrate the steps involved in the validation method, one execution path has been chosen and is visualised in Figure 8.5 through

- activities that are expected to be activated;
- the process flow sequence that is expected to be executed;
- the state transition sequence that is expected to be fired;
- the context situations required at certain points.

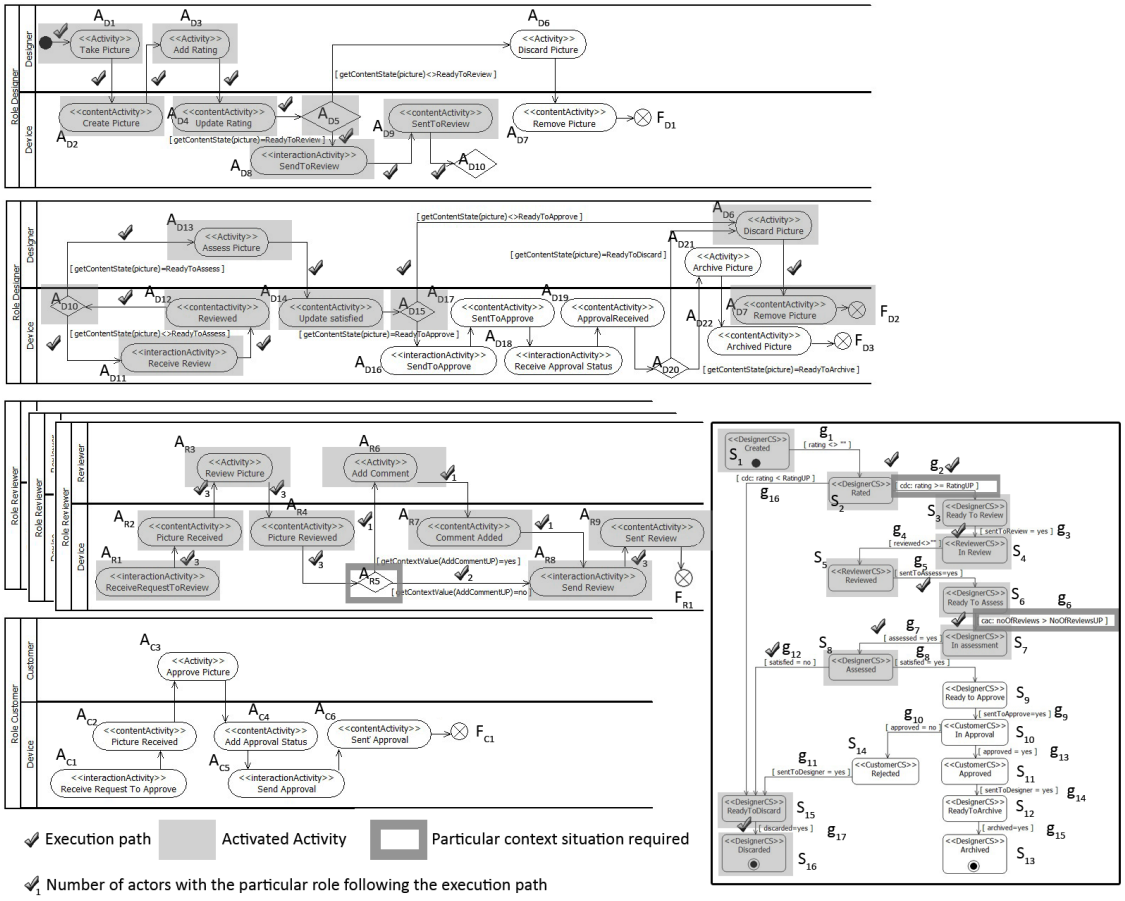

Figure 8.5: Workflow Instance

Next, based on the selected execution path, the expected behaviour of the particular workflow instance is derived by using MobWEL semantics as follows:

- **Designer**

  Number of Actors: 1

  Workflow-active Context:
  $Ctx_D = \{Connectivity, NoOfReviewsUP, RatingUP, Availability, LocationContext\}$

  D1 - Execution Sequence:
  $\omega_0^D \xrightarrow{A_{D1},\widetilde{c}} \omega_1^D \xrightarrow{A_{D2},\widetilde{c}} \omega_2^D \xrightarrow{A_{D3},\widetilde{c}} \omega_3^D \xrightarrow{A_{D4},\widetilde{c}} \omega_4^D \xrightarrow{A_{D5},\widetilde{c}} \omega_5^D \xrightarrow{A_{D8},\widetilde{c}} \omega_6^D \xrightarrow{A_{D9},\widetilde{c}} \omega_7^D \xrightarrow{A_{D10},\widetilde{c}} \omega_8^D \xrightarrow{A_{D11},\widetilde{c}} \omega_9^D \xrightarrow{A_{D12},\widetilde{c}} \omega_{10}^D \xrightarrow{A_{D10},\widetilde{c}} \omega_{11}^D \xrightarrow{A_{D11},\widetilde{c}} \omega_{12}^D \xrightarrow{A_{D12},\widetilde{c}} \omega_{13}^D \xrightarrow{A_{D13},\widetilde{c}} \omega_{14}^D \xrightarrow{A_{D14},\widetilde{c}} \omega_{15}^D \xrightarrow{A_{D15},\widetilde{c}} \omega_{16}^D \xrightarrow{A_{D6},\widetilde{c}} \omega_{17}^D \xrightarrow{A_{D7},\widetilde{c}} \omega_{18}^D$

  D1 - Reachable Final State: F_{D2}

  D1 - Fired State Transition for O:
  $s_1 \xrightarrow{g_1} s_2 \xrightarrow{g_2} s_3 \xrightarrow{g_3} s^{POOL} \xrightarrow{g_5} s_6 \xrightarrow{g_6} s_7 \xrightarrow{g_7} s_8 \xrightarrow{g_{12}} s_{15} \xrightarrow{g_{17}} s_{16}$

  D1 - Reachable Content State for O: s_{16}

  D1 - Reachable Execution States with corresponding content states for O: (ω_9^D, S_{16}, O)

  D1 - Required context situations:
  $c/RatingUP =< (RatingUP, 2) >$;
  $c/NoOfReviewsUP =< (NoOfReviewsUP, 2) >$

- **Reviewer**

  Number of Actors: 3 (R1, R2, R3)

  Workflow-active Context: $Ctx_R = \{Connectivity, AddCommentUP, Availability\}$

  R1 - Execution Sequence:
  $\omega_0^R \xrightarrow{A_{R1},\widetilde{c}} \omega_1^R \xrightarrow{A_{R2},\widetilde{c}} \omega_2^R \xrightarrow{A_{R3},\widetilde{c}} \omega_3^R \xrightarrow{A_{R4},\widetilde{c}} \omega_4^R \xrightarrow{A_{R5},c/AddCommentUP} \omega_5^R \xrightarrow{A_{R8},\widetilde{c}} \omega_6^R \xrightarrow{A_{R9},\widetilde{c}} \omega_7^R$

  R1 - Reachable Final State: F_{R1}

  R1 - Fired State Transition for O:
  $s^{POOL} \xrightarrow{g_3} s_4 \xrightarrow{g_4} s_5 \xrightarrow{g_5} s^{POOL}$

  R1 - Reachable Content State for O: S^{POOL}

  R1 - Reachable Execution States with corresponding content states for O: $(\omega_7^R, S^{POOL}, O)$

R1 - Required context situations:
$c/AddCommentUP =< (AddCommentUP, NO) >$

R2 - Execution Sequence:
$\omega_0^R \xrightarrow{A_{R1},\widetilde{c}} \omega_1^R \xrightarrow{A_{R2},\widetilde{c}} \omega_2^R \xrightarrow{A_{R3},\widetilde{c}} \omega_3^R \xrightarrow{A_{R4},\widetilde{c}} \omega_4^R \xrightarrow{A_{R5},c/AddCommentUP} \omega_5^R \xrightarrow{A_{R6},\widetilde{c}}$
$\omega_6^R \xrightarrow{A_{R7},\widetilde{c}} \omega_7^R \xrightarrow{A_{R8},\widetilde{c}} \omega_8^R \xrightarrow{A_{R9},\widetilde{c}} \omega_9^R$

R2 - Reachable Final State: F_{R1}

R2 - Fired State Transition for O:
$s^{POOL} \xrightarrow{g_3} s_4 \xrightarrow{g_4} s_5 \xrightarrow{g_5} s^{POOL}$

R2 - Reachable Content State for O: S^{POOL}

R2 - Reachable Execution States with corresponding content states for O: $(\omega_7^R, S^{POOL}, O)$

