Automating the development of Physical Mobile Workflows

Pau Giner Blasco

UNIVERSIDAD POLITECNICA DE VALENCIA

Supervisor:
Dr. Vicente Pelechano Ferragud

Centro de Investigación en
Métodos de Producción de Software

Pau Giner

Automating the development of Physical Mobile Workflows

A Model Driven Engineering approach

PhD Thesis, January 2010

PROS
Centro de Investigación en Métodos de Producción de Software

UNIVERSIDAD POLITECNICA DE VALENCIA

Pau Giner

Automating the development of Physical Mobile Workflows

A Model Driven Engineering approach

PhD Thesis, January 2010

Automating the development of Physical Mobile Workflows: A Model Driven Engineering approach

This report was prepared by
Pau Giner

Supervisors
Vicente Pelechano

Centro de Investigación en Métodos de Producción de Software
Universidad Politécnica de Valencia
Camino de Vera s/n, 46022 Valencia
Spain

Tel: (+34) 963 877 007 (Ext. 83530)
Fax: (+34) 963 877 359
Web: http://www.pros.upv.es

Release date: 18-01-2010
Edition: First

Comments: A thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science at the Universidad Politécnica de Valencia.

Preface

The inception of this work has its roots many years before I was engaged in its elaboration. I remember myself being a child in front of a pile of random things at my grandparent's attic. I was fascinated trying to find out what secret stories were these objects keeping inside. Now, many years later, I have the opportunity to help these objects to tell their story.

As illustrated by the Heraclitus's river, objects and people are part of processes of continuous change. When we are able to flow through this river, our environment becomes like the thematic series of Richard Lohse or the ever-changing figure of a kaleidoscope. Each object turns into a colorful piece of our society. This fetichistic movement can help us to escape hyperreality since we can directly ask the world whatever we want to know.

This work is about touching, pointing at, and playing with, the real world. Having the opportunity with this work to look at the world with the eyes of a child has been a great challenge and a unique experience.

Valencia, January 2010 *Pau Giner*

Acknowledgements

This work would not have been possible without the support and encouragement of many people. I hope they have learnt half as much from me as I have from them.

Foremost, I would like to thank my supervisor, Dr. Vicente Pelechano. He trusted in my work from the very beginning. Thanks to his research insight I was able to start working in such an interesting field. I also like to express my gratitude to Prof. Oscar Pastor for his direction of our research center and for showing me that it is always possible to find the correct way (or highway).

A special gratitude is due to Carlos, my partner in crime, for sharing his bullet-proof working spirit. Tanks to Joan, Vicky, Manoli and Pedro, because working with them has been an experience that helped me to grow as a researcher but also as a person. I would like to thank Fani, Mario, Miriam, Paqui, Nacho and Isma for the good times we spent together in the lab and out of it. I am grateful to Ana for all the time she saved me with her work.

I would also like to express my thanks to the people that made me forget everyday that I was in my workplace. I have shared conversations, meals, delayed flights and even kidnap attempts with them. Thanks to Bea, Giovanni, Paco, Nathalie, and José Luis because their different kinds of humour make it possible to have fun with almost anything. I would also like to thank the rest of friends and colleagues from the ProS

research center for their collaboration.

Special thanks for the people that, from the place that they occupy in my soul, have injected me the dose of love that is required for feeling alive. Thanks to Patri for her unconditional support. I wish to thank my parents, Pilar and Josep for their constant encouragement and love. They have always supported me to do my best in all matters of life.

Abstract

The Internet of Things vision proposes a tight integration between real-world elements and Information Systems. Information Systems can be aware of physical objects thanks to Automatic Identification (Auto-ID) technologies such as Radio Frequency Identification (RFID). When physical elements participate actively in business processes, the use of humans as information carriers is avoided. Thus, errors are reduced and process efficiency is improved.

Although developing this kind of systems is feasible, the technological heterogeneity in Auto-ID and the fast-changing requirements of business processes hinders their construction, maintenance and evolution. Therefore, there is a need to move from ad-hoc solutions to sound development methods in order to assure the quality of the final product.

This thesis, based on Model Driven Engineering foundations, presents a development process for the construction of this kind of systems. The main goal of the present work is to systematize the development of business process-supporting systems that integrate physical elements. The development process defined covers from the system specification to its implementation and it is focused on the particular requirements of the linkage between physical and virtual worlds.

For the system specification, a modeling language is defined to cope with the particular requirements of the Internet of Things domain. From this specification, following a set of systematic steps, a software solution

is obtained. This solution is supported by an architecture specifically designed to cope with the Internet of Things requirements and to survive to technological evolution.

The proposal has been applied in practice with end-users. Although the development process is not completely automated, the guidance offered and the formalization of the involved concepts was proven helpful to raise the abstraction level of development avoiding to deal with technological details.

Resumen

La visión de la "Internet de las Cosas", hace énfasis en la integración entre elementos del mundo real y los Sistemas de Información. Gracias a tecnologías de Identificación Automática (Auto-ID) cómo RFID, los sistemas pueden percibir objetos del mundo físico. Cuando éstos participan de manera activa en los procesos de negocio, se evita el uso de los seres humanos como transportadores de información. Por tanto, el número de errores se reduce y la eficiencia de los procesos aumenta.

Aunque actualmente ya es posible el desarrollo de estos sistemas, la heterogeneidad tecnológica en Auto-ID y los requisitos cambiantes de los procesos de negocio dificultan su construcción, mantenimiento y evolución. Por lo tanto, es necesaria la definición de soluciones que afronten la construcción de estos sistemas mediante métodos sólidos de desarrollo para garantizar la calidad final del producto.

Partiendo de las bases de la Ingeniería Dirigida por Modelos (MDE), esta tesis presenta un proceso de desarrollo para la construcción de este tipo de sistemas. Este proceso cubre desde la especificación del sistema hasta su implementación, centrándose en los requisitos particulares del enlace entre los mundos físico y virtual.

Para la especificación de los sistemas se ha definido un Lenguaje de modelado adaptado a los requisitos de la "Internet de las Cosas". A partir de esta especificación se puede obtener una solución software de manera sistemática.

Como validación de la propuesta, ésta se ha aplicado en la práctica con usuarios finales. Pese a que el proceso de desarrollo no ofrece una automatización completa, las guías ofrecidas y la formalización de los conceptos implicados ha demostrado ser útil a la hora de elevar el nivel de abstracción en el desarrollo, evitando el esfuerzo de enfrentarse a detalles tecnológicos.

Resum

La visió de l'"Internet de les Coses", emfatitza la integració entre els elements del món real i els Sistemes d'Informació. Gràcies a les tecnologies d'Identificació Automàtica (Auto-ID) com l'RFID, els sistemes poden percebre objectes del món físic. Quan aquestos participen activament en els processos de negoci, s'evita l'ús dels éssers humans com a transportadors d'informació. Per tant, el nombre d'errors es redueix i l'eficiència dels processos augmenta.

Tot i que actualment ja és possible el desenvolupament d'aquestos sistemes, l'heterogenïtat tecnològica en Auto-ID i els requeriments canviants dels processos de negoci dificulten la construcció, manteniment i evolució d'aquells. Per tant, és necessaria la definició de solucions per abordar la construcció d'aquestos sistemes mitjançant mètodes sòlids de desenvolupament que garantisquen la qualitat final del producte.

Prenent com a base l'Enginyeria Dirigida per Models (MDE), aquesta tesi presenta un procés de desenvolupament per a la construcció d'aquest tipus de sistemes. El procés cobreix des de l'especificació del sistema fins la seua implementació, centrant-se en els requeriments particulars de l'enllaç entre el món físic i virtual.

Per a l'especificació dels sistemes s'ha definit un llenguatge de modelat adaptat als requeriments de l'"Internet de les Coses". A partir d'aquesta especificació és possible obtenir una solució de programari d'una manera sistemàtica.

Com a validació de la proposta, aquesta ha estat aplicada en la práctica amb usuaris finals. Tot i que el procés de desenvolupament no proporciona una automatització completa, les guies proporcionades i la formalització dels conceptes implicats han mostrat la seua utilitat per a elevar el nivell d'abstracció en el desenvolupament, evitant l'esforç d'enfrontar-se a detalls tecnològics.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Information Systems have existed for a long time. Humans have faced the need for recording and transmitting information long before the invention of the computer. The introduction of Information Technologies creates a digital world where information can be automatically processed, improving the Information System efficiency. However, computers have a limited vision of the real-world they are managing. Thus, there is still a challenge in **automating the linkage between digital and physical worlds**.

Nowadays, Information Systems that deal with real-world objects (such as baggage pieces in an airport or products in a supermarket) are normally informed by humans. This use of humans as information carriers becomes inefficient and error-prone. The gap between the physical and the digital world commonly results in mishandled luggage or long queues at the supermarket.

The Internet of Things vision (Gershenfeld et al., 2004) is about reducing this gap to make daily activities more fluent. By providing a digital identity to real-world objects, Information Systems can handle them in an automatic way. This enables physical objects to participate

actively in business processes by reducing the gap between physical and virtual worlds (Strassner & Schoch, 2002). In addition, the widely availability of mobile devices with advanced capabilities allow users to access the information and services where they need them.

This work deals with **physical mobile workflows**, which are business processes that take advantage of the capabilities of mobile devices for the identification of physical elements. The high heterogeneity in identification technologies, the fragmentation in mobile platforms and the fast-changing nature of business processes, make it hard to develop this kind of systems in a sound manner. In this work, modeling techniques are applied in order to face the development of such systems from a higher abstraction level.

The rest of this chapter is organized as follows: Section 1.1 explains the purpose of this work. Section 1.2 details the problem that the present thesis resolves. Section 1.3 introduces the goals defined for this work. Section 1.4 describes the approach followed in this thesis to fulfill the detected goals. Section 1.5 explains the context in which the work of this thesis has been performed. Finally, Section 1.6 gives an overview of the structure of this document.

1.1 Motivation

The capabilities of mobile devices for the Automatic Identification of real-world elements (Auto-ID) are becoming widespread. Some examples of these capabilities are the use of on-board cameras for decoding visual markers or the use of Near Field Communication (NFC) technology for reading RFID contactless tags. Auto-ID technologies alleviate the inherent I/O limitations of mobile devices when dealing with user tasks (Radi & Mayrhofer, 2008).

When leveraging the Auto-ID capabilities of mobile devices, daily activities become more fluid for users. Information can be transferred automatically between physical and digital spaces. Thus, people can focus on their real world activities while the system, hidden in the background, controls the business process in an unobtrusive way. For example, customers in a supermarket can directly access meaningful in-

formation such as the presence of ingredient to which they are allergic, opinions of their friends about the product, and price comparisons with similar products just by taking a picture of its barcode with their mobile devices. Then, they can exit the supermarket with their cart and the checkout is performed as the products are automatically detected at the exit door.

Integrating real-world objects in business processes has been demonstrated successful, reducing media breaks, human errors and delayed information problems (Strassner & Schoch, 2002). Many benefits are obtained in economic (Langheinrich et al., 2002) and process improvement terms (Fleisch, 2001; Sandner et al., 2005). A better integration of real and virtual worlds not only improves business processes, but also enables new business models (Fano & Gershman, 2002; Fleisch & Tellkamp, 2003).

Facing the development of this kind of systems, however, is not an easy task. Business processes are constantly changing, which in turn requires the corresponding evolution in the supporting Information System. In addition, systems in the Internet of Things context, involve a great diversity of technologies to bridge physical and digital worlds. This heterogeneity forces the developer to know the details of each technology involved in the system, making these systems difficult to develop. From the methodological perspective, there is a need for a solid development method that can free developers from technological details and allow a fast propagation of requirement changes to technological solutions.

1.2 Problem Statement

The development of applications for the Internet of Things is an emerging research topic. The above discussion indicates that some problems still need to be considered. This work seeks to improve the development of business process-supporting applications in the context of the Internet of Things. This challenge is faced from an engineering perspective where it has been considered at design, evaluation and implementation levels. The challenges faced at each level can be stated by the following

three research questions:

Research question 1. How should the requirements for the integration of physical elements be captured in business process specifications?

Research question 2. How to simulate business processes that integrate physical elements in order to validate the captured requirements in an early stage of the development?

Research question 3. How can business process specifications be systematically mapped to technological solutions that allow the process behavior to fit the requirements?

These research questions are analyzed and answered in the following sections.

1.3 Thesis Goals

The main goal of this thesis is to define a development process for the construction of business process-supporting applications that integrate real-world elements. Since business process modeling techniques have proven to be effective in reducing the development effort for business process support systems (Smith & Fingar, 2003), we have applied them to the particular domain of interest for this work.

First of all, regarding **research question 1**, one of the main goals of this work is the study of the linkage between the physical and the virtual space in business processes. Current business process modeling techniques provide little support for specifying how physical elements are integrated in a business process. In order to enable the specification of this aspect, it is required to define *what* to capture, and *how* to do it. From a modeling perspective, there is a need for the clear definition of the concepts to be captured and the modeling primitives for capturing them.

Regarding **research question 2**, another goal of this work is to determine whether a business process designed is appropriate or not by

avoiding to deal with technological details. Mechanisms must be provided in order to obtain valuable feedback regarding the user experience and the process performance increase in an early stage of development with minimal effort. This is required for the application of iterative design, which provides frequent feedback about the potential of the business process designs.

Regarding **research question 3**, one of the goals of the present work is to reduce the error-proneness of this kind of developments. In order to do so, a systematic method is defined to obtain a working system from specifications by following a sequence of well-defined steps. Systems in the business process area are usually built by composing existing functionality. However, the high dynamism of physical mobile workflows requires that the services involved in the process must be re-arranged at run-time in order to fulfill the design requirements (e.g., not disturbing the user when he/she is engaged in a relevant task).

1.4 The Proposed Solution

Model Driven Engineering (MDE) (Schmidt, 2006) proposes the use of models as the basis for system development. A model is a simplification of a system, built with an intended goal in mind, that should be able to answer questions in place of the actual system (Bézivin & Gerbé, 2001). The use of models (such as model of planes in a wind tunnel or models of software systems) in engineering has a twofold benefit. On the one hand, models **guide the development** of a system. On the other hand, models allow to **reason about the system** avoiding to deal with technical details.

Business Process Management (BPM) promotes a continuous re-engineering cycle for business processes. For BPM to succeed it is required a seamless transition from process modeling, simulation and execution. However, current BPM approaches are focused on the digital world and do not take into account the particularities of the physical-virtual linkage. In this work we provide solutions that complement the BPM techniques in order to consider the specific properties of this linkage.

A **design method** has been defined in order to capture the activities that take place in the process and the physical elements involved. The defined method enables designers to specify the way in which a user can interact with the environment and to which degree the system functionality must intrude the users mind.

An **evaluation method** is provided in order to simulate the integration of physical elements in business processes. This enables to anticipate business process results without actually implementing the supporting system. The method is fast to apply and it allows to reproduce a level of user experience that is considered to be very close to what users expect from a final system. The method allows to receive relevant feedback in terms of user experience and process improvement, which is essential to reconsider the designs prior to the development of the final system.

A **software architecture** is defined in order to integrate the components required for executing the business process. In order to provide a low-coupled and model-based solution, we make use of business process orchestration engines and reconfiguration engines. In this way, we can define how the different services involved in the process are coordinated (by means of an orchestration engine) and adapted to the current business context (by means of a reconfiguration engine) using executable models.

1.5 Thesis Context

This Thesis is being developed in the context of the research center *Centro de Investigación en Métodos de Producción de Software* of the *Universidad Politécnica de Valencia*. The work that has made the development of this thesis possible is in the context of the following research government projects:

- DESTINO: Desarrollo de e-Servicios para la nueva sociedad digital. CYCIT project referenced as TIN2004-03534.
- SESAMO: Construcción de Servicios Software a partir de Modelos. CYCIT project referenced as TIN2007-62894.

- OSAMI Commons: Open Source Ambient Intelligence Commons. ITEA 2 project referenced as TSI-020400-2008-114.
- Atenea: Arquitectura, Middleware y Herramientas. ProFIT project referenced as FIT-340503-2006-5.
- "Internet de las Cosas como soporte a Procesos de Negocio". Primeros proyectos de Investigación de la UPV, referenced as PAID-06-09 number 2920.

1.6 Thesis Structure

The approach followed in this work follows the BPM principles. The work has been structured to reflect this process. First, Chapter 2 gives an overview of some relevant concepts related to Business Process Management, mobile platforms, and Automatic Identification technologies in which this work relies. Chapter 3 compares this work with similar approaches in the area. Chapter 4 defines the design method that is followed in our approach to capture the requirements for physical mobile workflows. In BPM terms, this involves to provide support to the modeling and simulation stages of the development. Chapter 5 defines an architecture that supports physical mobile workflow execution and makes the architecture usable at modeling level. In order to do so, the architecture is formalized in a metamodel and model transformation techniques are used to automate the development based on this architecture. Chapter 6 extends the architecture capabilities in order to handle models at run-time for providing physical mobile workflows with a greater degree of adaptation. Chapter 7 details how the proposal has been validated. Finally, Chapter 8 summarizes the contributions and provides some insights about further work.

Chapter 2

Background

This work deals with systems that take advantage of the Auto-ID capabilities of mobile devices in order to improve the business processes in an organization. This is what we call **physical mobile workflows**. In order to define this specific domain, it has been situated with respect to different research areas that have some aspects in common. As it is shown in Fig. 2.1, physical mobile workflows are part of three disciplines: Business Process Management, the Internet of Things, and Mobile Applications.

This work relies on different concepts and technologies from these areas. In order to clarify the foundations in which our approach relies, different technologies and techniques are introduced in this chapter. The remainder of the chapter is structured as follows. Section 2.1 introduces concepts to support the modeling and execution of business processes. Section 2.2 provides an overview of the technologies and techniques that enable the construction of systems for the Internet of Things. Section 2.3 provides an overview of the impact of mobility in the development of a system and mobile platforms for coping with these requirements. Finally, Section 2.4 concludes the chapter.

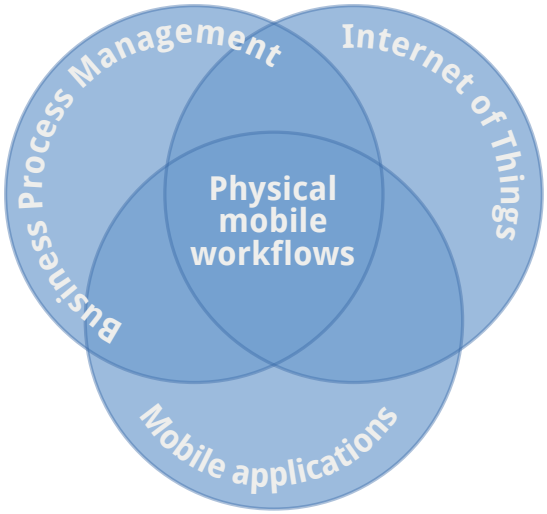


Figure 2.1: Application domains involved in this work

2.1 Business Process Management

Business Process Management (BPM) is a set of techniques and technologies that enables a company to build executable processes that span across multiple organizations or systems, usually through visual workflow steps (Davis, 2009). Different initiatives have emerged for supporting BPM such as Business Process Modeling Notation (BPMN) (OMG, 2006) and Web Services Business Process Execution Language (WS-BPEL) (Alves et al., 2007). BPM techniques have proven to be effective in reducing the development effort for business process support systems (Smith & Fingar, 2003).

2.1.1 Business Process Modeling

Different notations are used for the modeling of business processes such as UML Activity diagrams (Dumas & ter Hofstede, 2001), IDEF (Mayer et al., 1992), ebXML BPSS (Hofreiter et al., 2002) or Business Process Modeling Notation (BPMN) (OMG, 2006). The common characteristics of this notations is their capability for modeling the sequence of activi-

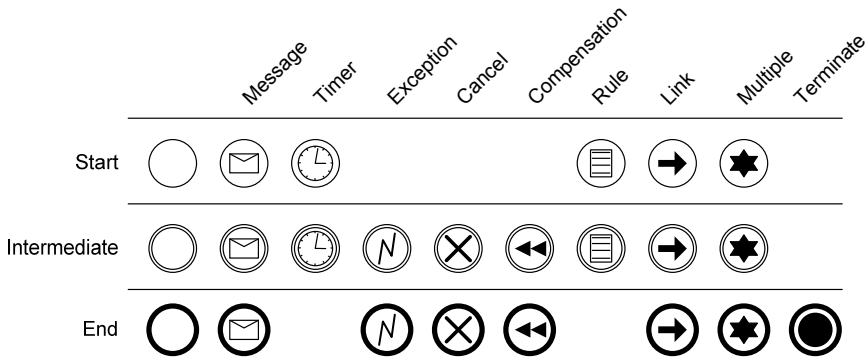


Figure 2.2: Event types defined by BPMN

ties, the participants involved in the process and the data or messages interchanged between them.

The BPMN standard was developed by the BPMI (Business Process Management Initiative) to provide a notation that could be easily understood by all business stakeholders. The specification was adopted by the Object Management Group (OMG) as the standard notation for the modeling of business processes.

In order to provide an overview of the expressivity obtained by business process modeling notations in general and BPMN in particular, the most relevant building blocks included in BPMN are described below. This set of elements is organized in four categories which are Flow Objects, Connecting Objects, Swimlanes and Artifacts:

Flow objects

These elements constitute the main graphical elements to define the behavior of a Business Process. These refer to:

Events. An event is something that "happens" during the course of a business process. These events affect the flow of the process and usually have a cause (trigger) or an impact (result). Events are

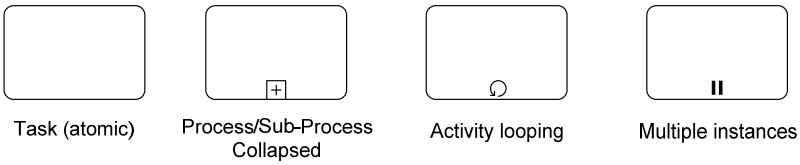


Figure 2.3: Activity types defined by BPMN

depicted as circles with open centre to allow internal markers to differentiate different triggers or results. There are three types of events, based on when they affect the flow: Start, Intermediate, and End. Figure 2.2 shows the complete set of events defined by the notation.

Activities. An activity is a generic term for work performed within a business process. An activity can be atomic or non-atomic (compound). The types of activities considered are: *Process*, *Sub-Process*, and *Task*. Only *Tasks* and *Sub-Processes* define a specific graphical object (a rounded rectangle). On the contrary, Processes are built as a set of activities and the controls that sequence them. In addition, *Tasks* and *Sub-Processes* include a set of attributes which determine if these activities are repeated or performed just once. This repetition can be performed either sequentially (loop marker) or in parallel (parallel marker). Figure 2.3 depicts the different types of activities and the markers available to specify when the activity can be repeated and how.

Gateways. A Gateway is used to control the divergence and convergence of Sequence Flows. Thus, it will determine branching, forking, merging, and joining of paths. Internal markers will indicate the type of behavior control. Figure 2.4 depicts the different types of gateways provided by the notation.

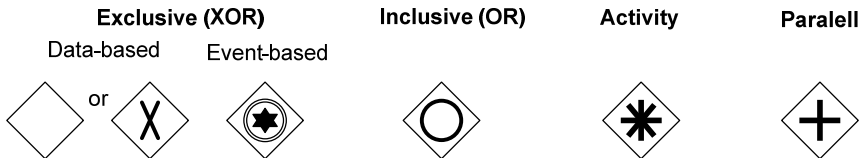


Figure 2.4: Gateway types defined by BPMN

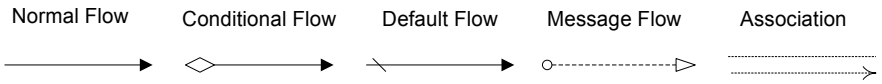


Figure 2.5: Connecting object types defined by BPMN

Connecting Objects

These elements allow connecting Flow objects or other information. The connecting objects defined by the notation are:

Sequence Flow. It is used to show the order in which activities will be performed in a process. This type of connecting object can in turn be specialized in *Normal*, *Conditional* and *Default Flow*.

Message Flow. It is used to show the flow of messages between two participants that are prepared to send and receive between them. In BPMN, two separate *Pools* in the diagram represent two different participants (e.g., business entities or business roles).

Association. An *Association* is used to associate information with *Flow Objects*. Text and graphical non-Flow Objects (i.e. data objects) can be associated with *Flow Objects*.

Figure 2.5 depicts the different connecting objects defined by the BPMN notation.

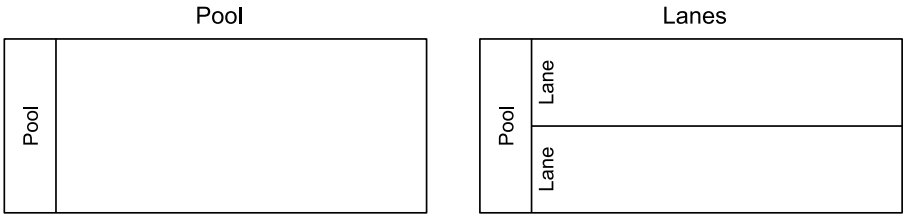


Figure 2.6: Gateway types defined by BPMN

Swimlanes

These elements allow grouping Flow objects based on a particular criterion. This category includes two types of elements (see Fig 2.6) which are:

Pools: represent a participant in a process. Normally it represents an organization and its different parts are represented by lanes in the pool.

Lanes: are used to organize and categorize activities. This is achieved by partitioning the *Pool* in different lanes.

Artifacts

The elements included within this type are introduced into business process models to improve their understanding (see Fig. 2.7). Within this category we find:

Data Object. This element provides information about activity requirements and results.

Group. The grouping can be used for documentation or analysis purposes. Groups can also be used to identify the activities of a distributed transaction that is shown across *Pools*.

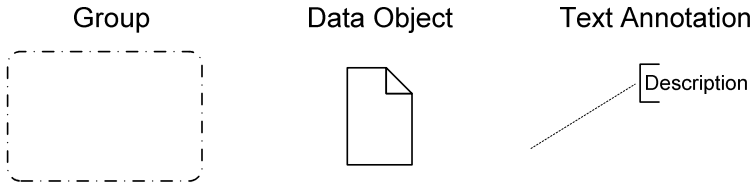


Figure 2.7: Artifact types defined by BPMN

Annotation: *Text Annotations* are the mechanism provided to modelers to introduce additional information for the reader of a BPMN diagram.

In addition to these four categories, the BPMN notation handles advanced modeling concepts such as exception handling, transactions and compensation.

2.1.2 Business Process Execution

Business process modeling becomes quite useful for capturing requirements, but in order to have a successful BPM, executable definitions of the process are needed. Many Information Systems supporting business processes contain the business process logic scattered through the system. This results in monolithic applications that become difficult to maintain and evolve. With an executable definition of the business process, the knowledge about the process is centralized and the process can be updated easily resulting in the immediate update of the corresponding Information System.

Different languages appeared to allow the definition of executable business processes such as XML Process Definition Language (XPDL), Yet Another Workflow Language (YAWL) or Web Service Business Process Execution Language (WS-BPEL) (Alves et al., 2007).

WS-BPEL is one of the most widespread languages in the business process execution area. WS-BPEL is an XML-based language for Web

Services orchestration. It provides constructs for the coordinated invocation of different Web Services. Many solutions exist, both commercial and open source, to provide execution support for this standard. Microsoft BizTalk, Oracle BPEL, Intalio BPMS, ActiveBPEL or Apache ODE are some of the business process execution engines with support for WS-BPEL.

WS-BPEL covers different aspects required for process execution. It includes a property-based message correlation mechanism, XML and WSDL typed variables, an extensible language plug-in model to allow writing expressions and queries in multiple languages (XPath is supported by default) and structured-programming constructs including *if-then-elseif-else*, *while*, *sequence* (to enable executing commands in order) and *flow* (to enable executing commands in parallel).

WS-BPEL processes interact with external web services in two ways: (1) invoking operations on other web services, and (2) receiving invocations from clients (either the client that initiated the process or an external system involved in an asynchronous communication). WS-BPEL defines the relationship with external entities using the *Partner Link* concept. The Partner Link has a name, a type (that defines the roles required for communication) and indicates which of the roles it plays. In this way, the functionality required by each party is exposed by the process engine to allow callback invocations.

Another interesting aspect of WS-BPEL is that there exist mappings (OMG, 2006; Ouyang et al., 2006; Recker & Mendling, 2006) that cover the gap between modeling notations such as BPMN and WS-BPEL. Although there is not a direct equivalence between both notations (Recker & Mendling, 2006), model transformations were defined to bridge them (Giner et al., 2007a) for a representative subset of their elements. Thus, business processes modeled with BPMN can be translated automatically to an executable WS-BPEL definition.

2.1.3 Analysis and Discussion

Business process management area has gained momentum from both academy an industry. Different kinds of solutions exist, from high level modeling notations to technological solutions that implement them.

However, business process management is mainly constrained to the digital world. When facing the integration of physical elements it is done at technological level. For example, using WS-BPEL any system accessible by means of Web Services can be integrated. However, dealing with Auto-ID particularities in the process implementation makes the process description difficult to maintain.

The extension of business process modeling notations to integrate physical elements is faced in this work. In this way, identification requirements are faced from a specification perspective. This work extends BPMN since it has become one of the most accepted modeling notations for business processes and the existing WS-BPEL mappings allows to turn specifications in executable systems easily. In this way, physical objects can be modeled and enter the BPM cycle from the beginning.

The possibility of defining business process that integrate seamlessly real-world elements can make analysts to consider the use of Auto-ID in the systems they specify, favoring the implantation of the Internet of Things.

2.2 The Internet of Things

New trends in computation are emerging with the goal of integrating computing services seamlessly in the environment and offering a "natural interaction" to users. Ubiquitous Computing (Ubicomp) (Weiser, 1991), Pervasive Computing (PerCom) (Hansmann et al., 2001), Ambient Intelligence (AmI) (Aarts et al., 2002) or Everyware (Greenfield, 2006) are some of the paradigms that share this goal.

The scenarios envisioned by these initiatives often demand a combination of advanced technologies such as sensor-networks, wearable computers, speech and gesture recognition or machine reasoning capabilities to make the environment behave intelligently. Far from the idea of making objects competing in intelligence with humans, the Internet of Things vision faces the integration of real and virtual worlds following a more practical approach.

The Internet of Things approach proposes **augmenting real-world**

elements with a digital identity to achieve this integration. This is not much demanding for physical elements. Real-world elements are not required to be augmented with complex computing capabilities but just labeled with a unique identifier to make them computer-aware. With a digital identity, the services that Information Systems offer can reach the physical world. This idea was well illustrated by Bruce Sterling in his talk at The Emerging Technology Conference in 2006:

> "We are not talking about a smart object that is ubiquitously computing. But the everyday object, the dumbest, cheapest, most obvious thing we can buy or use. Except it has a unique digital identity, so it becomes trackable, sortable, rankable, and findable in space and time."

Advances in Automatic Identification (Auto-ID) technologies have helped to start moving the Internet of Things vision into reality. Auto-ID enables real-world objects to be taken automatically in consideration by a software system, making objects not human-dependent anymore (Römer et al., 2004). Thanks to Auto-ID, people, places and things can be identified in a myriad of different ways. Radio Frequency Identifications (RFID), Smartcards, barcodes, magnetic strips, and contact memory buttons to name a few, are some Auto-ID enabler technologies with a different degree of automation (Want et al., 1999).

Automatically identifiable objects receive different names such as Spimes (objects that are trackable in space and time), Blogjects (objects that blog), UFOs (Ubiquitous Findable Objects) or EKOs (Evocative Knowledge Objects). This heterogeneity in terminology shows that the Internet of Things is still under construction and the discussions about what the Internet of Things exactly is, are not finished yet.

From the different emerging applications for the Internet of Things, the present work is particularly interested in the integration of real-world objects in business processes. The Internet of Things paradigm can provide numerous benefits in this field, leading to interesting challenges and opportunities in different business areas (Römer et al., 2004) such as source verification, counterfeit protection, one-to-one marketing, maintenance and repair, theft and shrinkage, recall actions, safety and liability, disposal and recycling as well as mass customizing.

There is an increasing interest in the Internet of Things technologies from academia and industry. Many prototypes have been developed to experiment its benefits in contexts as grocery retail (Roussos et al., 2002), aircraft maintenance (Lampe et al., 2004), air pollution monitoring (Kanjo et al., 2009) and home appliance orchestration (Urbanski et al., 2009). Some developments are also carried by different companies to improve their processes (Strassner & Schoch, 2002). Auto-ID is used at *Ford* to speed up replenishment of parts in its production process. The British retailer *Sainsbury* uses Auto-ID technologies to track chilled food products from receiving, through distribution, to the store shelf. *Infineon* uses Auto-ID for Cool Chain Management in order to control the temperature during the transport of chemical products.