R2 - Required context situations:
$c/AddCommentUP =< (AddCommentUP, YES) >$

R3 - Execution Sequence:
$\omega_0^R \xrightarrow{A_{R1},\widetilde{c}} \omega_1^R \xrightarrow{A_{R2},\widetilde{c}} \omega_2^R \xrightarrow{A_{R3},\widetilde{c}} \omega_3^R \xrightarrow{A_{R4},\widetilde{c}} \omega_4^R \xrightarrow{A_{R5},c/AddCommentUP} \omega_5^R \xrightarrow{A_{R8},\widetilde{c}}$
$\omega_6^R \xrightarrow{A_{R9},\widetilde{c}} \omega_7^R$

R3 - Reachable Final State: F_{R1}

R3 - Fired State Transition for O:
$s^{POOL} \xrightarrow{g_3} s_4 \xrightarrow{g_4} s_5 \xrightarrow{g_5} s^{POOL}$

R3 - Reachable Content State for O: S^{POOL}

R3 - Reachable Execution States with corresponding content states for O: $(\omega_7^R, S^{POOL}, O)$

R3 - Required context situations:
$c/AddCommentUP =< (AddCommentUP, NO) >$

- **Customer**

  Number of Actors: 0

  Workflow-active Context: $Ctx_C = \{Connectivity\}$

After the behaviour is predicted and expressed by using the MobWEL semantics, the workflow process is instantiated and required context situations simulated. Steps involved in the testing procedure are demonstrated in Figure 8.6. Assuming that the corresponding workflow partition has been added by pressing the 'Add Workflow' button in CAWEFA, the 'Run Workflow' button can be clicked to instantiate it (Step

Figure 8.6: Testing

1). The inputs for activities are set in advance for easier execution and monitoring of the workflow instance (Step 2). To enable easier simulation of context data, preferences are created for all atomic context components. Thus the context data can be simulated or changed as required (Step 3 and 4). Context Provider processes atomic context data and derives the aggregated context values. Both atomic and aggregated context values are displayed in the LogCat, the Android logging system. The log outputs are checked to determine whether the aggregated contexts are derived correctly and the required context situation is achieved (Step 5). After that, the workflow is instantiated by pressing the 'Run Workflow' button (Step 6). LogCat is used to derive the important information about the running workflow instances (Step 7). To monitor content states of the picture processed in the workflow instance, the 'StateMachine' filter is applied (Step 7a). To observe and collect data about the executed activities, the 'ActivityInstance' filter is applied (Step 7b). Then the actual results are collected and compared to the predicted results.

Results

Table 8.2 gives results gathered by performing the sequential instantiation testing.

Table 8.2: Sequential Instantiation - Results Summary

Running workflow instances	20
Completed workflow instances	20
Correct execution sequences (Overall)	18
Reached the expected final states (Designer)	18
Reached the expected final states (Reviewer1)	12
Reached the expected final states (Reviewer2)	11
Reached the expected final states (Reviewer3)	7
Reached the expected final states (Customer)	5
Correct fired state transitions (Overall)	18
Correct fired state transitions (Designer)	18
Correct fired state transitions (Reviewer1)	12
Correct fired state transitions (Reviewer2)	11
Correct fired state transitions (Reviewer3)	7
Correct fired state transitions (Customer)	5
Number of recorded errors	2

Next section describes the parallel instantiation method and presents the obtained results.

8.4.2 Parallel Workflow Instantiation

Parallel workflow instantiation has been conducted in order to monitor the behaviour of multiple workflow instances running at the same time. Context data is changed randomly, therefore, the behaviour of each workflow instance from its start to the end would be difficult. Therefore, it is monitored whether instances reached final process

flow states and whether the reached content state corresponds to the expected content state associated with the particular final process flow state.

Milestones

Table 8.3 gives an overview of the key milestones which have been set to determine the possible final states of workflow instances. To each final process flow state, a corresponding final content state is assigned.

Table 8.3: Milestones

Reachable Final Flow State	Reachable Content State
F_{D1}	S_{16}
F_{D2}	S_{16}
F_{D3}	S_{13}
F_{R1}	S^{POOL}
F_{C1}	S^{POOL}

Results

Table 8.4 gives results gathered by performing the parallel workflow instantiation.

Table 8.4: Parallel Instantiation - Results Summary

Max of parallel running workflow instances	5
Completed workflow instances	95%
Correct execution sequences	92%
Correct fired state transitions	88%
Number of recorded errors	5

The results have been analysed and the findings are discussed next.

8.5 Discussing Findings

In this section, the results of the evaluation are discussed. The evaluation has been based on the usage scenario described in Chapter 2. The scenario represents a certain class of workflows. However, using only one scenario in the validation process means that the results cannot be generalised, and their implication is limited only to the class of scenarios.

The results of this validation indicate that the MobWEL workflow language can be used to build mobile context-aware content-centric workflows. The experiment was successful. It has been shown that the particular MobWEL workflow for the usage scenario could be constructed and its instances could be managed and executed.

By constructing the workflow description for the teamwork described in the usage scenario, it was demonstrated that the proposed MobWEL metamodel provides all constructs needed to build the context-aware content-centric workflow.

Using two different workflow instantiation approaches allowed monitoring behaviour of workflow instances and its partitions. Based on the obtained results and equivalence checking between the expected and actual behaviour of workflow instances, it is concluded that MobWEL workflows are manageable and executable. Although there were errors recorded, the errors could arise from mistakes made in the source code.

Context situations have been monitored in the LogCat system. Successful delivery and consumption of context information has been particularly shown in the sequential workflow instantiation where the workflow instances behaved as expected. It means that context and content state changes have been realised. Propagation of context and content state changes to all running workflow instances adequately have been demonstrated in the parallel workflow instantiation.

However, the findings might not be transferable to all MobWEL workflows. It is difficult to accept the hypotheses generally as the designed artifact is a workflow language that can be used to build workflows, and this activity is slightly subjective depending on the intent of the particular workflow designer. The MobWEL workflow language provides constructs and elements which can be combined in several ways, and it is hard to validate the correctness of the constructed workflow models without its extensive testing.

In addition, more research on this topic needs to be undertaken in order to clearly establish the consistency between the process-based workflow model and the associated content lifecycle(s).

With this section, this chapter is concluded and its summary is given next.

8.6 Summary

This chapter has described the evaluation process of the proposed MobWEL workflow approach. It has been shown that mobile context-aware content-centric workflows can be constructed, managed and executed. However, more extensive testing is needed in order to generalise the findings.

With this chapter, the validation part of the thesis is concluded. The next part provides the overall summary of the thesis and outlines the possibilities for future work.

Part IV

Conclusion

Chapter 9

Conclusion and Future Work

Contents

In this thesis, a context-aware content-centric workflow approach for mobile peer-to-peer collaboration has been presented. This final chapter concludes this thesis by providing the overall summary in Section 9.1, summarising the main contributions and the impact of this work in Section 9.2, and discussing possible future work and research in Section 9.3.

9.1 Summary of This Thesis

With mobile devices becoming a part of human daily life, people's expectations and demands on the services the mobile devices offer have been increased. By using mobile devices, workers can collaborate and remain productive regardless their location, thereby, they expect to have tools which would enable easier, faster and more convenient collaboration process. Activity-oriented workflow management has become a crucial technology that has been implemented by many organisations to automate their business processes and enhance collaboration among workers. This technology has been also adapted for mobile collaboration, however, there are still research challenges as shown in first part of this thesis. The successful execution of mobile workflows depends on various factors such as the execution environment, collaborators' preferences or individual situations. Therefore, there is a need to monitor the context situations the mobile devices reside in and adapt the workflow technology to react to the constantly changing context.

Collaborators share mobile content such as pictures or documents. Although these content objects can be processed in activity-oriented workflows, their behaviour is hardly visible. Visualising the states of the content objects during their lifecycle brings a number of benefits. Firstly, an additional dimension to workflow is added which means that two complementary views on such workflow are available. Secondly, information about content states can be used to communicate progress among collaborators who do not need to know the workflow execution logic of their peers. Finally, content objects can be associated or dependent on each other. Knowing the content state of a certain content object can influence the behaviour of another object. Therefore, the behaviour of content objects needs to be expressed similarly as the control flow logic is, and these two workflow aspects have to be integrated.

The main objective of this thesis was to contribute to the long-term vision of an intelligent mobile collaborative environment by adapting the collaborative workflow technology for mobile peer-to-peer collaboration. This work was undertaken to design a workflow model that defines context-aware content-centric processes at an abstract level and in a machine-interpretable form, and a generic workflow management software package that is capable of managing and executing of such workflow processes and operates in a peer-to-peer manner on mobile devices.

Our research focused on bringing a solution to these challenges. Although there have been existing workflow approaches presented in Chapter 2 which partially fulfill the objectives, none of the approaches has been designed to *a*) be context-aware; and *b*) integrate content behaviour; and *c*) be designed for mobile devices. Therefore, following the scenario-based design, the MobWEL workflow approach has been designed and introduced in this thesis. The MobWEL workflow approach addresses all three requirements. The MobWEL language allows to define mobile context-aware content-centric workflows. MobWEL workflows are carried out by the MobWEL workflow management system which logical and run-time architecture have been described.