The increase in the maturity level of the Internet of Things can be shown in the presence of Auto-ID in many real in-production systems (Federal Trade Commission, 2005; Weinstein, 2005). Auto-ID is present in car keys, employee cards and event tickets to control the access. *SpeedPass* for purchasing gas and goods at *Exxon Mobile* shops or the transport cards for the London and Hong Kong transport system are some examples based on Auto-ID technologies. Some organizations such as US Department of Defense and Wal-Mart require its suppliers the use of the most advanced Auto-ID technologies.

The current technological support for applications in the context of the Internet of Things, and the available frameworks and languages that allow the development for these technologies, are presented below.

2.2.1 Technological Support

Auto-ID is the core technology for the Internet of Things. Many existing technologies permit to attach a digital identity to an object. However, there are six main forms of automatic identifications in use today (Jamali et al., 2007):

Barcodes. Barcodes use an optical machine-readable representation for identifiers. Their use requires a direct line of sight between readers and tags, demanding user intervention in many cases. Despite their limitations, this is the dominant Auto-ID technology in

the retailer industry. Traditional barcodes (i.e., linear barcodes) use parallel lines and the width between them to represent data. There are many different bar code languages. Each language has its own rules for encoding characters (e.g., letter, number, punctuation), printing, decoding requirements, and error checking. Universal Product Code (UPC) and European Article Number (EAN) are some of the numbering schemata used to express identifiers when linear barcodes are used.

More recently, *bi-dimensional barcodes* appeared. These technologies encode information in a two-dimensional image. Images can be made from a matrix of black and white squares (e.g.: *DataMatrix*, *Aztec Code*, *QR Code*, etc.) or by any other symbology (such it is the case of *fiducials* (Bencina & Kaltenbrunner, 2005)). Bi-dimensional barcodes are a practical solution for Auto-ID since they are cheap to produce and camera phones can easily read them.

Contact memory buttons. This consists in a coin-shaped stainless steel container that encapsulates a memory. This memory can be accessed when it is in contact with a *touch probe* (that can act as a reader and writer for the memory). Since this technology requires direct contact, it has been applied in the access control field as a digital key. When disconnected from a host controller the data stored can be retained for over 100 years. Memory in these buttons is passive, containing no battery or internal power source to retain data.

Magnetic Strips. A band of magnetic material on a card is used to store data. By modifying the magnetism of the iron-based magnetic particles that form the band, data can be recorded. The most common use of this technology is for financial cards.

Optical Strips. A panel of laser sensitive material is laminated in a card and is used to store the information in a similar way as it is stored in optical discs such as CD ROMs or DVDs. Since the material is altered by a laser when it is written, the media can be only written once and the data is non volatile. ISO/IEC 11693 and 11694 standards cover the encoding details.

Radio Frequency Identifications (RFID). RFID consists in the transmission of the identity of an object wirelessly, using radio waves. An RFID tag consists of a microchip and an antenna. Since there is no need for line of sight or contact between tags and readers, this technology provides a high level of automation.

RFID tags fall into two general categories, active and passive, depending on their source of electrical power. Active RFID tags contain their own power source, usually an on-board battery. Passive tags obtain power from the signal of an external reader and they simply reflect the energy back. In addition, RFID tags can incorporate read/write memory. Electronic Product Code (EPC) is the numbering scheme defined for encoding identifiers in RFID tags.

The reading distance for RFID varies from a maximum of few meters for passive RFID and hundreds of meters for active RFID. In addition to reaching long distances, there is also interest in short distance reading. RFID based technologies such as Near Field Communication (NFC) provide a 10 centimeter distance communication to ensure the communication is made explicit by the user.

Smartcards. These cards include embedded integrated circuits which can process information. Several types exist. *Memory cards* contain only non-volatile memory storage components, and perhaps some specific security logic. *Microprocessor cards* contain volatile memory and microprocessor components. *Contactless smartcards* rely on technologies such as NFC to avoid the inconvenience of requiring direct contact between readers and cards but restricting the communication to a certain distance for security reasons.

Despite of the diffusion of the above technologies, **text-based identification** is still common nowadays. This consists in writing (or printing) the natural name or part number of an object as simple text. Human intervention is required for reading tags resulting in an inefficient process. However, this mechanism cannot be overlooked as a complementary technology for the above ones. In this way an alternative human-readable identifier is used as backup in case the main Auto-ID technology fails.

2.2.2 Auto-ID Frameworks

Deploying an Auto-ID-enabled system involves a lot more than purchasing the right tags and installing the right readers. To get business value from all of the information collected, companies will need middleware to filter the data. They may need to upgrade enterprise applications and integrate them with Auto-ID middleware. However, the connections to existing software infrastructure results in a mismatch of capabilities and requirements (Sarma, 2004).

The need for defining architectures that support Auto-ID has lead to the development of frameworks and middleware to abstract from the filtering and aggregation tasks needed when tags are processed.

Some middleware is technology-specific such as it is the case of RFID. The EPC Network standard published by EPCglobal (the predominant RFID standardization body) defines a number of functional roles that an RFID middleware must provide as well as the interfaces that must be implemented around these roles. This include the reader, the filtering and collecting middleware, and the EPC information service (EPCIS). Accada (Floerkemeier et al., 2007) (later renamed to Fosstrak) an open source implementation of the EPC network standard was developed by the Auto-ID labs. Other RFID-specific solutions such as SAP's Auto-ID infrastructure (Bornhövd et al., 2004), Siemens RFID Middleware Architecture (Wang & Liu, 2005) or Sun RFID also provide solutions to integrate different RFID readers with Information Systems.

The present work however is interested in the integration of physical elements from a technological independent perspective. In order to achieve this, several initiatives emerged to offer middleware to support Auto-ID in a technology independent fashion. A representation of these proposals are described below.

Web presence. This work (Kindberg et al., 2002) is defined to provide web presence for people, places, and things. The identifier resolution mechanism presented is inspired in the Web not only as a technological foundation but it also adapts metaphors from the Web to the physical world such as the hyperlink concept. Identifier resolution is presented as a way to link the physical world

with virtual Web resources. In this paradigm, designed to support nomadic users, the user employs a handheld, wirelessly connected, sensor-equipped device to read identifiers associated with physical entities. The identifiers are resolved into virtual resources or actions related to the physical entities (as though the user "clicked on a physical hyperlink").

Physical entities are divided in three categories: people, places and things. Entities are bound to a resource that has an URL and it is accessible by the standard HTTP protocol. Two modes of web presence are considered: (1) internal support (for devices whose internal state is readable and/or settable via HTTP operations) and (2) external support (for non-electronic entities that cannot have an embedded web server). URLs for elements can be discovered using broadcast, sensed directly, or provided by another system.

Open lookup infrastructure. This work (Roduner & Langheinrich, 2007) presents an architecture for the publication and discovery of resources (information and services) associated with physical elements. It is based on the idea of physical objects having a unique identity, and different users extending them with associated services in an open manner. A lookup service to locate services that can scale to large networks such as Internet is defined.

The architecture for the lookup service is based on the following concepts: resources and their descriptions, resource repositories, a manufacturer resolver service, and search services. Resources offer information on, or services for, a physical product. Resources can be provided by the original product manufacturer or any other party.

Resource descriptions include a unique identifier, a list of the physical elements this resource is associated with (referenced by their tag identifiers), the profile they follow (an agreed syntax and semantics to which the resource adheres), the URL to the actual resource, and some context (time, location, status, etc.) and descriptive (title and description) information.

Resource descriptions are stored at the *resource repositories*. The

manufacturer resolver service (based on the Object Naming Service (ONS) defined by the EPCglobal Network) is used to find the resource repository that contains the information about an element given its tag identifier. A *search service* is also provided to locate resources based on queries. Search services crawl all registered resource repositories and create an index in a similar way as search engines do with the Web.

Event-based framework for smart identification. An event-based architecture for Auto-ID applications is defined in this work (Römer et al., 2004). Identification is based on *enter* and *leave* events. Physical elements are associated with a *virtual counterpart* (a representation of a physical element in the digital world). Virtual counterparts are classified according to which kind of element (objects or locations) are associated to, and their cardinality (a single element or a set of elements). The presented architecture stresses the relevance of locations (either geographic or symbolic), considering relationship of neighborhood (elements that are close to a location), containment and hierarchical organization. Time dimension is also considered and a query interface is defined to obtain information about the history of detections.

A virtual counterpart repository is defined to make virtual counterparts accessible. The architecture is defined in a technological-independent fashion. Different implementations of the architecture have been developed, using Jini and Web Services.

A service oriented smart items infrastructure. An architecture to support real-world objects with computing capabilities is used to decentralize business processes in this work (Spieß et al., 2007). This distributed schema is intended to increase scalability, data accuracy and response time.

The architecture is service based and it is structured in the following layers: device layer, device level service layer, business process bridging layer, system connectivity layer, and enterprise application layer.

The *device layer* comprises the actual smart item devices (sensors and Auto-ID devices) and the communication between them. The

device level service layer manages the deployable services used by the device layer. It contains a service repository that for each service stores a service description (service description provides metadata like name, identifier for the service, version, vendor, etc.) and one or more service executables (since a service may be deployable on different platforms, an atomic service may have more than one service executable).

The *business process bridging layer* has two major functions: (1) to aggregate and transform data from the devices to business-relevant information, thereby reducing the amount of data being sent to the enterprise application systems, and (2) to execute business logic for different enterprise application systems. The *system connectivity layer* provides system and data integration by routing messages and data to the correct back-end systems. Finally, the *enterprise application layer* consists of traditional enterprise IT systems responsible for controlling and managing enterprise business applications.

Defining Auto-ID architectures that are independent from the used technology supposes a conceptualization effort. In this line is worth noting the definition of data models for the data handled by Auto-ID infrastructures (specially RFID systems). This is the case of Physical Markup Language (PML) (Brock, 2001). PML is an XML-based language used for the description of physical elements including its hierarchy, classification and categorization, description and ascribed information (e.g., name, ownership or cost). Wang proposed a data model (Wang & Liu, 2005) based on the Entity-Relationship paradigm considering temporal aspects.

2.2.3 Languages for Specification

The specification of systems can be improved by using a language based on concepts that are close to their application domain. Different languages (graphical and textual) have been defined to support several of the aspects involved in the application domain this work deals with, such as the definition of pervasive services, context information or poli-

cies. A representation of such languages is provided below.

VRDK. The Visual Robot Development Kit (VRDK) (Heil et al., 2006) is a graphical tool that enables users to script their AmI environment. The user can create scripts either via drag&drop or by handwriting commands directly on the screen. Its target audience are technical interested users who do not necessarily master a general purpose programming language. A VRDK script consists of a set of processes and a set of hardware. The tool builds on the following concepts: components, events, commands, mathematical expressions, workflows and context.

The code generator transforms the script into executable code (currently C# and C are supported) and automatically deploys it on the participating devices: the application runs distributed in the environment of the user.

PervML. Pervasive Modeling Language (PervML) (Muñoz & Pelechano, 2005) is a domain specific language for the development of pervasive systems. PervML provides a a set of conceptual primitives that allow the description of the system independently of the technology. PervML covers the full development process of a pervasive system by defining a development method and providing the needed tools to support it.

PervML promotes the separation of roles where developers can be categorized as system analysts and system architects. Systems analysts capture system requirements and describe the pervasive system at a high level of abstraction using the service metaphor as the main conceptual primitive. Analysts build three graphical models: (1) The *Services Model* describes the kinds of services (by means of their interfaces, their relationships, their triggers and a State Transition Diagram for specifying the behavior of each service); (2) The *Structural Model* describes the components that are going to provide the defined services; (3) The *Interaction Model* describes how these components interact to each other.

System architects specify what devices and/or existing software systems support system services. *Binding Providers* (elements

that are responsible of binding the software system with its physical and logical environment) become the basic building blocks for PervML systems. Architects build three models: (1) The *Binding Provider Model* specifies every kind of binding provider (their interfaces and their relationships); (2) The *Component Structural Model* specifies which binding providers are used by each system component; (3) The *Functional Model* specifies which actions should be executed when a component operation is invoked.

The use of precise models to capture the requirements of a Pervasive System, allows the automatic generation of code. PervGT is a tool to support PervML method, enabling the definition of diagrams and supporting the code generation. Generated systems rely on the OSGi platform.

Context Modeling Language. In order to assist designers with the task of exploring and specifying the context requirements of a context-aware application, the Context Modeling Language (CML) (Henricksen & Indulska, 2005) is defined. CML provides a graphical notation for describing types of information (in terms of fact types), their classification (sensed, static, profiled or derived), relevant quality meta-data, and dependencies between different types of information. CML also allows fact types to be annotated to indicate whether ambiguous information is permitted (e.g., multiple alternative location readings), and whether historical information is retained. Finally, it supports a variety of constrains, both general (such as cardinality relationships) and special purpose (such as snapshot and lifetime constraints on historical fact types).

A software infrastructure is also proposed, which is organized in loosely coupled layers. The *context gathering layer* acquires context information from sensors and processes this information to bridge the gap between raw sensor output and the level of abstraction required by the context management system. The *context reception layer* provides a bi-directional mapping between the context gathering and management layers. The *context management layer* is responsible for maintaining a set of context models and their instantiations. The *adaptation layer* manages common

repositories of situation, preference and trigger definitions and evaluates these on behalf of applications.

Finally, a software engineering methodology is briefly described. This methodology is organized in the following tasks: analysis, design, implementation, infrastructure customization and testing.

Rei. *Rei* (Kagal et al., 2003) is a Policy Language for a pervasive computing environment. *Rei* is based on deontic concepts and includes constructs for rights, prohibitions, obligations and dispensations (deferred obligations). The policy language is not tied to any specific application and permits domain specific information to be added without modification. *Rei* is based on the believe that most policies can be expressed as what an entity such as users agents or services, can/cannot and should/should not do in terms of actions, services, conversations etc. *Rei* is implemented in Prolog, a logic programming language. The *Rei* policy language includes certain domain independent ontologies and accepts domain dependent ontologies. The former includes concepts for permissions, obligations, actions, speech, acts, etc. The later is a set of ontologies, shared by the entities in the system, which define domain classes (e.g., person, file, readBook) and properties associated with the classes (e.g., age, num-pages, email). *Rei* includes three types of constructs: (1) policy objects to represent rights, obligations, prohibitions and dispensations; (2) meta-policy for conflict resolution; and (3) speech acts to modify policies dynamically (delegate, revoke, cancel and request). Associated with the policy language is the policy engine that interprets and reasons about user rights and obligations from what is specified in policies.

2.2.4 Analysis and Discussion

Several conclusions arise from the analysis of the Internet of Things carried in terms of technologies, frameworks and languages.

Several **technologies** exist to support Auto-ID offering different properties. Depending on the targeted application, a particular technology should be considered. There is not a one-size-fits-all solution

in identification technologies. For industrial applications RFID provides an optimal degree of automation. However, for casual users bidimensional barcodes result attractive because of the easy way in which they can be produced and processed.

Regarding **frameworks**, the frameworks introduced provide generic capabilities for integrating physical and virtual workds but they do not provide explicit support to the specific requirements for business processes support. An analysis of frameworks that are closer to our domain of interest is provided in Chapter 3.

Regarding **languages**, there is no specific language for modeling the particular requirements of the physical-virtual linkage in terms of identification requirements. No integration with business process is provided by the considered languages. VRDK is the only language to consider a notion of workflow, however, the notation used to define this is quite basic (since end-users are the target audience) lacking the expressivity of a business process modeling language. Context description languages such as **Context modeling language** have no specific constructs for identification, being identifiers considered as sensed information.

The conclusion is that support for Auto-ID is mainly provided at technological level. Frameworks are starting to abstract the technological heterogeneity but still lacking business process integration. There is a completely lack for specification languages that can cover the physical-virtual gap for business processes.

2.3 Mobile Applications

The use of mobile devices is widespread nowadays. With the advanced capabilities of mobile devices such as connectivity, positioning systems, sensors, and advanced interaction mechanisms, new valuable services can be provided. Mobile devices are considered the first pervasively available computer and interaction device.

Mobility emphasizes several concerns (space, time, personality, society, environment, and so on) often not considered by the traditional desktop systems (Krogstie et al., 2004). In addition, many limitations in terms of computing capabilities, screen size and so on, must be con-

sidered when systems are designed for being accessed through a mobile device. Mobile information systems users are characterized by frequent changes in the context (Siau, 2003). In particular, this work is interested in *environment context* (entities that surround the user) and *task context* (what the user is doing).

Software development facilities must be provided for leveraging the great capabilities of mobile devices. The development of application for mobile devices has not been an easy task. Mobile operating systems have been mainly closed to developers. While a few have opened up to the point where they will allow some Java-based applications to run within a small environment on the phone, many do not allow this (DiMarzio, 2008). Even the systems that do allow some Java apps to run do not allow the kind of access to the "core" system that standard desktop developers are accustomed to having.

The Open Handset Alliance[1] (OHA) was formed by Google with the goal of providing the first open, complete, and free platform created specifically for mobile devices. Different organizations joined the OHA including mobile operators, semiconductor companies, handset manufacturers, and software companies. The result is the Android platform. The first version of Android was released on November 2007, almost one year later (in October 2008) the first mobile device supporting Android was released. In 2010 Android was featured in 20 devices and many more were planned including mobile phones, laptops, digital picture frames, e-book readers, home appliances, and gaming devices among others.

The availability of an open platform for mobile devices enables the development of innovative applications. Considering the number of sensors that are present in current mobile devices, developing mobile applications that integrate the physical and the virtual spaces is feasible with the Android platform. For example, Google Goggles[2] allows to search information by taking a picture of a physical element element such as a landmark, a book, a business card, or a painting. In this work, the Android platform is used for the development of physical mobile workflows.

[1] http://www.openhandsetalliance.com
[2] http://www.google.com/mobile/goggles/

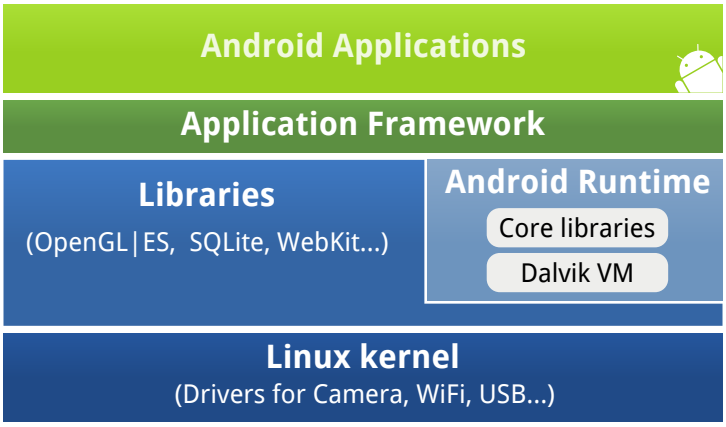


Figure 2.8: Android architecture

Next section provides more detail about this platform.

2.3.1 The Android platform

Android is an open software platform for mobile development that is intended to provide a full-stack for developers. Android includes an operating system, middleware, applications and development tools. This section describes the most innovative aspects of the Android platform since it has been used in this work.

Figure 2.8 provides an overview of the architecture of the platform. The platform is defined in different layers. Is worth noting that all the applications make use of the platform in the same way. There are not special privileges for certain applications (e.g., preinstalled applications) and any application can access to any platform service.

The Android Operating System is based on Linux 2.6. Mobile device manufacturers are in charge of developing drivers for their devices that follow the Linux directives. On top of the operating system, basic system functionality is provided by native C/C++ libraries. Most of them provide low-level functionality based on well established open source projects such as OpenGL|ES, FreeType, SQLite or WebKit. This layer abstracts the particularities of each hardware piece. For example,

OpenGL can be used to program graphics regardless of the technology used for the device screen.

One of the native components that plays a central role for the development of Android applications is the Dalvik Virtual Machine. Dalvik allows the development of applications for Android in Java. The Dalvik virtual machine allows the system functionality to be accessed from a higher abstraction layer. The Dalvik can run classes compiled by a Java language compiler that have been transformed into Dalvik Executable (*.dex) format, a format that is optimized for efficient storage and memory-mappable execution.

On top of the virtual machine Java utility functions based on Apache Harmony[3] are provided. These libraries provide interfaces and classes for programming. Data structures such as collections, connectivity functions to handle sockets and input and output access are only some of the functionalities that are provided with these core libraries.

In order to facilitate the development an application framework is also provided. This framework provides the basic building blocks, communication mechanisms and APIs that any Android application will use. From the software engineering perspective, the application framework is the most relevant layer in the Android platform since it defined the main components that form any Android-based application. The main components that form the framework are detailed below.

The Android Application framework

The Android application framework is based on loosely-coupled components. Each component developed is declared in the Android Manifest. When a component is described in the Android Manifest, it defines the way it is integrated in the platform by indicating the possibilities for communicating with other components and the permissions that each application requires form the platform. In this way, when an application is installed by users, they can know which kind of use of their mobile device would make a certain application.

The Android application framework provides the following main

[3] http://harmony.apache.org/

components: *Activity*, *Service*, *Content Provider* and *Broadcast Receiver*. These components are introduced below.

Activity. An Activity presents a visual user interface to the user. An activity is designed around a well-defined purpose (e.g., viewing, editing, dialing the phone, taking a photo, etc.). It handles a particular type of content (e.g., a list of contacts) and accepts a set of actions. The user interface provided by the Activity is composed by a hierarchy of interaction nodes (*View* and *ViewGroup* elements following the composite pattern) that process input events. Each activity has a lifecycle that is independent of the other activities in its application or task and it is managed by the application framework.

Service. A Service provides functionality that is executed in the background (e.g., a service that plays music). It is possible to connect to an ongoing service and start it if it is not already running. While connected, communication with the service is performed through an interface that the service exposes. Different components such as Activities or other Services can be binded to a Service.

Content Provider. A content provider makes data available to other applications. The data can be internally stored in the file system, in an SQLite database, or in any other particular mechanism. A Content Provider exposes generic mechanisms for accessing the information regardless of the underlying implementation technology used.

Broadcast Receiver. A broadcast receiver is a component that reacts to announcements from other components. Broadcasts can originate from system code (e.g., indicate that the battery is low) or other applications. In response to the broadcast, Broadcast Receivers can start an activity or use the *NotificationManager* to alert the user. Notifications can get the user's attention in various ways (flashing the backlight, vibrating the device, playing a sound, etc.).

Android allows different components to execute simultaneously and it provides an inter-component communication mechanism based on Intents. An *Intent* is an abstract description of a desired action (e.g., obtaining an image) regardless of the component that provides this functionality. Components declare in their manifest which kinds of intents they can respond to, and the linkage between the caller and the callee is performed dynamically at run-time. This enables the extensibility of the platform since newly installed applications can take advantage of already installed components.

2.4 Conclusions

This chapter provides an overview of different techniques and tools that are related with the work presented in this document. The analysis has considered three application domains: Business Process Management, the Internet of Things and Mobile Applications. This work is aimed at providing development support for systems that fit in these three areas. Thus, much of the technologies and techniques introduced are applied in the following chapters. A more detailed analysis of the proposals that are more close to the goal of this work is provided in Chapter 3.

Chapter 3

State of the art

This chapter introduces different approaches that support the development of physical mobile workflows. Once we have analyzed in Chaptr 2 the general application domains in which physical mobile workflows fit, we analyze the specific proposals in this domains that are closely related to our approach. this analisys allows us to determine the way in which each proposal addresses the aspects that are central in our approach.

In order to classify the different approaches and determine to which extent they can support physical mobile workflows, we have characterized the systems of interest for this work. The necessary properties that a system must fulfill to be considered a physical mobile workflow are detailed below.

A well-defined business process. Most of the research in Ubiquitous Computing has been focused on supporting the informal and unstructured activities that are typical of much of our everyday lives (Abowd & Mynatt, 2000). However, the application domain targeted in this work is different, since it is focused on well-defined and structured activities. Applications of this domain are char-

acterized by a clear definition of the activities involved and their coordination. For example, the loan process of a library or the production method in a factory are composed by a clear sequence of steps that must be followed.

Explicit identification. This work deals with business processes in which physical elements are involved. For the integration of these elements, Auto-ID technologies are used. The Auto-ID approach involves to artificially augment physical elements with a digital identity (e.g., a barcode, a visual marker, an RFID tag, etc). This requires an effort in labeling physical elements with the purpose of simplifying the identification of the physical context. The use of explicit identification of the physical context results in a simpler approach in contrast to processing the environment "as is". Whether the labeling effort is acceptable or not depends on the particular application.

User participation. Although Auto-ID enables process automation in many cases, a complete automation of business processes is not always possible or desirable (Tedre, 2008). This work considers the user participation as central in the workflow. In particular, we are interested on the interaction between the user and their physical environment by means of a mobile device, and how this is adapted according to the business context to provide different levels of automation. The use of a mobile device allows the user to access the information and services directly from the domain objects involved in the process.

The above properties make the application of modeling techniques appropriate for the development of physical mobile workflows. On the one hand, business processes can be easily described by workflow notations. In contrast to most of the user processes, the sequence of steps that must be followed to support a business goal of a company or organization is simpler that the description of the personal motivations in long-lived, everyday human activities (Li & Landay, 2008) such as staying fit. On the other hand, Auto-ID technologies share a set of basic concepts (Kindberg et al., 2002) that can be used to characterize the physical-virtual linkage regardless of the particular technology used.

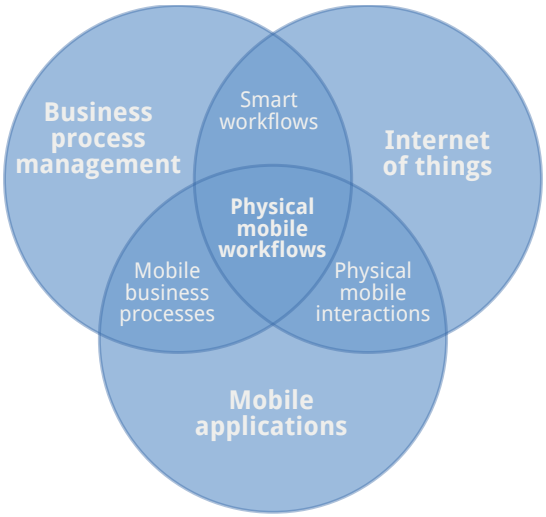


Figure 3.1: Application domains in this work and their intersecting subdomains

Physical mobile workflows are a specific kind of systems that can be considered in the intersection of *Business Process Management* (BPM), the *Internet of Things* (IoT) and *Mobile Applications* domains. An overview of the technologies and techniques used in these areas was presented in Chapter 2. Figure 3.1 illustrates different research areas that are relevant to the present work and their intersections. In particular, we considered approaches that deal with systems that fulfill most of the features defined above for physical mobile workflow characterization. We identified three research areas where physical mobile workflows fit: *smart workflows*, *physical mobile interactions* and *mobile business processes*. Relevant approaches in these areas have been analyzed and discussed in this chapter.

The remainder of the chapter is structured as follows. Section 3.1 presents related work in the *smart workflow* area. Section 3.2 introduces the research carried out in the *physical mobile interaction* area. Section 3.3 studies different approaches in the *mobile business process* area. Finally, Section 3.4 concludes the chapter.

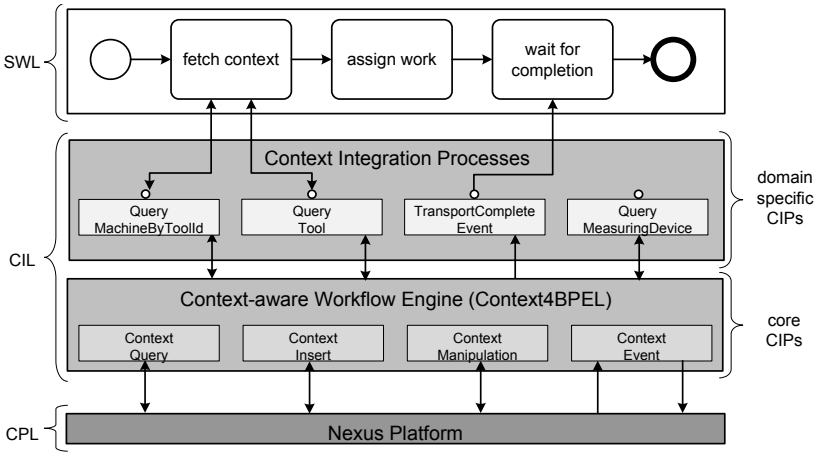


Figure 3.2: Architecture proposed by Wieland for the support of smart workflows (Wieland et al., 2008)

3.1 Smart workflows

Smart Workflows (Wieland et al., 2008) are business processes that cross the boundary between the digital and the physical worlds. Physical mobile workflows can be defined as a specific kind of smart workflows that are accessed by means of a mobile device. Relevant proposals from the smart workflow research area are introduced below.

Context-aware workflows. Wieland defines the Smart Workflow (Wieland et al., 2008) concept and provides an architecture for transforming the low-level data that is captured by different kinds of sensors (not only those based on Auto-ID) into information at the business level. Wieland proposes a three layered architecture model which relies on the Nexus platform[1].

Figure 3.2 shows the architecture for a system supporting a production process in a factory. The context provisioning layer (CPL) is responsible for managing the context information and for providing that information using pull-based and push-based communication. Nexus platform is used for context provisioning. The

[1] http://www.nexus.uni-stuttgart.de/

context integration layer (CIL) uses the generic interface provided by the CPL to integrate information into higher-level representations by filtering and aggregating information. Context Integration Processes (CIPs) together with a context-aware workflow engine is used to implement this layer. The CIL provides context information at different semantic levels for smart workflows and other context aware applications. Finally, the smart workflow layer (SWL) realizes the smart workflows. To support smart workflow definition, Context4BPEL is defined. Context4BPEL is a modeling language for context-aware workflows. The primitives introduced allow the workflow to (1) subscribe to certain context events, (2) query context information, and (3) route the control flow according to context conditions.

PerFlows. PerFlows (Urbanski et al., 2009) are defined as personal workflows. The PerFlow system is aimed at addressing the interoperability between devices and applications for achieving the user's personal goals. PerFlows are defined by end-users to model their everyday tasks. A PerFlow allows specifying conditions based on context information such as location, that cause tasks to start, tasks to skip, and tasks to abort. Figure 3.3 shows the PerFlow modeler tool which is used to define workflows.

In order to support PerFlows, context information is integrated in the workflow. This approach makes also use of the Nexus platform to handle context information but also integrates a location system based on GPS. PerFlow provides a distributed workflow system with dynamic adaptable processes which is more focused to support daily activities which are not easy to completely formalize. Users participate in the workflow by means of their mobile devices, and different levels are considered for workflow automation. In the process definition, tasks are classified as reversible or not reversible. For tasks that are considered reversible, the system acts in a proactive manner: the system automates the action since users can go back to the previous state. In addition, PerFlow allows to define conditions for skipping tasks to make the process more fluent if a given condition is met.

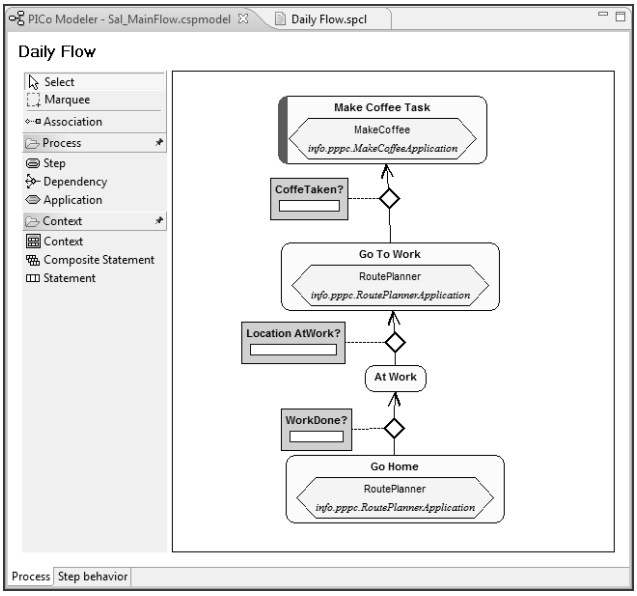


Figure 3.3: Personal workflow (PerFlow) editor (Urbanski et al., 2009)

Decoflows. *Taskable spaces* are defined as places fitted with a task computing framework (Loke, 2009). A task computing framework is in charge of bridging the gap between user tasks (what the user wants to accomplish) and system services (environmental capabilities to complete the tasks). Based on this idea, Loke defines mechanisms for nontechnical users to specify service composition to fulfill their tasks (Loke, 2003). Workflow definitions are based on Eco scripts. Eco is a textual language with semantics based on π-calculus and can be translated into WS-BPEL.

Eco language allows to define service orchestration for multi-agent environments that can interact with the environment that is easy to understand and provides execution support. An example expression in Eco is provided below:

```
make coffee; dim lights; wait for lights; download news; wait
 for news; show news on tv;
```

The system executes ECO task statements concurrently, subject

to explicit prerequisites; the operator ";" indicates a parallel composition. Each task statement sends a command to a particular agent (e.g., *coffee* refers to the brewing agent in the coffee machine) which processes one command at a time.