The contributions of this thesis are described next.

9.2 Contributions of This Thesis

This thesis contributes to the existing knowledge domain of collaborative workflow in the following ways:

- **Mobile Workflow Execution Language (MobWEL) Metamodel:** Four workflow parts of the MobWEL workflow language (Workflow-specific context defi-

nition, Context-aware content lifecycle, Group identification, and Process Control Flow) allow to define workflows which are context-aware, have content behaviour integrated and support peer-to-peer interaction. The solution benefits from the well-defined syntax and semantics provided by $BPEL$, a light interaction model specified in $BPEL^{light}$, the workflow contextualisation approach used in $Context4BPEL$, and by adopting the dimensions defined in BALSA, the artifact-centric workflow model.

- **MobWEL Syntax and Semantics:** Syntax and semantics of MobWEL process control flow and content lifecycles have been formalised. A context situation has been formalised to capture the state of the execution environment. Formal semantics of MobWEL process control flow has been defined and can be used to express the possible workflow execution sequences. The process control flow semantics has been extended with data flows. Semantics has been defined for processing of content lifecycles and possible paths in content behaviour. Finally, definitions for consistency between the process control flow and content lifecycle has been provided to determine whether both workflow parts evolve consistently and dependably. The provided MobWEL semantics can be used to describe the behaviour of MobWEL workflows in a formal way.

- **MobWEL Workflow Management System Specifications:** The logical and run-time architecture of the MobWEL workflow management system that extends BPEL engine has been described. The system consists of a number of components (Context Provider, Context Manager, Content Manager, MobWEL Engine, and Peer-to-peer Interaction Manager). The system carries out workflows described by using the MobWEL workflow language. Context Provider monitors, acquired, processes and disseminates context information. Context Manager is able to filter workflow-active and relevant context information, and route it to right internal component. Content Manager provides advanced content management functionalities and a state transition system that manages content behaviour. The MobWEL engine manages and executes workflow instances. Peer-to-peer Interaction Manager is responsible for handling interaction and message exchange among collaborators.

- **Workflow-Specific Context Management Approach:** The context definition approach has been designed in a workflow-specific way. The same context definition model is parsed into Context Provider and Context Manager, however, each component extracts different bits of information. While Context Provider extracts context hierarchies, Context Manager focuses on workflow-specific parts related to the consumption of context information within the Mob-

WEL workflow management system. This approach supports separation of the context acquisition logic from context adaptation logic which enables to externalise Context Provider. As an external component, Context Provider can be used by multiple context aware applications running on the same mobile device.

- **Context-Aware Content Management Approach:** In activity-oriented workflows, the business processes are described by stating the actions that need to be taken to achieve a common goal and mobile content objects are seen only as side products produced by the workflows. In MobWEL workflows, a mobile content object is considered as a significant workflow artifact, and its lifecycle is visualised and managed independently from the main control flow logic. This enables to constantly monitor the state of content objects. The information is consumed by the process control flow, and is used either within the same workflow instance, or to communicate progress to fellow collaborators. In addition, context awareness has been integrated into content lifecycle so the transitions between two content states can be based also on external events related to the execution environment and collaborator's personal needs.
- **Software Prototypes:** Two prototypes have been built for the Android platform and will be released as open source software. The prototype of an externalised Context Provider is ContextEngine. The prototype of the MobWEL workflow management system has been named CAWEFA.

To sum up, design research adopted in this work has produced a viable artefacts for workflow designers and software developers. The MobWEL workflow language is a tool that can be used by workflow designers to model context-aware content-centric mobile workflows. A detailed description of a logical architecture of the mobile workflow management system can be used by software developers who would be able to develop and implement such mobile workflow management system on any mobile platform. In this thesis, first milestone to the definition, management and execution of the MobWEL workflows has been presented, and several possible extensions and directions for future work have been envisioned as presented next.

9.3 Future Work

Integration of context and content awareness into the workflow technology offers numerous benefits, however, there have been also some limitations discovered that should be addressed in future work. This section lists several possible extensions

and directions for future work. As described next, some extensions would improve the designed MobWEL workflow model, whereas others would expand its application use for other classes of usage scenarios.

Although integration of context awareness enables workflow adaptation to individual user needs and situation in the current run-time environment, the proposed approach requires a prediction of possible context situations beforehand. The consensus of ambient intelligence includes a vision that context emerges in the moment and cannot be fully predicted. Thus it would be beneficial to add functionality that determines the future context states based on prognosis, past context states or other machine-learning technique. In addition, produced context information can be incorrect, or inaccurate, thus a context validation technique should be employed to deal with information imperfection. On the other hand, there can be situations that context information cannot be provided at certain times, for instance when a sensor is switched off. In this case, if context definition model is enhanced by adding priorities to context values, the context value with the highest priority is supplied instead.

In addition, content sharing between devices might be a time consuming and costly operation, especially when one collaborative task may be accomplished by a number of actors with the same role but only few of them might be able to perform the task. Sending the piece of content to all of them would be inefficient in terms of transfer cost, device resource usage and user time consumption. A workflow management system running on each device would need to cope with the incoming content, store it and trigger an according action. Every participant would be informed about the task despite the fact that he might not be able to accomplish it within the required time. So another valuable extension of this work would be in the development of an appropriate content sharing strategy.

Further, the proposed approach to model MobWEL workflows from start to finish before its deployment to the mobile workflow management system limits its flexibility at run-time. Some approaches towards more flexible workflows, presented in Chapter 3, suggest the change and evolution of workflow schema and workflow definition at run-time. This can include addition or removing of activities, or building workflow cases on-the-fly. It indicates that it would be valuable to investigate how MobWEL workflows schema can be modified and adapted at run-time.

There is also lots of potential to further develop and enhance the components of the MobWEL workflow management system. For example, Communication Manager could be optimised by grouping messages that are supposed to be sent to one collaborator. Identity Manager could be improved by enabling modification of the team of collaborators at run-time.

Also, task management is another key area that should be investigated in the

future. With context awareness integrated, there are many possibilities to design a component for task management that would manage tasks in a suitable way that is also adapted to individual user needs and situation.

Finally, the development of the MobWEL language has been tailor-made for a specific class of workflows. It would be interesting to use and validate the MobWEL workflow approach in other scenarios. Furthermore, creating MobWEL workflows is a slightly subjective activity based on the workflow designer's perception and domain expertise. Although a MobWEL design methodology is proposed in Appendix A, this methodology is too general and should be amended in order to simplify the design of MobWEL workflows.

Appendix A - Design Methodology for MobWEL workflows

The following design methodology determines the steps that need to be taken to fully define MobWEL workflows:

STEP.1: **Group Identification**

1.a Identify roles.

1.b Identify actors and their assigned roles.

STEP.2: **Define Workflow Partitions**

2.a Define activities that are performed by each role.

2.b Define the flow of the activities and identify the decision points where the decisions are based on context information.

STEP.3: **Design Context Model**

3.a For each role, identify atomic contexts and possible context values which may have an indirect influence the execution of the particular workflow partition.

3.b Aggregate contexts and define rules for context aggregations.

3.c Define active contexts (direct influence on workflow) and type of consumption.

STEP.4: **Content Discovery**

4.a Identify key content objects that are processed in the given workflow and which behaviour should be monitored.

4.b Define content-related metadata and content information model.

STEP.5: **Design of Content Lifecycles**

5.a Discover key stages of content behaviour.

5.b Logical design of lifecycles (including transitions).

5.c Identify Context-Driven Conditions and Context-Aware Conditions for transitions.

5.d Add content-related context to the context model.

STEP.6: **Describe the whole workflow process in MobWEL**

6.a Describe roles and collaborators in an XML document that conforms to the *groupIdentification.xsd* XML schema listed in Appendix 2.

6.b Describe lifecycles of the identified content objects (business artifacts) in an XML document that conforms to the *contentLifecycle.xsd* XML schema listed in Appendix 2.

6.c Describe a context model for each role in XML documents which conform to the *contextDefinition.xsd* XML schema listed in Appendix 2.

6.d Describe a workflow partition for each role by using the MobWEL process flow language definition described in Chapter 4.

STEP.7: **Workflow Realisation**

7.a Deploy to the MobWEL workflow management system.