SAP's Auto-ID infrastructure. The Auto-ID Infrastructure (Bornhövd et al., 2004) defined by SAP, has as a main goal to convert RFID or sensor data into business process information by associating it with specified mapping rules and metadata. The infrastructure is tailored to meet the requirements of RFID and it makes use of the standard services that EPCGlobal defined for dealing with RFID detection events. The architecture defines four layers: *device layer*, *device operation layer*, *business process bridging layer* and *enterprise application layer*.

At the *device layer* a hardware-independent low-level interface is defined to integrate different kind of identification devices. It consists of the basic operations for reading and writing data and a publish/subscribe interface to report observation events. The *device operation layer* coordinates multiple devices. It provides functionality to filter, condense, aggregate, and adjust received sensor data. The *business process bridging layer* associates incoming messages with existing business processes. At this layer information of tracked objects is maintained such as object location, aggregation information, and information about the environment of a tagged object. Finally, the *enterprise application layer* supports business processes of enterprise applications such as Supply Chain Management (SCM), Customer Relationship Management (CRM), or Asset Management running on SAP or non-SAP backend systems.

3.1.1 Analysis and Discussion

The different approaches in the smart workflow area are mainly focused on supporting business process execution. These proposals normally observe events form the real world and provide automatic service orchestration accordingly. Conversely, our approach allows to better de-

Property	Context-aware workflows	Perflows	SAP	De-coflows
Workflow support	yes	yes	yes	yes
Real-world interaction	context	context	RFID	service invocation
User participation	–	yes	–	–
Automation levels	automated	skip/undo	automated	automated
Modeling technique	WS-BPEL	graphical	–	textual
Method	–	–	–	–

Table 3.1: Related work from the smart workflow perspective

tail the way in which the physical-virtual linkage is established and the degree to which users participate in the workflow.

Table 3.1 shows a comparison of the presented approaches. The table shows to which extent the different approaches cope with some of the most relevant aspects of our proposal. All the introduced approaches have the goal of supporting business processes (see *workflow support* row). Nevertheless, different techniques are used for modeling (see *modeling technique* row). These techniques include proprietary notations (either graphical or textual) and integration with WS-BPEL. The modeling languages used are focused on business process execution, providing low-level abstractions for business process modeling (compared to BPMN, for example).

The presented approaches do not provide facilities for user participation (see *user participation* row) in the processes with the exception of PerFlow. Our approach provides mechanisms for an effective participation of users in workflows. PerFlow allows users to participate in a workflow with their mobile devices. However, in the case of PerFlows the automation level cannot be adjusted in a flexible manner. PerFlow

provides only two mechanisms to vary the automation degree of the process (some tasks can be skipped under certain conditions, and reversible user tasks can be performed in advance by the system). Our approach provides richer mechanisms for regulating the process automation in different levels and the user participation in the process. The rest of the proposals are mainly focused on the support for automated processes (see *automation levels* row).

These proposals consider the interaction with the real world in a different way (see *real-world interaction* row). Context-aware workflows and PerFlows react in response to context changes, SAP'a Auto-ID Infrastructure provides Auto-ID capabilities by means of RFID and Decoflows are mainly concerned in device communication to request and invoke distributed functionality. Our approach focuses on Auto-ID but it is not biased towards any particular technology.

Since the presented approaches are mainly concerned with execution aspects, they lack methodological guidance (see *method* row), and only Perflow provides tool support for the easy edition of workflow definitions.

3.2 Physical Mobile Interactions

Physical mobile interactions (Rukzio et al., 2006) are mobile interactions between a user, a mobile device, and a smart object in the real world. In this approach the user interacts with digitally-augmented objects through the mobile device as he/she interacts with the mobile device and the mobile device interacts with the object. Physical mobile workflows can be considered a specific kind of system based on physical mobile interactions that supports a well-defined business process. The approaches studied from the physical mobile interaction area are detailed below.

Physical Mobile Interaction Framework. The Physical Mobile Interaction Framework (PMIF) was defined to support rapid development of physical mobile applications (Rukzio et al., 2005b). PMIF provides an architecture that is based on a stream metaphor. That is, the application sees the connection between the mobile

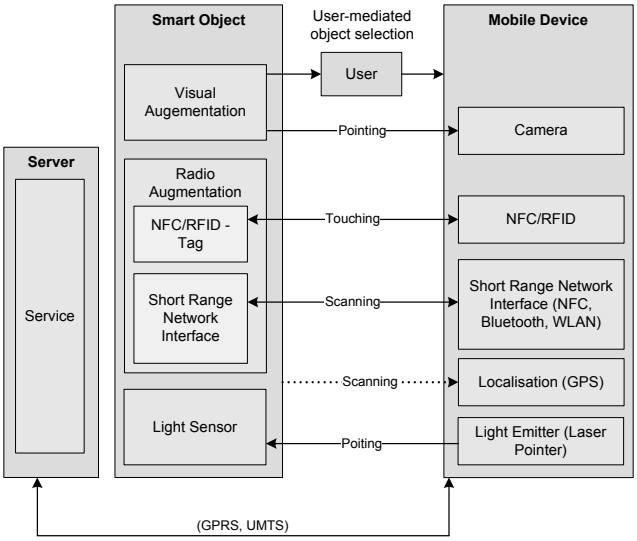


Figure 3.4: Generic architecture of the Physical Mobile Interaction Framework (Rukzio, 2007)

device and the physical object as a *stream* on which the application can read or write.

Figure 3.4 shows all elements involved in the architecture of PMIF: the mobile device, the smart object, and related services running on a server. The mobile device acts as a mediator between the physical and the digital world. The server represents the digital world which offers information and services related to the smart object. The latter represents the physical world and provides entry points into the digital world (the smart object provides a link to corresponding services that are made available by a server).

The communication between the mobile device and the smart object can be based on different modalities as it is illustrated at Fig. 3.4: information provided by the smart object can be sensed by the mobile device (unidirectional arrow from smart object to mobile device), the mobile device can submit information to the smart object which senses it (unidirectional arrow from the mobile device to the smart object) or there can be a bidirectional

communication between the mobile device and the smart object (bidirectional arrow between mobile device and smart object).

Rukzio et al. compare different physical mobile interaction techniques (Rukzio et al., 2006) and provide guidelines for their selection during design. These techniques are *touching*, *pointing*, *scanning*, and *user-mediated interaction*. PMIF provides implementations of the interaction techniques touching using NFC or RFID, pointing using visual markers or a laser pointer, scanning using Bluetooth or GPS and user mediated object selection. In addition, new interaction techniques can be added to the architecture in a pluggable manner.

Context-sensitive User Interface Profile. Berg and Coninx introduced the Context-sensitive User Interface Profile (CUP) (den Bergh & Coninx, 2005). CUP is a UML-based notation that allows the specification of requirements for context integration into an interactive application. CUP proposes some improvements on UMLi (da Silva & Paton, 2003) to consider context aspects. CUP takes some notions from the Context Modeling language (Henricksen & Indulska, 2005) and defines a UML-based modeling language to define interactions based on them.

CUP is defined by means of the UML profile extension mechanism. the following models are defined: The *application model* shows the concepts and the relations between them that are used within the application. The *task dialog model* provides an hierarchical view to the activities that need to be accomplished. The *context model* shows the concepts that can influence the interaction of the user with the application directly or indirectly. The *abstract presentation model* shows the composition of interactors in the user interface and describes the general properties of the interactors (the data they interact with and meta information about them). The *concrete presentation model* describes the user interface for a specific set of contexts or platforms.

The context model defined considers the nature of gathering context information (manually entered into the system by the user/designer of the software or automatically sensed/interpreted), the

information about the platform through which the user interacts with, context information ambiguity, and the topic of interest.

The Physical User Interface Profile. The Physical User Interface Profile (PUIP) (Rukzio et al., 2005a) supports the design of different aspects of a physical mobile interaction. PUIP extends CUP and it is also based on UML. PUIP supports the integration of aspects like the type of information presented at some point and how the real world context changes during an interaction.

PUIP is defined to allow designers to classify, compare and evaluate new physical mobile interactions. PUIP is intended to support all the phases of the development process with a main role in the communication among stakeholders.

The profile is defined to capture some of the aspects that characterize physical mobile interactions. For the user interface description PUIP considers the physical constraints of mobile interactions with the real world (e.g., the user must be in a specific distance to a physical object before starting the interaction). Device features (e.g., different screen sizes) are considered to support different modalities of interaction since most existing physical mobile interactions are multimodal. Finally, the temporal context (e.g., duration of interaction) and the social context (e.g., presence of multiple users in front of a display) are also considered.

Figure 3.5 shows a task model based on PUIP. Since it is based on CUP, PUIP uses UML 2.0 activity diagrams to model tasks based on the concepts used in ConcurTaskTrees (Mori et al., 2002). PUIP further details the semantics of CUP stereotypes. For example in Fig. 3.5 the user interface element *StartFocusAndBrowse* is annotated with property values showing that the element is part of the group component *InitialDisplay* and rendered by the device *MobileDevice*.

3.2.1 Analysis and Discussion

Different kinds of interactions between mobile devices and the environment have been studied (Iftode et al., 2004). The previous chapter

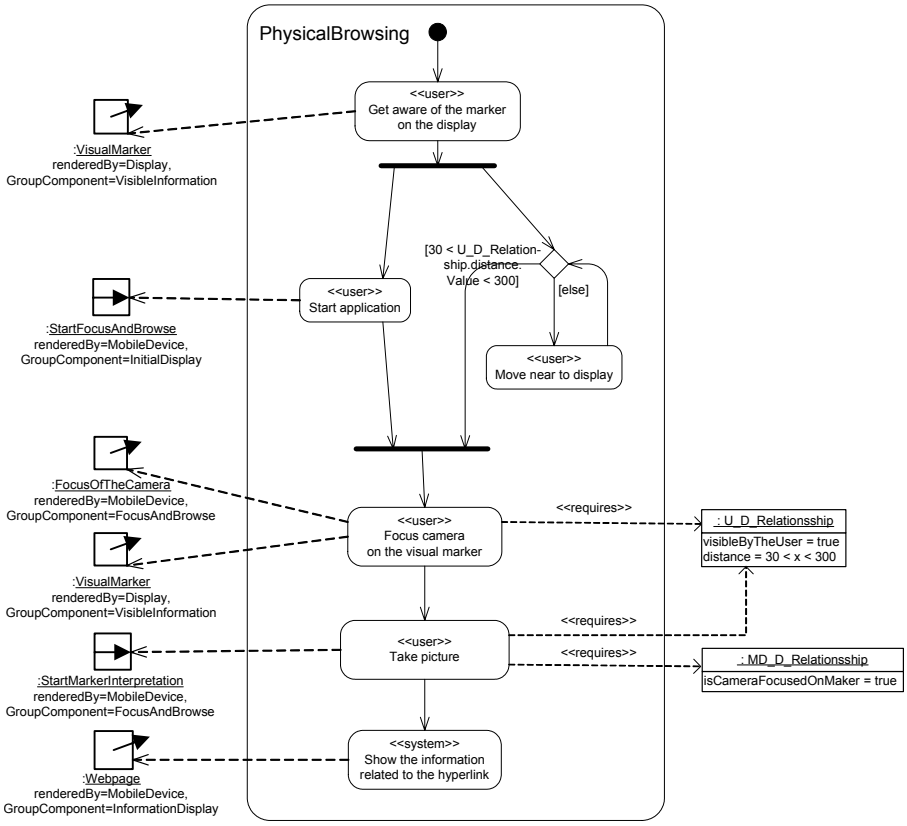


Figure 3.5: Dialog model used by PUIP (Rukzio et al., 2005a)

detailed in Section 2.2 the technologies, infrastructures and languages to bridge the gap between the physical and the digital world. The approaches introduced in this section organize these interaction concepts and technologies in methods that allow the development of systems supporting more complex interactions.

Table 3.2 shows a comparison of the presented approaches. All these approaches define methods (see *method* row) for specifying interactions between the user (see *user participation* row) and the environment. For specifying these interactions UML-profiles are used (see *modeling tech-*

Property	PMIF	CUP	PUIP
Workflow support	no	low-level	low-level
Real-world interaction	pluggable	yes	yes
User participation	yes	yes	yes
Automation levels	guidelines	low-level	low-level
Modeling technique	UML	UML	UML
Method	yes	yes	yes

Table 3.2: Related work from the physical mobile interaction perspective

nique row). Although these proposals are not intended to describe workflows (see *workflow support* row), some of them (CUP and PUIP) make use of UML activity diagrams to model the coordination of the different services in each interaction technique described.

These techniques can be used in conjunction with our approach if there is a need for describing the internals of an interaction technique. When an interaction technique is not natively supported by a platform, a description of the interaction and the computing elements involved could be useful for developers in order to guide its implementation in a particular device. However, our approach is not focused on describing interaction techniques but describing how these techniques can be combined to support a business process. Thus, the interaction techniques described with the introduced approaches are used as building blocks in our approach to support workflows.

Since these approaches describe the internals of physical interactions, the automation level offered (see *automation levels* row) depends on the components used at modeling, but there is no explicit notion of the automation level. PMIF allows different Auto-ID technologies to be incorporated in a pluggable manner which provides more flexibility in defining the automation level at architecture level (see *real-world in-*

teraction row), but mechanisms for defining it explicitly or adapting it to context conditions are lacking. Conversely, our approach takes into account the user attention as a resource that must be considered when deciding which interaction technique to use.

3.3 Mobile business processes

Mobility allows users to access digital services on the go. The use of mobility to support business processes is an essential aspect of physical mobile workflows since it allows users to interact with business process services closer to the place they are needed. Physical mobile workflows can be defined as a specific kind of mobile business processes where physical elements are integrated in the process by means of Auto-ID. The approaches studied from the mobile business process area are detailed below.

PerCollab. PerCollab (Chakraborty & Lei, 2004) uses an extended version of WS-BPEL (xBPEL) to formally define business processes with human partners and exploits dynamic user context to address mobility issues. PerCollab seeks to address the respective limitations in workflow systems (lack of support for direct human-to-human communication) and collaboration tools (lack of orchestration) by effectively integrating the two. Specifically, PerCollab allows people to participate in business processes using any traditional communication mechanism.

PerCollab provides mechanisms to dynamically select an appropriate device or modality to engage the user for a particular interaction. The choice of an appropriate device may change in the course of a business process. The xBPEL language allows to define new types of partner in WS-BPEL processes: human partner (a human participant in the process), process-to-people (communication between the human partners and the WS-BPEL process) and people-to-people (direct communication between the human participants).

The xBPEL definition is translated into WS-BPEL and it is de-

ployed into a business process execution engine that interacts with different PerCollab components that are exposed as web services. PerCollab depending on the location activity and preferences, selects a communication mechanism to fulfill a given task. The components of the PerCollab architecture include the *Interaction Controller* that acts as a proxy to represent all human participants, the *Context Service* that gathers and processes context information, and several *Modality Adapters* to transform messages to a specific format.

Mobile Process Landscaping. Mobile Process Landscaping (PML) (Köhler & Gruhn, 2004) is a method to decompose business processes in different levels of detail, in order to identify mobile sub-processes. The idea of the method is to split the modeling of processes into different tiers, starting with a coarse and simplified form of the process description and then increasing the level of detail with each tier. The method can be applied to different modeling notations and it is useful for determining portions that can be distributed.

A business process is considered mobile according to the MPL criteria if there is an "uncertainty of location" that is caused by external factors (i.e., the process-executing person has no freedom of choice regarding the place of the process execution), and a cooperation with external resources (from the process-point of view) is needed in the execution of the process.

Based on the previous definition, MPL proposes the following steps: (1) analysis of the process model and identification of mobile business processes, (2) Redesign of the identified process partitions, (3) specification of the mobile part as required by the new business processes, (4) validation of the profitability of the change, and (5) implementation of the change.

MPL method is useful for detecting the different parts of a "traditional" business process that can be distributed. A mobile business process version of a process is obtained by analyzing the activities at different levels. The modeling layers considered are *company structure and organization*, *functions*, *activities*, and the *dialog*

and information flow. A systematic approach is provided for analyzing business processes at the different levels.

Sliver. Sliver (Hackmann et al., 2006) is an execution engine for mobile devices that supports SOAP and WS-BPEL. Sliver is designed to overcome the limitations of WS-BPEL when it is used for supporting the dynamic nature of ubiquitous services. In particular, Sliver deals with the limited computing resources of mobile devices and the connectivity problems. Sliver is designed to (1) have a suitably small storage and memory footprint; (2) depend only on the Java APIs that are available on mobile devices; and (3) support a wide variety of communication media and protocols flexibly.

Sliver supports the core features of WS-BPEL and has a total code base of 114 KB. Sliver supports all of the basic and structured activity constructs with the exception of the compensate activity, and supports basic data queries and transformations expressed using the XPath language. Sliver also supports the use of WS-BPEL Scopes and allows for local variables and fault handlers to be defined within them. Sliver performs different optimizations (e.g., avoid message schema validation) to fit mobile limitations at run-time. In order to provide flexibility, Sliver enables the dynamic change of partner links during WS-BPEL execution. In order to be transport-agnostic, the WS-BPEL specifications do not define how to assign actual communication endpoints to each partner link. Thus, Sliver maps the names of incoming partner links to the kinds of messages that they accept as input. then, it maps the names of outgoing partner links to concrete endpoints.

Mobile Workflow Engine. As opposed to an isolated and standalone application environment, this workflow engine (Pajunen & Chande, 2007) is presented as a composition platform which can orchestrate local applications and system services.

In order to support this approach, different workflow-related standards are extended. Support is provided (1) to query local deployment information (e.g., phone numbers) by means of XPath, (2)

better handle attachments during WS-BPEL execution, and (3) support mobile specific protocols (e.g., SMS, MMS and Bluetoth).

The mobile workflow engine supports integration to local applications from the mobile device (e.g., browser, map, calendar, and messaging). In order to enable SOAP and WSDL based interfaces for these applications; the mobile workflow engine includes wrappers on them. These wrappers transform coming SOAP messages to function calls and invoked call back calls to out-going SOAP messages.

A model-driven approach was considered for the development of user interfaces for processes based on the workflow engine (Ruokonen et al., 2008). This approach provides mechanisms for extending mobile business process models with UI models. Model-driven techniques are applied in order to obtain customized UIs to support user interaction with heterogeneous devices. For example, if camera application is not supported, a textual description can be required instead (i.e., a text field is provided).

Requirements of UI customization are captured by the *plaftorm model* and the *presentation model* with a role similar to those defined in the reference framework for multi-target user interfaces (Calvary et al., 2003). The final result for each process is one WS-BPEL file and several (X)HTML files to present all the required UI views and WSDL files. To support customization of UIs, in addition to process description, platform customization rules are provided.

3.3.1 Analysis and Discussion

Research in the mobile business processes area has been mainly focused on supporting service orchestration in a constrained environment as it is the mobile device. However, the capabilities for linking the physical and digital spaces in a way that fits the business process requirements has not been explored. Our approach analyzes the general requirements of business processes and provides mechanisms to define the physical-virtual linkage accordingly.

Property	PerCol-lab	MPL	Sliver	MWE
Workflow support	execution	modeling	yes	yes
Real-world interaction	–	–	–	–
User participation	yes	yes	yes	yes
Automation levels	modality selection	–	multiple partners	resource adaptation
Modeling technique	WS-BPEL	proprietary	WS-BPEL	WS-BPEL + UML
Method	–	yes	–	yes

Table 3.3: Related work from the mobile business process perspective

Table 3.3 shows compared results of the different proposals presented in this section. All the approaches presented provide support to business process (see *workflow support* row). Nevertheless, PerCollab is more focused on providing execution support to mobile workflows while Mobile Process Landscaping (MPL) supports the analysis phase. The rest of the approaches provide both mechanisms to model and execute mobile workflows.

None of the approaches provides mechanisms to describe how physical elements are integrated in the workflow (see *real-world interaction* row). Thus, these approaches should be complemented if there is a need for effectively linking the physical and the digital worlds. However, all the approaches analyzes consider the user as a central element in both analysis and execution of the workfow (see *user participation* row).

These proposals offer different mechanisms for adapting the workflow to the user needs. Allow participants to appear dynamically, provide multiple modalities and adapting to the available resources are the techniques used. By means of these techniques the automation of the process can be modified (see *automation levels* row). However, the adaptation provided by these proposals has the goal of avoiding techni-

cal aspects such as the screen size. Our approach addresses a different issue that is more related to human limitations of the user (e.g., attention) than technical limitations of the device (e.g., screen size).

Modeling techniques are applied by all the different approaches (see *modeling technique* row). WS-BPEL is used for describing process execution aspects. MWE complements WS-BPEL descriptions with UML Activity diagrams. Activity Diagrams are used to describe the tasks at the user interaction level. MPL and MWE define a sequence of steps to guide the system development according to their approach (see *method* row).

3.4 Conclusions

This chapter presents the state of the art in the disciplines that are related to this work. These areas are really active these days with many emerging initiatives. However, there is still a lack of proposals to provide mechanisms that allow the development of physical mobile workflows from a high level of abstraction. Modeling languages are used only to describe execution aspects providing minimal support to analysis, which avoids the application of the re-engeneering process defined by BPM.

For effectively capture workflow requirements in the IoT, other aspects than service orchestration must be considered. The present work, defines support for the analysis, simulation and execution by supporting the specific requirements of physical mobile workflows. Since many approaches already address them, our approach is not focused in defining service orchestration but describing the ways in which physical elements are involved in the process and the way users interact with them in this context.

Chapter 4

A design method for physical mobile workflows

The use of Auto-ID technologies for the support of business processes provides new opportunities for process automation (Strassner & Schoch, 2002). Depending on the use of a specific technology (RFID, barcodes, smart cards, or traditional keyboards and mice), a different degree of automation is possible. Furthermore, the automation level demanded by users varies from task to task. Sometimes users want to be aware of the system actions, however, in other situations users want the system to act silently without disturbing them.

According to Tedre (Tedre, 2008), a process should be carefully studied before determining what to automate and to what degree it should be automated. The need for studying what to automate becomes more relevant when business processes take place in a pervasive environment where many options for automation exist. For physical mobile workflows, in contrast to desktop-based systems, the design of the automation level becomes a must.

This chapter introduces a methodological approach for the design of physical mobile workflows. The goal of this method is to provide a

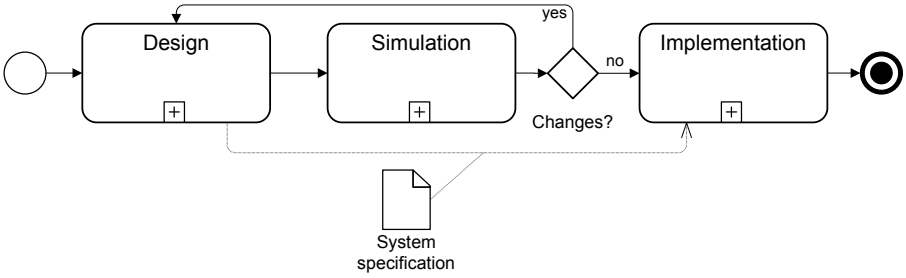


Figure 4.1: The stages proposed in the development of physical mobile workflows

mechanism for **defining the desired degree of automation for the physical-virtual linkage of a given business process**. In order to systematize the development of such systems, the method is based on the Business Process Management (BPM) initiative principles.

BPM is an initiative that promotes the continuous re-engineering of business processes. BPM conceives software development as an improvement cycle where processes are continuously analyzed, modeled, simulated, executed and monitored. Since current solutions for BPM are mainly focused on the digital world (i.e., service orchestration), support is lacking for coping with the particularities of the physical-virtual linkage in the different stages of the BPM cycle. This work builds onto existing BPM techniques and extends them to integrate business processes with the physical world at different levels. The method provides support to *model*, *simulate* and *execute* physical mobile workflows. Existing BPM techniques are complemented with support for capturing the identification requirements, evaluating the user participation in a real environment, and executing the workflow in a software platform.

Our approach is focused on the first phases of the BPM cycle: business process design, simulation and execution. In particular we are interested in providing mechanisms to project these techniques to the physical world. Other BPM phases such as the Business Activity Monitoring (BAM) are not considered in our approach since they take place mainly at the digital world and existing tools can be used for this purpose.

Figure 4.1 shows an overview of the stages in the development process proposed in our approach. The physical mobile workflow is iteratively designed. In each design cycle, the workflow is put into practice to obtain feedback from the users. The feedback obtained is used to improve the original design. When no further changes are required, the system specification is used to guide the implementation of the final system. This chapter provides detail on the stages involved in the iterative design of physical mobile workflows. It introduces the aspects to be considered during design and simulation. Chapter 5 provides detail on the implementation of physical mobile workflows. Finally, Chapter 6 covers the specific case of workflows that require to be modified at run-time instead of following the redesign cycle.

The remainder of the chapter is structured in the following manner. Section 4.1 provides an overview of the design method and how it is integrated in our approach. Section 4.2 introduces the concepts used to describe the physical-virtual linkage in an abstract manner. Section 4.3 introduces mechanisms to describe the capabilities of the Auto-ID technologies available. Section 4.4 provides guidelines to validate in practice the designs obtained with the method. Section 4.5 concludes the chapter.

4.1 Design method overview

This section provides more detail on the development method introduced in this work. The design stage is the initial stage in our method (see Fig. 4.1). Since we are following a model-driven approach, the specification obtained at design drives the later stages in the development of the system. Thus, the design stage becomes central to the development method.

The design method captures by means of models the concepts that are relevant in the development of physical mobile workflows. The benefits of using a model-driven approach and the steps to be followed in the design of physical mobile workflows are introduced below.

4.1.1 Why a modeling approach?

Traditionally, the application of Auto-ID to business processes has mainly been approached from a technological perspective (by developing integration middleware and architectural designs). However, deploying an Auto-ID-enabled system involves a lot more than purchasing the right tags and installing the right readers (Sarma, 2004).

The way in which a business goal is achieved depends on the properties of the physical-virtual integration. Certain business models are only feasible with an adequate level of automation in the physical-virtual linkage (Fano & Gershman, 2002). For example, using RFID for identifying products in a supermarket allows checkout to be automated (Roussos et al., 2002), and does not require the participation of a cashier in the process. Thus, when modeling a business process it is not possible to determine which tasks are required for handling physical elements (e.g., requiring a cashier to handle them or not) if there is no notion of how they participate in the process. Models are key in our proposal to provide this notion by linking identification requirements to technological requirements in a gradual manner.

Abstraction is one of the fundamental principles of software engineering in order to master complexity (Kramer, 2007). Our approach makes use of modeling techniques in order to promote abstraction in the development of physical mobile workflows. By abstracting technical details, we can describe the physical-virtual connection of a workflow regardless of the particular technology used for the implementation. In the case of physical mobile workflows, modeling techniques are applied to obtain the following benefits:

Focus on the process. Separation of concerns is promoted by our approach in order to allow designers to focus on a specific aspect of the workflow at a time. Business analysts can define the way in which physical elements participate in a business process without thinking on technology limitations. They can think on the way they want the process tasks to flow, and later, the appropriate identification mechanisms can be chosen to cope with their requirements.

Explore the solution space. The use of models allows to capture not only a specific solution but also the rationale behind it. In this way, alternative solutions can be re-considered and the design knowledge can be better reused for similar problems. In addition, support for traceability allows to easily identify the model elements affected when different issues are detected during the system evaluation.

Support system evolution. The fast changing nature of business processes and the technological heterogeneity of identification technologies suggests that systems in this area must be designed to evolve. By analyzing the knowledge captured in models by our approach, it can be easily determined whether or not a new technology fits better with the requirements of the physical-virtual linkage for a given process.

4.1.2 Steps of the method

The development method proposed in this work supports the design, simulation and execution of physical mobile workflows. The method defined involves three development roles: *Business Analysts*, *Designers* and *Developers*. Each one makes decisions at different abstraction levels. Following BPM principles, we propose modeling a workflow before a software system is developed for it. In the design stage, the aspects that are considered relevant for the final system are captured and they are used to guide the implementation. Figure 4.2 details the tasks involved during the design of physical mobile workflows. The steps performed during design and how the obtained design provides feedback to the simulation and execution phases is detailed below:

1. **Business process specification.** *Business Analysts* are in charge of modeling the business process in a technological-independent fashion. Business Process Modeling Notation (BPMN) (OMG, 2006) is used to capture the **user activities** involved in the process and their temporal dependencies in a business process diagram. In addition, the analysts must **detect the physical elements** that are involved in each task and specify their properties.

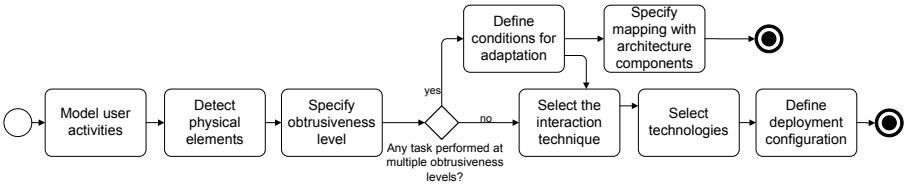


Figure 4.2: The different tasks in the desing method proposed

Once tasks and physical elements are identified, the requirements for physical interaction are captured. Since we want business process descriptions to be high-level representations of process requirements, technological detail regarding the particular mechanism used for identifying physical elements is avoided. In order to determine the degree of automation for each task in the process, the **obtrusiveness level required by each task is specified**. By considering the desired obtrusiveness level, an adequate **interaction technique is selected** for each physical element that participates in a task. The above aspects constitute physical mobile workflow requirements that are independent of any particular technology. More detail regarding the specification of these aspects is provided in Section 4.2.

In the case that a task could be performed at different obtrusiveness levels, designers must **specify the conditions** for each level to be used at run-time and **the effect of the adaptation in the architecture**. The run-time adaptation aspects are covered in Chapter 6.

2. **Technological selection.** *Designers* are in charge of deciding the technologies to be used in the support of the business process. Since many different technologies can be applied for supporting the participation of physical elements in business processes, the different technologies are analyzed. By studying their capabilities and limitations, designers can **select the technologies** that best fit with the requirements of the business process.

   The **deployment configuration** defines which computing re-

sources (e.g. mobile devices, sensors, etc.) are used for each task in the process. In a mobile business process, each mobile device could be used for supporting from one single tasks to a whole process. Depending on this, the technologies required for each device can be determined.

Further detail regarding the linkage between the business process requirements and the definition of the technological setting where the process is deployed is provided in Section 4.3.

3. **Workflow simulation.** *Designers* perform several design iterations to obtain feedback from end-users and improve the workflow design. Before efforts are put into development, fast-prototyping is applied to validate the workflow defined. Feedback is gathered from end-users in order to determine whether the proposed workflow improves the existing business processes in terms of user experience and process performance. Section 4.4 provides the guidelines to perform the workflow simulation.

4. **System implementation.** The models obtained at this point are the input to the next stages of the BPM cycle. Once the models have been adjusted to fit with the user needs, a final software solution can be obtained. The derivation of a software solution from the system specification is described in Chapter 5.

The steps proposed in the method go from the abstract description of the workflow to the concrete system that supports it. Although it is natural when modeling to go from the abstract to the concrete, the design method has been defined to also support the reevaluation of the design decisions.

Figure 4.3 illustrates a possible decision process performed through the different modeling layers considered in this work. Decisions in one layer affect the choices available on the next level. For example, by selecting a particular technology for a task, only devices that are compatible with the technology can be used. Nevertheless, designers are not forced to follow a linear process. Since modeling layers are connected, decisions made on upper planes force the reevaluation of decisions made on lower planes to keep the system consistent. For example, if a mobile

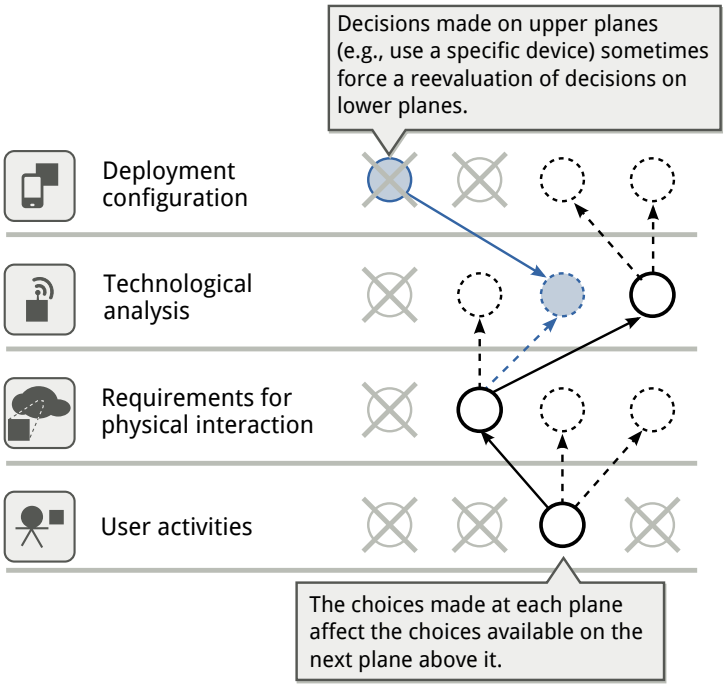


Figure 4.3: Design decisions across the different modeling layers

device must be used for a particular system (e.g., due to vendor lock-in), the technology used to identify physical elements may be changed to fit the device capabilities. The different aspects of the system specification can be kept synchronized thanks to the use of modeling techniques. The modeling technologies used allow to formalize the concepts used for the design of physical mobile workflows in order to automatically detect inconsistencies. More detail about this is provided in Chapter 5.