Appendix B - XML schemas

Context Definition XML Schema

Listing 1: Context Definition XML Schema

```
<?xml version="1.0" encoding="UTF 8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <!  targetNamespace="http://uk.ac.uwl.mdse/MobWEL/ContextDefinition"  >
  <!  definition of complex elements  >
 <xs:element name="ContextDefinition">
   <xs:complexType>
     <xs:sequence>
       <xs:element ref="Context" maxOccurs="unbounded"/>
       <xs:element ref="CompositeContext" minOccurs="0" maxOccurs="unbounded"/>
     </xs:sequence>
     <xs:attribute name="AppKey" type="xs:string"/>
   </xs:complexType>
 </xs:element>
 <xs:element name="CompositeContext">
   <xs:complexType>
     <xs:sequence>
       <xs:element ref="ContextValue" maxOccurs="unbounded"/>
       <xs:element ref="ChildContext" minOccurs="2" maxOccurs="unbounded"/>
       <xs:element ref="Rule" maxOccurs="unbounded"/>
     </xs:sequence>
     <xs:attribute ref="ContextName" use="required"/>
     <xs:attribute ref="ContextAssociation" use="required"/>
     <xs:attribute ref="WorkflowActive" default="no"/>
   </xs:complexType>
 </xs:element>
 <xs:element name="Context">
   <xs:complexType>
     <xs:choice>
       <xs:element ref="ContextValue" maxOccurs="unbounded"/>
       <xs:element ref="Range" maxOccurs="unbounded"/>
       <xs:element ref="SpecificContextValue" maxOccurs="unbounded"/>
```

```
    </xs:choice>
    <xs:attribute ref="ContextName" use="required"/>
    <xs:attribute ref="ContextType" default="Default"/>
    <xs:attribute ref="ContextAssociation" use="required"/>
    <xs:attribute ref="WorkflowActive" default="no"/>
  </xs:complexType>
</xs:element>
<xs:element name="Range">
  <xs:complexType>
    <xs:all>
      <xs:element ref="RangeName"/>
      <xs:element ref="Min"/>
      <xs:element ref="Max"/>
    </xs:all>
  </xs:complexType>
</xs:element>
<xs:element name="SpecificContextValue">
  <xs:complexType>
    <xs:all>
      <xs:element ref="ContextValue"/>
      <xs:element ref="NumericValue1"/>
      <xs:element ref="NumericValue2"/>
    </xs:all>
  </xs:complexType>
</xs:element>
<xs:element name="Rule">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="ChildValue" maxOccurs="unbounded"/>
      <xs:element ref="ParentValue" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<!  definition of simple elements  >
<xs:element name="ContextValue" type="xs:string"/>
<xs:element name="Min" type="xs:decimal"/>
<xs:element name="Max" type="xs:decimal"/>
<xs:element name="RangeName" type="xs:string"/>
<xs:element name="NumericValue1" type="xs:decimal"/>
<xs:element name="NumericValue2" type="xs:decimal"/>
<xs:element name="ChildContext" type="xs:string"/>
<xs:element name="ChildValue" type="xs:string"/>
<xs:element name="ParentValue" type="xs:string"/>
<!  definition of attributes  >
```

```
  <xs:attribute name="ContextName" type="xs:string"/>
  <xs:attribute name="AppKey" type="xs:string"/>
  <xs:attribute name="ContextType">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="Location"/>
        <xs:enumeration value="UserPreference"/>
        <xs:enumeration value="Default"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="ContextAssociation">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="Communication"/>
        <xs:enumeration value="UserProfile"/>
        <xs:enumeration value="Workflow"/>
        <xs:enumeration value="Content"/>
        <xs:enumeration value="Collaboration"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="WorkflowActive">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="yes"/>
        <xs:enumeration value="no"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:schema>
```

Content Lifecycle XML Schema

Listing 2: Content Lifecycle XML Schema

```
<?xml version="1.0" encoding="UTF 8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <! targetNamespace="http://uk.ac.uwl.mdse/MobWEL/ContentLifecycle" >
  <xs:element name="Lifecycle">
  <xs:annotation>
    <xs:documentation>This is the root element for a content lifecycle</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:all>
      <xs:element ref="Content" minOccurs="1" maxOccurs="1"/>
      <xs:element ref="States" minOccurs="1" maxOccurs="1"/>
      <xs:element ref="Transitions" minOccurs="1" maxOccurs="1"/>
    </xs:all>
    <xs:attribute name="lifecycleName" type="xs:string"/>
    <xs:attribute name="targetNamespace" type="xs:anyURI" use="required"/>
    <xs:attribute name="queryLangauge" type="xs:anyURI" default="http://uk.ac.uwl.mdse/
        Cawefa/CEQL"/>
    <xs:attribute name="expressionLangauge" type="xs:anyURI" default="http://uk.ac.uwl.mdse
        /Cawefa/CEQL"/>
  </xs:complexType>
  </xs:element>
  <xs:element name="Content">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="ContentType"/>
        <xs:element ref="Metadata" minOccurs="1" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="ContentType">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="IMAGE"/>
        <xs:enumeration value="AUDIO"/>
        <xs:enumeration value="VIDEO"/>
        <xs:enumeration value="SMS"/>
        <xs:enumeration value="CONTACT"/>
        <xs:enumeration value="CAWEFA"/>
        <xs:enumeration value="NONE"/>
      </xs:restriction>
    </xs:simpleType>
```

```
</xs:element>
<xs:element name="Metadata">
  <xs:complexType>
    <xs:all>
      <xs:element name="MetadataName"/>
      <xs:element ref="DataType"/>
      <xs:element ref="MetadataType"/>
      <xs:element ref="MultiplicityAllowed"/>
    </xs:all>
  </xs:complexType>
</xs:element>
<xs:element name="MultiplicityAllowed">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="yes"/>
      <xs:enumeration value="no"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
<xs:element name="DataType">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="text"/>
      <xs:enumeration value="number"/>
      <xs:enumeration value="date"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
<xs:element name="MetadataType">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="location"/>
      <xs:enumeration value="management"/>
      <xs:enumeration value="control"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
<xs:element name="States">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="State" minOccurs="1" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```
  <xs:element name="State">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="StateName" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Transitions">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Transition" minOccurs="1" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Transition">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="SourceState" minOccurs="1" maxOccurs="1"/>
        <xs:element name="DestinationState" minOccurs="1" maxOccurs="1"/>
        <xs:element ref="Guard" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Guard">
    <xs:complexType>
      <xs:complexContent mixed="true">
        <xs:extension base="tExpression"/>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="tExpression" mixed="true">
    <xs:sequence>
      <xs:any minOccurs="1" maxOccurs="unbounded" processContents="lax"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Group Identification XML Schema

Listing 3: Group Identification XML Schema