The following sections provide further detail on the concepts (dependent and independent from identification technologies) that are captured to specify physical mobile workflows. These concepts are later used to guide the execution and adaptation of the workflow at run-time.

4.2 Capturing technology-independent requirements

When users participate in a workflow, they are under a constrained freedom. They can choose which tasks to perform, but they are limited in their decisions by the rules of the business process (e.g., a book can be borrowed by a library member only if he/she has not been sanctioned). Business process modeling promotes the definition of such rules in an explicit manner. This is usually done graphically by means of a business process diagram. A business process diagram defines the activities that are performed in the process and their temporal dependencies, the roles that participate in the process, the events that can occur and the different decision points that determine the flow of the process.

A graphical diagram is a good tool for designers to discuss the different ways a business goal can be achieved. Designers can agree on *who* participates in the process, as well as *when* and *how*. In this work BPMN is used for capturing such aspects in a graphical diagram. BPMN has been selected since it provides a standard graphical notation for describing service orchestration that is easy to understand for analysts and it is independent form the underlying implementation technology. Our approach enriches business process diagrams to capture the particular requirements for the integration of physical elements. In particular, it is specified (1) the obtrusiveness level for each task in the process and (2) the interaction mechanism used for connecting the physical and digital spaces.

One of the aspects that must be considered when a physical mobile workflow is modeled is the **participation of physical elements in the process**. When relevant elements of interest for the process have presence in both digital and physical worlds our approach can be applied to improve the physical-virtual linkage in the process. Thus, it is important to detect the physical elements involved in the process and their properties.

Modeling notations usually provide primitives to indicate the income resources or the outcome for tasks, as is the case with the *Data Object* element in BPMN. However, as its name indicates, this primitive only

deals with digital aspects. Thus, requirements regarding the integration of physical elements can only be informally stated in BPMN by means of documentation annotations.

In this work, we annotate *Data Object* elements with new properties in order to cover its connection with the physical space. Different information is defined for physical elements in addition to the ones such as *name* or *documentation* attributes, which are inherited from the *Data Object* element as indicated by the BPMN specification. The information to be captured for each object is the following:

Information structure. A Class Diagram can be used to capture relevant entities, their information and the relationships that connect this information. The data model should not include any artificial attribute for identification. In this way, information is decoupled from the way in which it is identified.

When modeling real-world elements it is important to decide at which granularity level information is defined. For example, if the products of a supermarket are labeled on an individual basis they could have an individual price potentially different from other products of the same type (e.g., based on expiration date). In this case, the price should be defined as an attribute of the product. However, if all the products of the same type have the same price, price attribute should be considered as part of the product type. Even more, if all the relevant information is about the product type, is the product type identifier what should be present in each product label.

Use for correlation. A physical element can be used to find the current process instance. In this way, the process becomes more fluent since the detection of a given element is used to determine the task being performed. Each *Physical Object* primitive includes a Boolean attribute to indicate whether the physical element is used for correlation in the process.

Physical elements that can participate only once in a process instance (such as a book in a loan) are good candidates for correlation. In the library example, when returning a book it is clear to

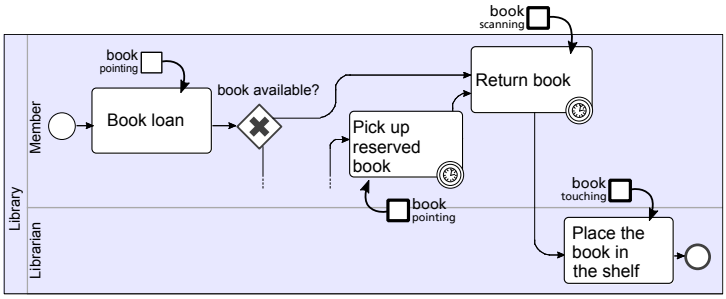


Figure 4.4: Excerpt of the BPMN model for the book loan workflow in a library

which process instance it belongs since one book can be involved in only one loan at a time.

Figure 4.4 shows an excerpt of the BPMN diagram that represents the loan process for the *Smart Library* scenario. The participation of physical elements in a task is represented by means of a small square. Physical elements can either be required or generated by a task, which is depicted with an association between the object and the task that is defined in one direction or another. When a task needs the information associated to a physical element, the direction of the association is from the object to the task. In contrast, when a physical element is generated by a task, the association is defined in the opposite direction. The correlation role is represented using a thick border.

In the example, books are detected using different identification technologies and interaction techniques depending on the automation level required. Identifiers printed on paper (e.g., barcodes) can be accessed by the mobile devices of much of the potential library members. Users can make use of an implementation of the pointing interaction technique to access barcodes by means of their phone camera. In this way, users can carry out the *book loan* and *pick up reserved book* tasks. These tasks are performed autonomously by library members without the assistance of the libarian.

In order to allow an autonomous return of books, books should be detected when they are placed inside the return box without further

human intervention in their identification. Since a greater degree of automation is desired for this task, radio frequency-based identification (i.e., using the *scanning* interaction technique) is used for book identification in this task. This task also illustrates the use of physical elements for process correlation. Books are used to determine which process instance is associated with the *return book* task.

In order to determine the adequate interaction technique to be used in each task, the automation level required for the task and the available identification mechanisms are studied in an abstract manner. Each task in the workflow is analyzed to determine to which extent the interactions involved must intrude the user mind (the obtrusiveness level). In this way, the degree of automation of the process can be adjusted to the requirements of each particular task and the appropriate interaction technique can be selected from the ones available. Some tasks could require implicit interaction in order to allow the process to be more fluent, while others would require the users to be completely aware of the system actions.

The following subsections provide more detail on the specification of the obtrusiveness level and the selection of the appropriate interaction technique.

4.2.1 The obtrusiveness concept

Since user attention is a valuable but limited resource, an environment full of embedded services must behave in a considerate manner, demanding user attention only when it is required (Gibbs, 2004). In a business process, the user must be aware of the relevant information only. It is the task of the designer to determine which tasks the system can perform automatically and which ones must the user be aware of.

We make use of the conceptual framework presented in (Ju & Leifer, 2008) to determine the **obtrusiveness level** for each task in a physical mobile workflow. This framework defines two dimensions (see Fig. 4.5) to characterize implicit interactions: *initiative* and *attention*. According to the *initiative* factor, interaction can be *reactive* (the user initiates the interaction) or *proactive* (the system takes the initiative). With regard to the attention factor, an interaction can take place at the *foreground*

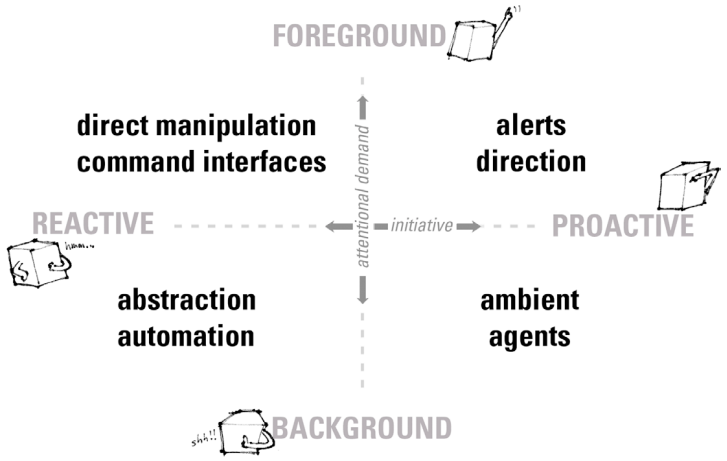


Figure 4.5: Framework for characterizing implicit interactions (Ju & Leifer, 2008)

(the user is fully conscious of the interaction) or at the *background* of user attention (the user is unaware of the interaction with the system).

Other frameworks exist for the definition of implicit interactions (Buxton, 1995; Horvitz et al., 2003). However, we found it very useful to consider *initiative* and *attention* as independent concepts. In the case of physical mobile workflows, automation and user awareness are factors that usually vary independently from task to task.

According to our proposal, once a business process is defined, the designers must indicate the possible obtrusiveness levels for each task. The conceptual framework used in this work defines two axes to represent obtrusiveness. For each task, we take this space and make different divisions for each axis. Designers can divide the obtrusiveness space into many disjoint fragments to provide specific semantics to each fragment. The only rule that must be followed when dividing an axis is that the ordering must be preserved in each axis for the defined values.

Figure 4.6 illustrates the linkage between different tasks and the obtrusiveness space for the interactions that support each task. The *initiative* axis in this case is divided into two parts: *Reactive* and *Proactive*. The *attention* axis is divided into three segments, which are associated

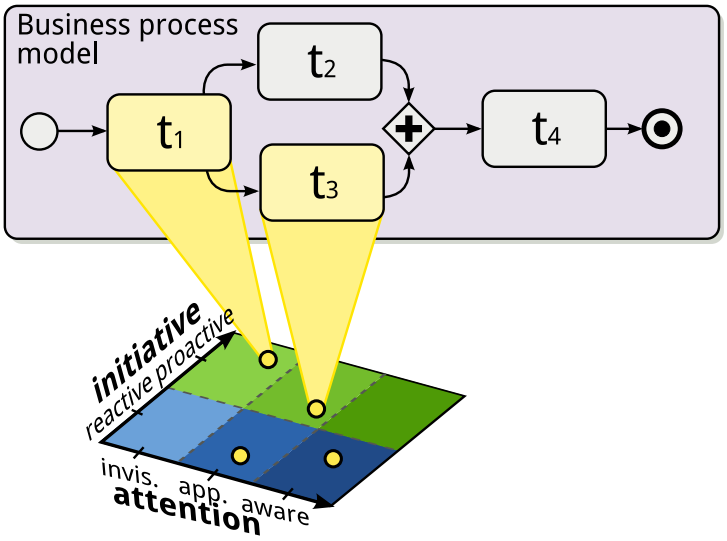


Figure 4.6: Tasks from a business model are associated to a region of the obtrusiveness space.

with the following values: *invisible* (there is no way for the user to perceive the interaction), *slightly-appreciable* (usually the user would not perceive it unless he/she makes some effort); and *awareness* (the user becomes aware of the interaction even if he/she is performing other tasks). Designers can divide each axis in as many parts as they require for describing the obtrusiveness level for the process. This division is later considered for selecting the appropriate interaction mechanisms for each element involved in the process.

These divisions allow designers to classify the different interaction techniques available. In this way, designers can later chose the interaction technique that best fits with the requirements captured. More detail on physical interaction is provided below.

In some cases the most adequate obtrusiveness level for a task depends on some context conditions and cannot be determined at design time. For example, tasks such as remember hospital patients to take their medication can be performed at different obtrusiveness levels depending on the delay from the moment each patient was supposed to

take the medication. Chapter 6 provides mechanisms to adapt the workflow to changes in the obtrusiveness level.

4.2.2 Physical interaction

Users can interact with a physical element in different manners. For example, users can access the services that are associated with an element either by *pointing* to the element, *touching* it, or scanning nearby elements with their mobile device. These are only an example of the interaction techniques that have been defined for the interaction between users and their surroundings in the literature (Ballagas et al., 2004; Broll et al., 2008; Iftode et al., 2004; Rukzio et al., 2006). However, there is not a universal interaction technique that is well suited for any situation. The selection of an adequate interaction technique depends on (1) the user activities and (2) the feasibility for its application.

The analysis of user activities and the physical elements that are involved in each task is part of the design method proposed and is useful to determine the most appropriate techniques for each task in the workflow. Designers can found in the literature guidelines to determine the pros and cons of each technique (Rukzio et al., 2007, 2006). For example, the *scanning* interaction technique has been selected for the return of books in the example since it allows books to be detected automatically with minimal user intervention. However, the interaction technique selected also depends on the identification technology used for the physical element. For example, *touching* cannot be applied if the physical element is only identified by means of a barcode.

This section introduces the *medium* concept in order to represent which interaction techniques can be applied in each kind of identification technology. In order to handle the technology heterogeneity in Auto-ID, the medium concept is useful to represent technologies with similar properties.

Applications in the IoT lay on top of a heterogeneous collection of ubiquitous devices (Rellermeyer et al., 2008). As a consequence, physical elements can be identified in a myriad of different ways (e.g., using a sequence of bars on a paper or a radio wave emitted by an RFID tag). *Mediums* are defined in this work as physical supports

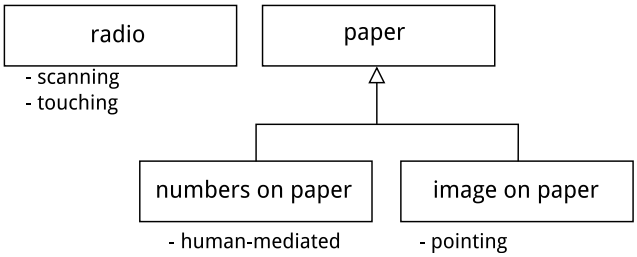


Figure 4.7: An example of different mediums and specialization relationships

for identifiers. *Business Analysts* can define *mediums* to represent a kind of technologies in an abstract manner. For example, the *Business Analyst* can define the *paper* medium (see Fig. 4.7) in order to describe in an abstract manner a set of identification technologies that have some requirements in common: (1) the technologies must be **cost-effective** by using identifiers that are cheap to produce and that require simple devices for their capture; and (2) the technologies require **direct line-of-sight** since identifiers are recognized optically.

For each medium we consider (1) which regions of the obtrusiveness space it supports and (2) the interaction mechanisms that can be used to access them. The regions of the obtrusiveness space are defined according to the framework introduced in Section 4.2.1. The interaction mechanisms for interacting with the real world through a mobile device are based on those defined by Rukzio (Rukzio et al., 2006). Some examples are the selection of a real world object by *touching* it with a mobile device, *pointing* at it with the device, *scanning* the environment for nearby objects, or applying *user-mediated object interaction* (the user types in information provided by the object to establish a link).

The *medium* concept is a technological-independent mechanism for describing identification requirements. This concept is useful for guiding the later technological decisions. For the Business Analyst, the *paper* medium is related to an identification mechanism that requires direct line-of-sight and that is cheap to process. This information is useful to determine that the *paper* medium is appropriate for supporting the participation of the library members in the process. However, the analyst does not need any knowledge about the technologies that support this

medium (e.g., using fiducials with a video camera or QR Code with a photo camera from a mobile device). Alternative identification mechanisms (e.g. enclose the machine processable identifier with a more human-usable one) are expressed by indicating multiple mediums for a *Physical Object*. By combining different mediums their weaknesses can be palliated. Later, any technology that supports those requirements can be used for the system implementation.

Business Analysts can organize mediums in a hierarchy in order to better capture their commonalities and variability. In the example, the *paper* medium is specialized in two mediums which are considered as sub-types. These sub-types distinguish between an identifier that is expressed by means of numbers from one that is expressed by means of an image. In the example of the figure, *image on paper* and *numbers on paper* are paper-based mediums but the interaction mechanisms they support are different. The *image on paper* medium supports the *pointing* mechanism, while the *numbers on paper* medium requires users to perform the identification and type the associated identifier (i.e., user-mediated interaction).

This study of the commonalities and variability of medium properties is also useful for **supporting process evolution**. When business processes are re-engineered new technologies can be considered for their support. By classifying the new technologies according to the existing medium hierarchy, it can be determined whether these new technologies are really adding value to the process or not. For example, fiducials (Bencina & Kaltenbrunner, 2005) are visual markers that can be detected in real-time with a video camera. One particular aspect of this identification technology is that multiple elements can be detected at the same time. Thus, the *scanning* interaction technique could be used for elements identified by means of fiducials. From this analysis we can conclude that using fiducials to replace QR Codes is only worth it if the use of the *scanning* interaction technique is really required by any process task currently supported by QR Codes. For any other case fiducials fall in the same medium category as QR Codes (image on paper medium), and there is no need for re-tagging all the physical elements involved in the process.