```
<?xml version="1.0" encoding="UTF 8"?>
<xsd:schema elementFormDefault="qualified" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <! targetNamespace="http://uk.ac.uwl.mdse/MobWEL/GroupIdentification" >
  <xsd:element name="collaboratorsGroup">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="collaborators"/>
        <xsd:element ref="roles"/>
      </xsd:sequence>
    </xsd:complexType>  </xsd:element>
  <xsd:element name="roles">
    <xsd:complexType>
      <xsd:sequence>
          <xsd:element ref="role" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="role">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="actor" maxOccurs="unbounded"/>
      </xsd:sequence>
    <xsd:attribute name="roleName" type="xsd:string" use="required"/>
  </xsd:complexType>  </xsd:element>
  <xsd:element name="actor" type="xsd:string"/>
  <xsd:element name="collaborators">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="collaborator" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="collaborator">
    <xsd:complexType>
      <xsd:attribute name="name" type="xsd:string" use="required"/>
      <xsd:attribute name="number" type="xsd:string" use="required"/>
      <xsd:attribute name="wifi" type="xsd:string" use="required"/>
      <xsd:attribute name="bluetooth" type="xsd:string" use="required"/>
    </xsd:complexType>  </xsd:element>
 </xsd:schema>
```

Appendix C - MobWEL Workflow Models

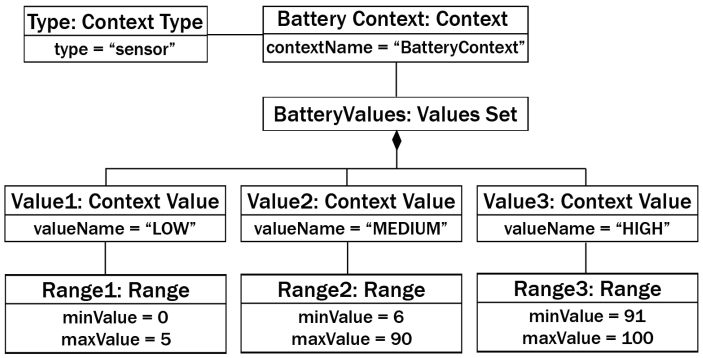

Figure 1: Model for Battery Context

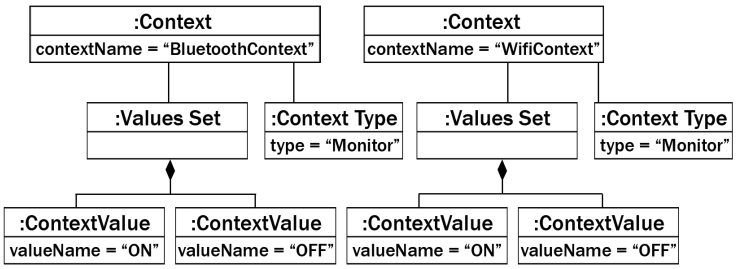

Figure 2: Models for Bluetooth Context and Wifi Context

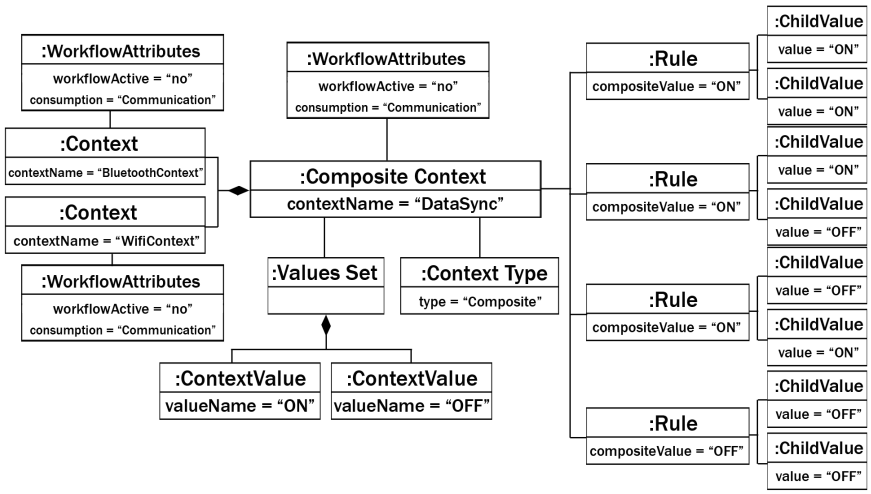

Figure 3: Model for DataSync

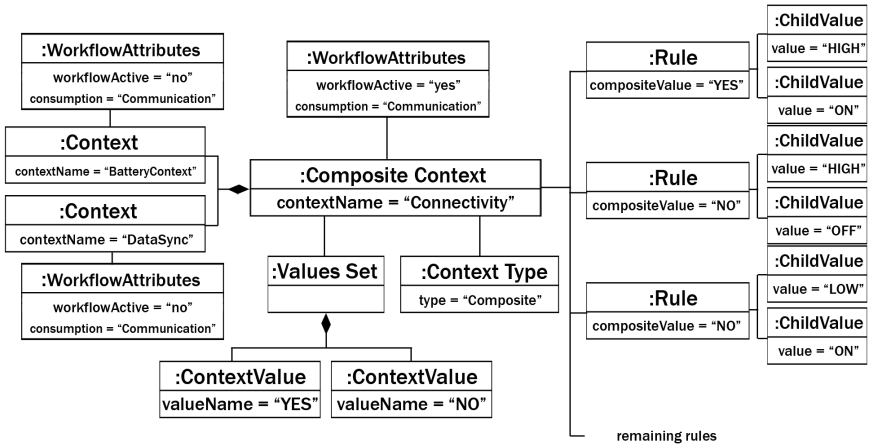

Figure 4: Model for Connectivity

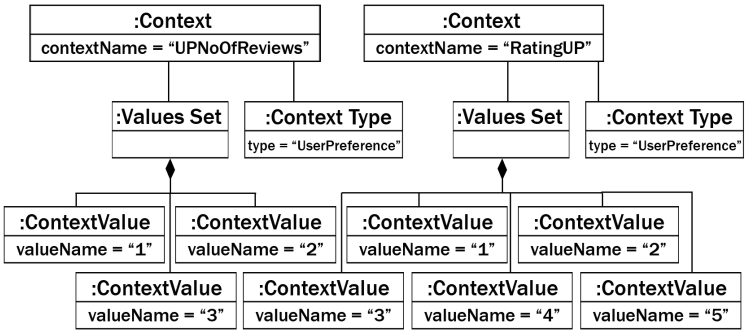

Figure 5: Models for User Preferences

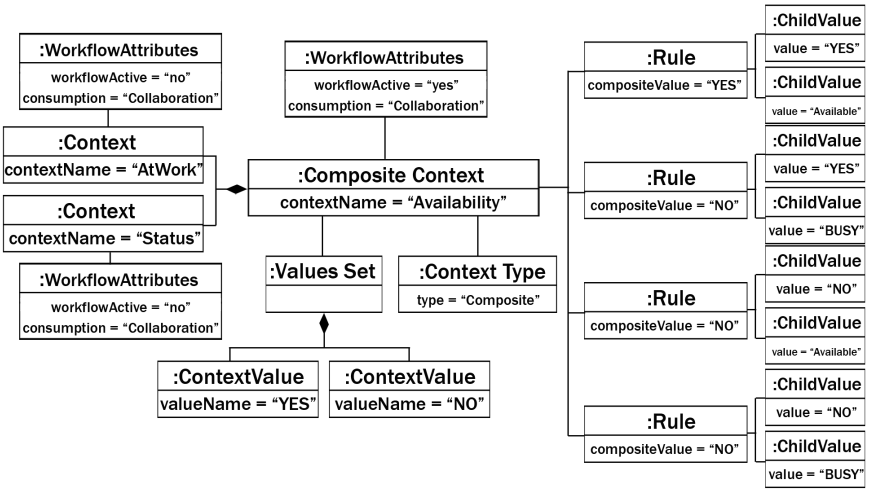

Figure 6: Model for Collaborator’s Availability

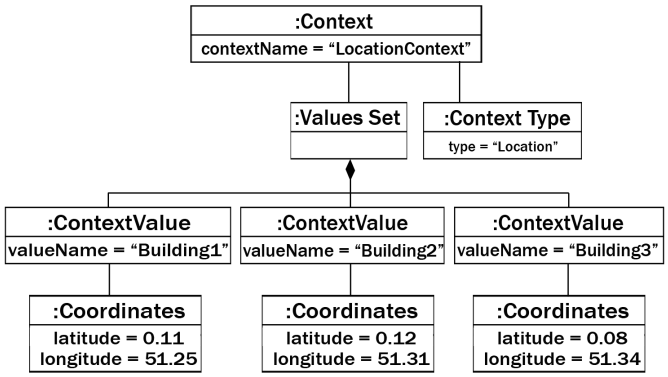

Figure 7: Model for Location

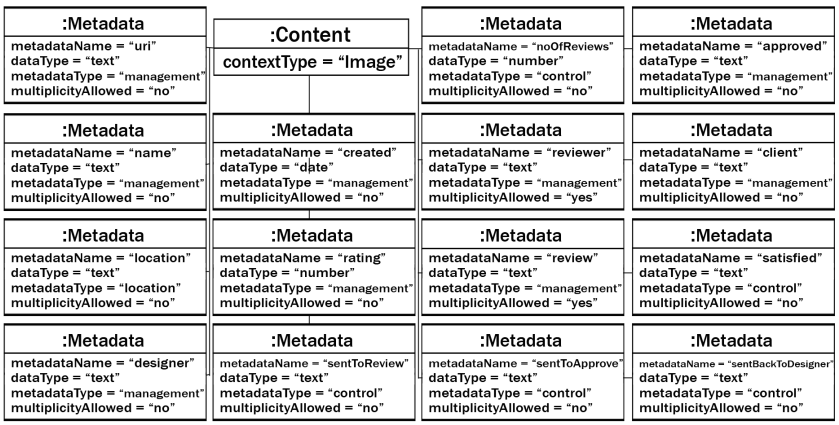

Figure 8: Picture Information Object Model

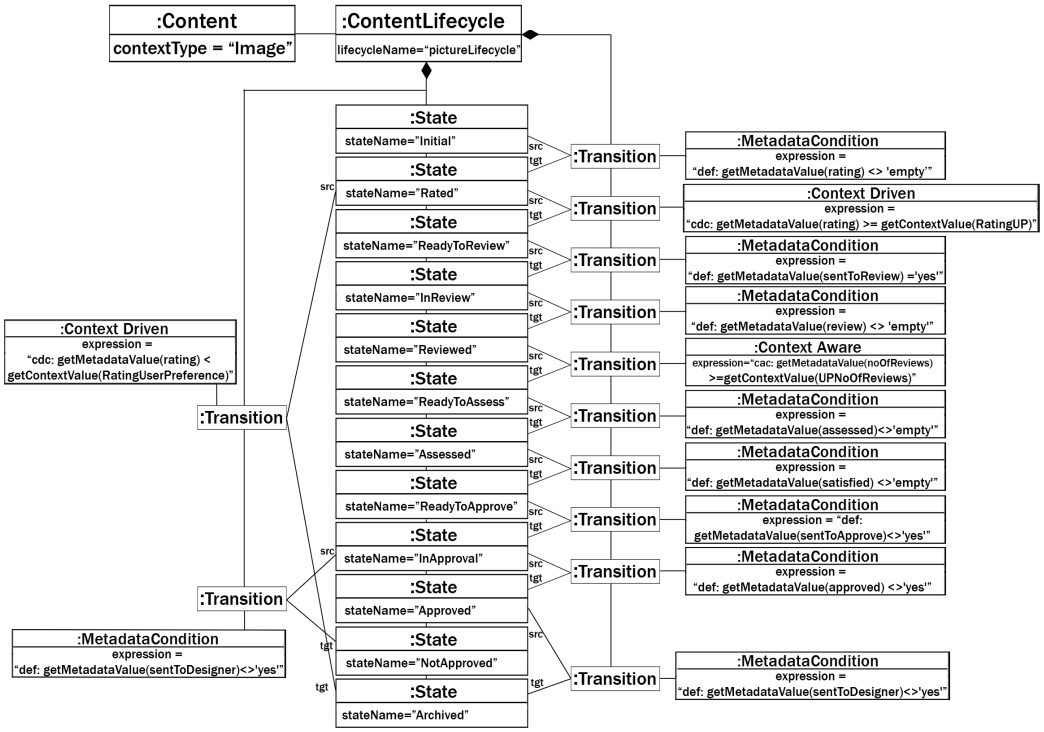

Figure 9: Picture Lifecycle Object Model

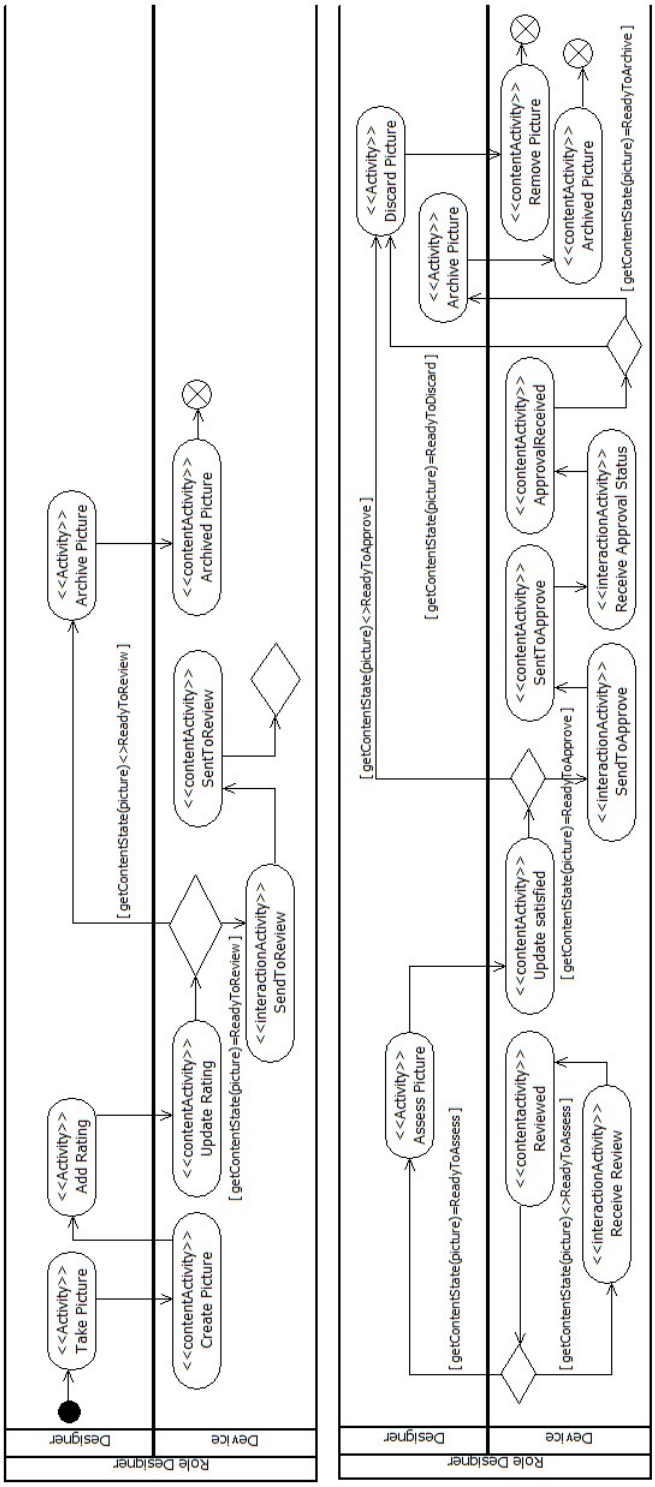

Figure 10: Process Control Flow for Designer

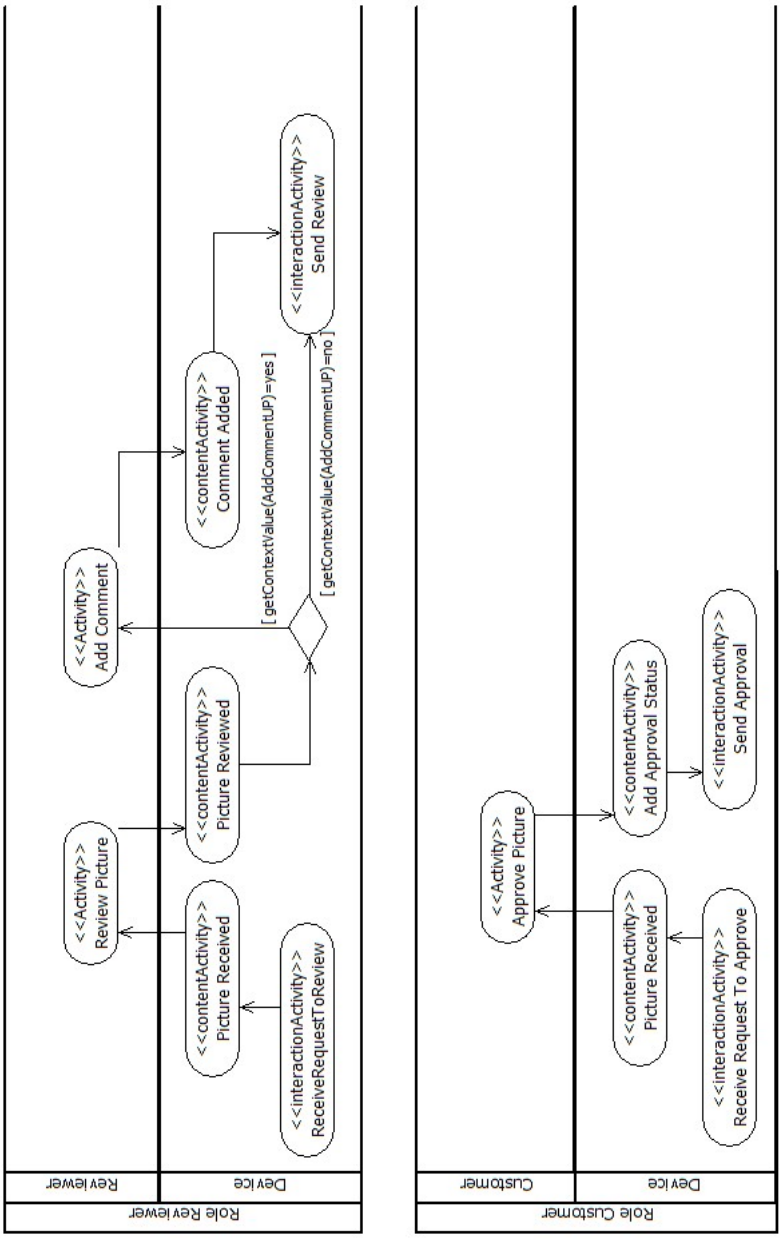

Figure 11: Process Control Flow for Reviewer and Customer

Bibliography

Aalst, W., Barthelmess, P., Ellis, C. A., & Wainer, J. (2001). Proclets: A framework for lightweight interacting workflow processes. *International Journal of Cooperative Information Systems*, *10*(4), pp. 443–481.

Aalst, W., Basten, T., Verbeek, H. M. W., Verkoulen, P. A. C., & Voorhoeve, M. (2000). Adaptive workflow. (pp. 63–70). Norwell, MA, USA: Kluwer Academic Publishers.

Aalst, W., & Hee, K. (2004). *Workflow management: models, methods, and systems*. MIT Press.

Aalst, W., Pesic, M., & Schonenberg, H. (2009). Declarative workflows: Balancing between flexibility and support. *Computer Science-Research and Development*, *23*(2), pp. 99–113.

Aalst, W. M., Ter Hofstede, A. H., Kiepuszewski, B., & Barros, A. P. (2003). Workflow patterns. *Distributed and parallel databases*, *14*(1), 5–51.

Ableson, F., Sen, R., & King, C. (2011). *Android in Action*. Manning Publications Co., 2nd. ed.

Adams, M., Hofstede, A., Russell, N., & Aalst, W. (2009). Dynamic and context-aware process adaptation. *Handbook of research on complex dynamic process management: techniques for adaptability in turbulent environments/Ed. MM Wang, Z. Sun*, (pp. 104–136).

Androutsellis-Theotokis, S., & Spinellis, D. (2004). A survey of peer-to-peer content distribution technologies. *ACM Comput. Surv.*, *36*(4), pp. 335–371.

Bardram, J. E., & Hansen, T. R. (2004). The AWARE architecture: supporting context-mediated social awareness in mobile cooperation. In *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, (pp. 192–201). Chicago, Illinois, USA: ACM.

Battista, D., De Leoni, M., De Gaetanis, A., Mecella, M., Pezzullo, A., Russo, A., & Saponaro, C. (2008). ROME4EU: a web service-based process-aware system for smart devices. *Service-Oriented Computing–ICSOC 2008*, (pp. 726–727).

Bellavista, P., Corradi, A., Fanelli, M., & Foschini, L. (2013). A survey of context data distribution for mobile ubiquitous systems. *ACM Computing Surveys*, *45*(1), pp. 1–49.

Bettini, C., Brdiczka, O., Henricksen, K., Indulska, J., Nicklas, D., Ranganathan, A., & Riboni, D. (2010). A survey of context modelling and reasoning techniques. *Pervasive and Mobile Computing*, *6*(2), pp. 161–180.

Bhattacharya, K., Hull, R., & Su, J. (2009). A data-centric design methodology for business processes. In *Handbook of Research on Business Process Modeling, chapter 23*.

Bierig, R. (2008). *Event and Map Content Personalisation in a Mobile and Context-Aware Environment*. Ph.D. thesis, The Robert Gordon University.

Blessing, L. T. M., & Chakrabarti, A. (2009). *DRM, A Design Research Methodology*. Springer.

Boiko, B. (2005). *Content management bible*. Wiley Pub.

Bolchini, C., Curino, C. A., Orsi, G., Quintarelli, E., Rossato, R., Schreiber, F. A., & Tanca, L. (2009). And what can context do for data? *Communications of the ACM*, *52*(11), pp. 136–140.

Buhler, P. A., & Vidal, J. M. (2003). Adaptive workflow = web services+agents. *In Proceedings of the International Conference on Web Services*, *3*, 131–137.

Clark, T., Sammut, P., & Williams, J. (2008). Applied metamodelling: A foundation for language driven development | lambda the ultimate. [Online] Available at: http://lambda-the-ultimate.org/node/2711 [Accessed on 24 July 2012].

Cook, D. J., Augusto, J. C., & Jakkula, V. R. (2009). Ambient intelligence: Technologies, applications, and opportunities. *Pervasive and Mobile Computing*, *5*(4), pp. 277–298.

Coulouris, G. F., Dollimore, J., & Kindberg, T. (2005). *Distributed systems: concepts and design*. Addison-Wesley Longman.

Daniele, L. M., Silva, E., Pires, L. F., & Sinderen, M. (2009). A SOA-based platform-specific framework for context-aware mobile applications. In *Enterprise Interoperability*, vol. 38 of *Lecture Notes in Business Information Processing*, (pp. 25–37). Springer Berlin Heidelberg.

Dawson, L., Ling, S., Indrawan, M., Weeding, S., & Fernando, J. (2008). Towards a framework for mobile information environments: a hospital-based example. In *Proceedings of the 6th International Conference on Advances in Mobile Computing and Multimedia*, (pp. 490–494). Linz, Austria: ACM.

Decker, G., Kopp, O., Leymann, F., & Weske, M. (2007). BPEL4Chor: extending BPEL for modeling choreographies. In *IEEE International Conference on Web Services, 2007. ICWS 2007*, (pp. 296–303). IEEE.

Denning, P. J. (2001). *The Invisible future: the seamless integration of technology into everyday life*. McGraw-Hill, Inc. New York, NY, USA.

Dey, A., Abowd, G., & Salber, D. (2001). A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction*, *16*(2-4), pp. 97–166.

Ellis, C., Keddara, K., & Rozenberg, G. (1995). Dynamic change within workflow systems. In *Proceedings of conference on Organizational computing systems*, COCS '95, (pp. 10–21). New York, NY, USA: ACM.

Erickson, J., Rhodes, M., Spence, S., Banks, D., Rutherford, J., Simpson, E., Belrose, G., & Perry, R. (2009). Content-Centered collaboration spaces in the cloud. *IEEE Internet Computing, pp. 34-42, September/October, 2009*.

Fling, B. (2009). *Mobile Design and Development*. O'Reilly Media, Inc.

Grossmann, M., Bauer, M., Honle, N., Kappeler, U., Nicklas, D., & Schwarz, T. (2005). Efficiently managing context information for large-scale scenarios. In *Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications*, (pp. 331–340).

Hackmann, G., Haitjema, M., Gill, C., & Roman, G. (2006b). Sliver: A BPEL workflow process execution engine for mobile devices. *Service-Oriented Computing–ICSOC 2006*, (pp. 503–508).

Hackmann, G., Sen, R., Haitjema, M., Roman, G., & Gill, C. (2006a). MobiWork: mobile workflow for MANETs. Tech. rep., Washington University, Department of Computer Science and Engineering.

Henricksen, K., & Indulska, J. (2006). Developing context-aware pervasive computing applications: Models and approach. *Pervasive and Mobile Computing*, *2*(1), pp. 37–64.

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. *MIS Quarterly*, *28*(1), pp. 75–105.

Hollingsworth, D. (1995). Workflow management coalition: The workflow reference model. [Online]. www.wfmc.org [Accessed on 15 march 2011].

Hull, R. (2008). Artifact-centric business process models: Brief survey of research results and challenges. *On the Move to Meaningful Internet Systems: OTM 2008*, (pp. 1152–1163).

Hull, R., Damaggio, E., Fournier, F., Gupta, M., Heath, F., Hobson, S., Linehan, M., Maradugu, S., Nigam, A., Sukaviriya, P., et al. (2011). Introducing the guard-stage-milestone approach for specifying business entity lifecycles. *Web Services and Formal Methods*, (pp. 1–24).

Informatica (2007). BPEL process structure. [Online] Available at: www.activevos.com/content/developers/education/bpel/unit04_bpelprocesses.pdf [Accessed on 20 May 2012].

Kloppmann, M., Koenig, D., Leymann, F., & Pfau, G. (2005). WS-BPEL extension for peopleBPEL4People.

Korpipaa, P., Mantyjarvi, J., Kela, J., Keranen, H., & Malm, E. J. (2003). Managing context information in mobile devices. *Pervasive Computing, IEEE*, *2*(3), pp. 42–51.

Kramer, D., Kocurova, A., Oussena, S., Clark, T., & Komisarczuk, P. (2011). An extensible, self contained, layered approach to context acquisition. In *Proceedings of the Third International Workshop on Middleware for Pervasive Mobile and Embedded Computing*, M-MPAC '11, (pp. 6:1–6:7). New York, NY, USA: ACM.

Kumaran, S., Liu, R., & Wu, F. (2008). On the duality of information-centric and activity-centric models of business processes. In *Proceedings of the 20th international conference on Advanced Information Systems Engineering*, (pp. 32–47).

Kunze, C. P., Zaplata, S., & Lamersdorf, W. (2006). Mobile process description and execution. *In Proceedings of the 6th IFIP WG 6.1 international conference on Distributed Applications and Interoperable Systems*, (pp. 32–47).

Kunze, C. P., Zaplata, S., & Lamersdorf, W. (2007). Mobile processes: Enhancing cooperation in distributed mobile environments. *Journal of Computers*, *2*(1), pp. 1–11.

Künzle, V., & Reichert, M. (2011). PHILharmonicFlows: towards a framework for object-aware process management. *J. Softw. Maint. Evol.*, *23*(4), pp. 205–244.

Liang, S., & Bracha, G. (1998). Dynamic class loading in the java virtual machine. *SIGPLAN Not.*, *33*, pp. 36–44.

Liu, R., Bhattacharya, K., & Wu, F. (2007). Modeling business contexture and behavior using business artifacts. In *Advanced Information Systems Engineering*, (pp. 324–339).

Love, S. (2009). *Handbook of Mobile Technology Research Methods*. Nova Science Publishers.

Mahmoud, Q. H. (2004). *Middleware for communications*. John Wiley and Sons.

March, S. T., & Smith, G. F. (1995). Design and natural science research on information technology. *Decision Support Systems*, *15*(4), pp. 251–266.

Nandi, P., & Kumaran, S. (2005). Adaptive business objectsa new component model for business integration. In *Proceedings of International Conference on Enterprise Information Systems (ICEIS)*, (pp. 179–188).

Nitzsche, J., Van Lessen, T., Karastoyanova, D., & Leymann, F. (2007a). BPEL for semantic web services (BPEL4SWS). In *Proceedings of the 2007 OTM confederated international conference: On the move to meaningful internet systems*, (pp. 179–188).

Nitzsche, J., Van Lessen, T., Karastoyanova, D., & Leymann, F. (2007b). BPEL light. In *Proceedings of the 5th international conference on Business process management*, (pp. 214–229).

Nurcan, S. (2008). A survey on the flexibility requirements related to business processes and modeling artifacts. In *Proceedings of the Proceedings of the 41st Annual Hawaii International Conference on System Sciences*, (pp. 378–387).

OASIS (2007). BPEL specification, version 2. [Online] Available at: http://docs.oasisopen.org/wsbpel/2.0/OS/wsbpelv2.0OS.pdf [Accessed on 13 March 2011].

OMG (2007). Unified Modelling Language - UML. [Online] Available at: http://www.uml.org/ [Accessed on 11 October 2012].

OMG (2011). BPMN specification, version 2. [Online] Available at: http://www.omg.org/spec/BPMN/2.0/PDF/ [Accessed on 10 April 2012].

Oren, E., & Haller, A. (2005). Formal frameworks for workflow modelling. *Digital Enterprise Research Institute, National University of Ireland, Technical Report*, (pp. 4–7).

Pajunen, L., & Chande, S. (2007). Developing workflow engine for mobile devices. In *Enterprise Distributed Object Computing Conference, 2007. EDOC 2007. 11th IEEE International*. IEEE.

Peltz, C. (2003). Web services orchestration and choreography. *Computer*, *36*(10), 46–52.

Potts, J. (2008). *Alfresco Developer Guide*. Packt Publishing.

Pryss, R., Tiedeken, J., Kreher, U., & Reichert, M. (2011). Towards flexible process support on mobile devices. *Information Systems Evolution*, (pp. 150–165).

Redding, G., Dumas, M., Hofstede, A. H. M., & Iordachescu, A. (2010). A flexible, object-centric approach for business process modelling. *Service Oriented Computing and Applications*, *4*, pp. 191–201.

Reichle, R., Wagner, M., Khan, M., Geihs, K., Lorenzo, J., Valla, M., Fra, C., Paspallis, N., & Papadopoulos, G. (2008). A comprehensive context modeling framework for pervasive computing systems. In *Distributed applications and interoperable systems*, (pp. 281–295).

Rosemann, M., Recker, J., & Flender, C. (2008). Contextualisation of business processes. *International Journal of Business Process Integration and Management*, *3*(1), pp. 47–60.

Rosemann, M., & Recker, J. C. (2006b). Context-aware process design: Exploring the extrinsic drivers for process flexibility. In *The 18th International Conference on Advanced Information Systems Engineering. Proceedings of Workshops and Doctoral Consortium*, (pp. 149–158).

Rosemann, M., Recker, J. C., Flender, C., & Ansell, P. D. (2006a). Understanding context-awareness in business process design. *Faculty of Science and Technology; Institute for Creative Industries and Innovation*.

Rosson, M. B., & Carroll, J. M. (2002). Scenario-based design. In *The human-computer interaction handbook*, (pp. 1032–1050).

Saidani, O., & Nurcan, S. (2007). Towards context aware business process modelling. In *Proceedings of Workshop on Business Process Modelling, Development, and Support*.

Saidani, O., & Nurcan, S. (2009). Context-awareness for adequate business process modelling. In *Third International Conference on Research Challenges in Information Science, 2009. RCIS 2009*, (pp. 177–186). IEEE.

Salber, D., Dey, A. K., & Abowd, G. D. (1999). The context toolkit: aiding the development of context-enabled applications. In *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*, (pp. 434–441).

Schoder, D., & Fischbach, K. (2003). Peer-to-peer prospects. *Commun. ACM*, *46*(2), pp. 27–29.

Schonenberg, H., Mans, R., Russell, N., Mulyar, N., & Aalst, W. (2008a). Process flexibility: A survey of contemporary approaches. *Advances in Enterprise Engineering I*, (pp. 16–30).

Schonenberg, H., Mans, R., Russell, N., Mulyar, N., & van der Aalst, W. (2008b). Towards a taxonomy of process flexibility. In *CAiSE Forum*, (pp. 81–84).

Sen, R. (2008). *Supporting collaboration in mobile environments*. Ph.D. thesis, Washington University.

Shariff, M., Bhandari, A., Majumdar, P., & Choudhary, V. (2009). *Alfresco 3 Enterprise Content Management Implementation*. Birmingham: Packt Publishing.

Sipser, M. (2006). *Introduction to the Theory of Computation*. Thomson Course Technology: Boston, 2 ed.

Smanchat, S., Ling, S., & Indrawan, M. (2008). A survey on context-aware workflow adaptations. In *Proceedings of the 6th International Conference on Advances in Mobile Computing and Multimedia*, (pp. 414–417). ACM Press.

Vaishnavi, V., & Kuechler, W. (2004). Design science research in information systems. [Online]. Available at: http://www.desrist.org/desrist. [Accessed on 13 March 2011].

Vanderfeesten, I., Reijers, H., & Aalst, W. (2008). Product based workflow support: dynamic workflow execution. In *Advanced Information Systems Engineering*, (pp. 571–574).

Vanderfeesten, I., Reijers, H. A., & Aalst, W. M. P. (2011). Product-based workflow support. *Information Systems*, *36*(2), pp. 517–535.

Vara, J. L., Ali, R., Dalpiaz, F., Snchez, J., & Giorgini, P. (2010). COMPRO: a methodological approach for business process contextualisation. In R. Meersman, T. Dillon, & P. Herrero (Eds.) *On the Move to Meaningful Internet Systems: OTM 2010*, (pp. 132–149). Berlin, Heidelberg: Springer Berlin Heidelberg.

Wahler, K. (2009). *A Framework for Integrated Process and Object Life Cycle Modeling*. Ph.D. thesis, University of Zurich.

Weber, B., Sadiq, S., & Reichert, M. (2009). Beyond rigidity dynamic process lifecycle support. *Computer Science-Research and Development*, *23*(2), pp. 47–65.

Wieland, M., Kopp, O., Nicklas, D., & Leymann, F. (2007). Towards Context-Aware workflows. In *CAiSE´07 Proceedings of the workshops and doctoral consortium*, (pp. 11–15).

Wissen, B., Palmer, N., Kemp, R., Kielmann, T., & Bal, H. (2010). ContextDroid: An expression-based context framework for Android. In *Proceedings of PhoneSense 2010*.

Wohlin, C., Runeson, P., Hst, M., Ohlsson, M. C., Regnell, B., & Wessln, A. (2012). *Experimentation in Software Engineering*. Springer.

Yan, J., Yang, Y., & Raikundalia, G. K. (2006). SwinDeW-a P2P-based decentralized workflow management system. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, *36*(5), pp. 922–935.

Yu, J., & Buyya, R. (2006). A taxonomy of workflow management systems for grid computing. *Journal of Grid Computing*, *3*, pp. 171–200.

Published Papers

- Kocurova, A., Oussena, S., Komisarczuk, P. and Clark, T. (2013) MobWEL - Mobile Context-Aware Content-Centric Workflow Execution Language. 3rd International Conference on Advanced Collaborative Networks, Systems and Applications (COLLA 2013), Nice, France, July 21 - 26, 2013.
- Kocurova, A., Oussena, S., Komisarczuk, P. and Clark, T. (2012) Context-Aware Content- Centric Collaborative Workflow Management for Mobile Devices. 2nd International Conference on Advanced Collaborative Networks, Systems and Applications (COLLA 2012), Venice, Italy, June 24 - 29, 2012.
- Kocurova, A., Oussena, S., Komisarczuk, P., Clark, T. and Kramer, D. (2012) Towards improved distributed collaborative workflow management for mobile devices. In Lecture Notes in Business Information Processing: Data-Driven Process Discovery and Analysis, 2012.
- Kramer, D., Kocurova, A., Oussena, S., Clark, T., Komisarczuk, P. (2011) An extensible, self contained, layered approach to context acquisition. M-MPAC, Middleware 2011.
- Kocurova, A. (2011) Towards improved distributed collaborative workflow management for mobile devices. In Pre-proceedings of 1st Symposium on Data-driven Process Discovery and Analysis (SIMPDA'11), PhD Seminar, Campione d'Italia, Italy, June 29 - July 1, 2011.
- Sauer, Ch., Kocurova, A., Kramer, D., and Roth-Berghofer, T. (2012) Using canned explanations within a mobile context engine. Exact 2012
- Kocurova, A., Clark, T., and Oussena, S. (2012) Applied metamodelling to Collaborative Document Authoring. 2nd Workshop on Model Driven Approaches in System Development (FedCSIS/MDASD 2012), Wroclaw, Poland, September 9-12, 2012.