4.3 Technological requirements

The modeling primitives introduced in the section above allow the specification of business processes where physical elements are involved. Using these primitives, requirements for the integration of the physical and the digital spaces are defined in an abstract manner. In order to determine the technological resources for supporting the business process, *Designers* should take different implementation decisions. These decisions are also captured by means of models in order to (1) explicitly state their rationale and (2) to use this knowledge for automating the development in later steps.

The aspects that are captured in this technological perspective of the system specification are (1) the Auto-ID technologies selected for supporting the business process, and (2) how the system functionality is distributed in different computing devices. the following subsections provide more detail on these aspects.

4.3.1 Technological analysis

The different *mediums* considered in the business process define a set of requirements for identification (such as being machine-processable, cost-effective, etc.). The available technologies should be analyzed in order to determine which one best fits the *medium* requirements. The mediums supported by each technology are indicated in Table 4.1 for supporting the example based on a library workflow.

For the *Smart Library* scenario, the considered technologies are *RFID*, *QR Code*, and *Text Label* (text printed on a label that can be read by a human). *RFID* provides identification capabilities that support the *radio* medium. *QR Code* and *Text Label* provide identification capabilities by means of the *paper* medium, but both have specific requirements. *QR Code* technology uses images, while the *Text Label* uses numbers to represent identifiers.

Books in the Smart Library scenario are required to be identified in such a way that (1) minimal human intervention is required for the library staff and (2) mechanisms in the average mobile devices of the library members can be used. To respond to these requirements, two

Technology	Medium/Technique	Function	Resource Type
RFID	radio/scanning, touching	capture, produce	RFID antenna
QR Code	paper/pointing	capture	photo camera
		produce	printer
Text Label	paper/user-mediated	capture	keyboard, keypad
		produce	printer, pen

Table 4.1: Technologies, mediums and resources for the Smart Library scenario

technologies are used for book identification: RFID and QR Code. On the one hand, RFID provides the highest level of automation from the considered technologies. Thanks to this, when returning a book, technology is completely transparent for the user. In comparison to RFID, QR Code offers a lower level of automation since the user interaction is more explicit. However, QR Code results in a cost-effective identification mechanism that can be used by much of the average mobile devices available today.

Once the technologies are selected, the required resources for their implementation should be determined. Table 4.1 defines the possible resources that can be used for each technology and interaction technique supported for a medium. *RFID* technology requires an antenna for both producing and capturing identifiers. RFID is capable of supporting two interaction techniques (touching and scanning). Image capturing devices are required for *QR Code*. While input devices are required for *Text label* capturers (for the example, a keyboard or a numeric keypad are considered). Printing resources are required for the identifier production of paper-based technologies.

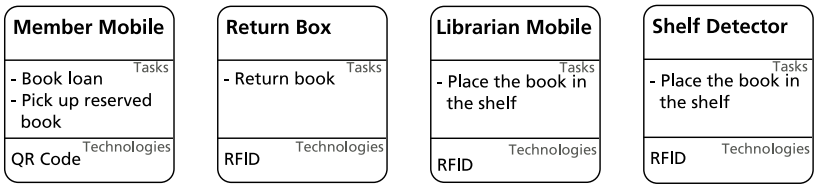


Figure 4.8: Auto-ID services involved in the Smart Library

4.3.2 Deployment configuration

Functionality in support for a mobile business process is normally distributed across different computing resources. In order to support the integration of physical elements in the different tasks from a business process, the identification functionality should be organized. This involves defining the setting of the different resources for the system. For example, in the Smart Library case study, each return box should be equipped with an RFID antenna to enable the detection of books when they are returned. We use the concept of *deployment unit* to encapsulate the functionality required for the support of different tasks by means of a set of technologies that is deployed in a particular device.

To provide an abstract view of the *Smart Library* setting, the different deployment units should be specified. Figure 4.8 represents a diagram for the Auto-ID-related *deployment units* for the *Smart Library* case study. The following are defined for each *deployment unit*; the task that it supports, the physical elements that are involved and the technologies that are used. The *Member Mobile* deployment unit represent a set of software components that support the *book loan* and *pick up reserved book* tasks by making use of QR Code technology for their completion. The software solution for this deployment unit is accessed by the library members from their mobile devices. The *Return Box* deployment unit, is in charge of automatically detecting the returned books by means of RFID. Thus, each *return box* requires one or several RFID antennas capable of detecting its content. The *Mobile Librarian* deployment unit is accessed by the librarians from their RFID-equipped mobile devices in order to transfer the returned books from the *return box* to their shelf. The *Shelf Detector* deployment unit

is also supporting the *place the book in the shelf* task. In this case, it detects whether a book is placed in a wrong shelf.

4.4 Validating the design

The previous sections introduced a design method for the definition of physical mobile workflows. However, when a workflow is designed, there is no guarantee that the resulting system could met the user expectations. The BPM cycle suggest to perform simulations of a business process before it is finally executed. For the digital space, current business process design tools provide simulation capabilities. These tools can be used to determine whether the flow defined for tasks is the appropriate or not. They provide capabilities for simulating the messages that are exchanged during the business process. In this way, external services are not required to be actually implemented. However, when these business processes take also part in the physical world different issues arise for their evaluation.

This section provides techniques based on User Centered Design (UCD) (Mao et al., 2001) for the evaluation of the impact for users of a physical mobile workflow when it is performed in the real world. UCD techniques are common to evaluate systems in the Internet of Things domain (Carter & Mankoff, 2004). Thus, we applied some of these techniques for the simulation of business processes that take place in the Internet of Things. This section introduce a technique for the early-stage evaluation of physical mobile workflows by means of fast-prototyping. Our research results show that even though the proposed prototypes can be built quickly, they are capable of reproducing a level of user experience that is considered to be very close to what users expect from a final system. Thus, flaws in the workflow design can be detected before efforts are put into the development of the final system.

4.4.1 Requirements for the evaluation

Researchers have shown that evaluating ubiquitous systems can be difficult (Neely et al., 2008). Many factors required for the evaluation

of a system cannot be reproduced in a lab, but in-situ evaluation is also challenging and not feasible in many cases. Since a one-size-fits-all approach for evaluating ubiquitous systems is unrealistic (Neely et al., 2008), we have analyzed the specific application domain we are targeting and propose an evaluation model that fits the detected requirements. We detected the following challenges for the evaluation of physical mobile workflows:

Fast evolution. Business processes are fast-changing. We must be sure that the implementation of the corresponding software solution is worth it. This requires performing an **evaluation of the system at an early stage** of the development process that is both accurate (it provides a good estimation of the benefits to be obtained) and easily developed (it requires minimal effort).

Concurrent environment. The evaluation of a ubiquitous system must take into account the integration with the rest of the activities that the user is involved in (Neely et al., 2008). This is especially relevant for physical mobile workflows since users are normally involved in many business processes at the same time. Thus, a workflow must be evaluated considering the interleaving tasks in which the user participates.

Evaluation from the user and the process perspectives. Usability and process efficiency are the key factors that determine the mobile business service experience (Vuolle et al., 2008). These aspects are affected by different factors such as the kind of mobile device used or the way in which the service is presented. When evaluating physical mobile workflows, both perspectives should be considered: user perception of the process and the productivity increase for the system.

To fulfill the above requirements, we propose the use of early stage prototypes. Early-stage evaluation techniques, such as paper prototyping, typically require little effort, time and money (Vredenburg et al., 2002). These techniques enable iterative design, and providing frequent feedback about the potential of the designs.

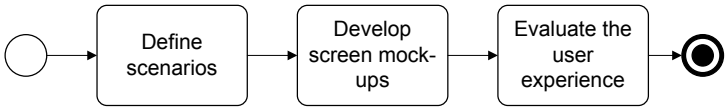


Figure 4.9: The different tasks in the development method proposed

4.4.2 Fast-prototyping for physical mobile workflows

The overall approach for the fast-prototyping of physical mobile workflows is illustrated in Figure 4.9. The goal of our evaluation method is to immerse the user in an environment that **makes the user feel as if he/she is using the final system** despite the fact that a non-functional prototype is being used. The first step in the evaluation is to define a scenario according to the workflow that is being designed. The scenario should consider the **concurrent nature of business processes**. According to this scenario, users are provided a script to guide their actions. For the library example, we combined book loan tasks with other tasks related to book information gathering (e.g., finding similar books).

An HTML mock-up is designed for each task in the BPMN diagram. These mock-ups provide the user with the expected response for each task in the process (e.g., a form, a confirmation message, etc.). Since the users have to follow a script that conforms to the business process, it is easy to anticipate the results that can be obtained. In addition, the availability of powerful editors for HTML documents and multimedia content makes this approach **quick to apply**.

The interaction with the real world is simulated using Wizard of Oz techniques (Dahlbäck et al., 1993). An operator provides the current physical context using another mobile device (see Fig. 4.10). The operator is in charge of providing the correct context information (e.g. triggering the book detection when a user "touches" the corresponding book). In this way, the user is immersed in an environment that behaves like a working system with Auto-ID capabilities, but it is much easier to produce.

In order to obtain fast-prototypes for physical mobile workflows, we have followed the steps described below.

Figure 4.10: In-situ evaluation applying Wizard of Oz and HTML prototypes

1. **Define a scenario according to the process definition.** A specific instance of the workflow is defined to illustrate a relevant use case. For example, we can define a scenario consisting in a user that tries to borrow a specific book which is blocked and then tries to find a similar one.

2. **Detect the physical objects that participate in the scenario.** An interface must be provided to the operator that exposes the possible detection events. This requires to analyze the relevant states for the physical elements that are involved in the scenario. Since prototypes are handling static information, a change in an object state is easily handled as a different object. For example, the operator can be provided two different events: one for triggering the detection of a book when it is blocked and another for triggering the detection og the book when it it is released. In this way, the associated mock-up screen can better reflect the appropriate information for the element.

3. **Define the screen mock-ups for each step of the process.** Depending on the obtrusiveness levels considered, the elements can be represented in a different manner (e.g., as taskbar notification). Links between screens must be provided in a decoupled manner. The screen to be shown depends on the physical context

and the action to be performed.

4. **Provide complementary options and error messages.** The screen mock-ups support the interaction that is required for the scenario defined. However, to provide more realism links to additional functionality can be provided. If the user access this links, a warning would be shown instead. This can be useful to monitor in which situations the users are missed in the script provided.

For the development of HTML mock-ups two opposed requirements are faced. We want mockups to be realistic but we also want them to be very easy to develop. To face both requirements we considered the use of a set of HTML utilities for the development of Mobile Web applications. In particular, we used the iUI framework [1] which provides javascript and CSS stylesheets in order to achieve a look-and-feel for HTML that is similar to mobile applications in the iPhone. Listing 4.1 shows an excerpt of an HTML prototype developed.

Listing 4.1: Excerpt from the HTML protoype.

```
<ul id="home" title="Tasks" selected="true" hideBackButton=
    "true">
  <li id="pending_group" class="group">Pending tasks</li>
  <li id="ret-link"><a href="#pending">Return two books</a
      ></li>
  <div id="p-task"></div>
  <li id="actions_group" class="group">Actions</li>
  <li><a href="#decode" onclick="setAction('borrow');">
      Borrow book</a></li>
  <li><a href="#decode" onclick="setAction('reservation');"
      >Fifteen minute reservation</a></li>
  <li><a href="#decode" onclick="setAction('comments');">
      User comments</a></li>
  <li><a href="#decode" onclick="setAction('similar');">
      Similar books</a></li>
</ul>

<ul id="pending" title="Return books">
   <li id="by">Start at the end with smart requirements</li
       >
```

[1] http://code.google.com/p/iui/

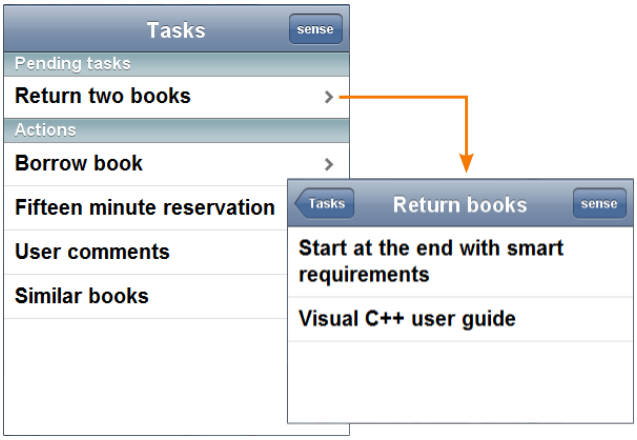


Figure 4.11: HTML prototype

```
    <li id="bx">Visual C++ user guide</li>
</ul>
```

The code illustrated above includes two screen mock-ups. The iUI framework shows only an HTML fragment at a time from the many that can be defined in an HTML file. Depending on the user navigation iUI performs the appropriate transitions among screens. Two fragments are defined in the listing above. The first one defines the initial pending tasks and actions. When an action is clicked, a variable is set to indicate the action. The second one defines the information associated with the "Return two books" pending task. Figure 4.11 shows the rendering of these HTML mock-ups.

The proposed technique makes it easy the application in an environment that is close to the real one since it does not require much infrastructure. To apply this approach we only need WiFi connectivity, a mobile device with a Web browser and a physical environment that is similar to the real one. For example, we use our department library to simulate the business process designed for the faculty library.

In order to obtain valuable feedback from users the MoBiS-Q questionnaire (Vuolle et al., 2008) has been used. MoBiS-Q is a questionnaire that evaluates user experience for mobile business services combined

with enhancements in work productivity. An example of the application of the evaluation technique can be found in Chapter 7.

4.5 Conclusions

This chapter introduces a design method to specify physical mobile workflows. The introduced approach is connected with other stages from the BPM cycle and allows physical mobile workflows to enter the BPM cycle from the beginning. It has been illustrated how to capture the requirements for the physical-virtual linkage in a gradual manner by means of modeling techniques. The use of models has been useful to centralize the knowledge about the workflow and organize it in a way that it is easy to handle for designers (e.g., work with technology-independent concepts, detect inconsistencies, etc.).

The designs defined according to the method can be easily put into practice. Fast prototyping techniques have been used to validate the mechanisms used for supporting the physical-virtual linkage for each task in the workflow. The feedback obtained from the evaluation can be used to better adjust the models defined at design-time.

The design method provided relies on proven techniques and frameworks for business process modeling, implicit interaction design and physical interaction patterns. The following chapters make use of the models defined to provide execution and adaptation capabilities for the final software solution.

Chapter 5

Automating the development

Modeling techniques describe a system by handling abstractions of the problem space. This allows to express designs in terms of concepts from the application domain instead of concepts from the technical space (Schmidt, 2006). By raising the level of abstraction, systems can be developed without taking into account much of the implementation details of the underlying platform. Models are used to organize the knowledge about the problem domain in order to guide the development. Furthermore, when models are machine-processable and precise-enough, they can be used to automate the production of a software system (Pastor & Molina, 2007).

This chapter applies Model Driven Engineering (MDE) principles (Favre, 2004) to automate the development of physical mobile workflows. Thanks to MDE techniques, it has been possible to traverse the gap between the high-level concepts used at design and the technical details of the particular mobile platform that is used for the system implementation.

Figure 5.1 illustrates how our approach connects the concepts used at design (e.g., task, obtrusiveness level, physical interaction, etc.) with a particular implementation platform. Our approach introduces two

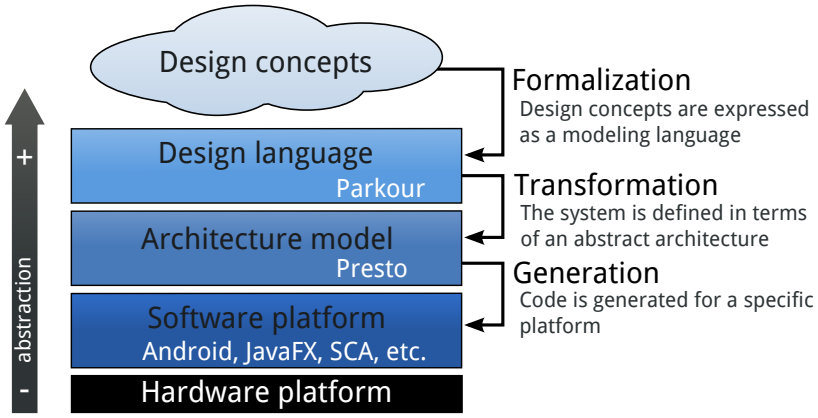


Figure 5.1: Strategy for covering the abstraction gap in the development of physical mobile workflows

layers to cover the abstraction gap between the design concepts and the software platform:

Parkour. The design concepts defined in Chapter 4 are formalized into **a modeling language** named Parkour. Designers can define the requirements for a workflow by following our design method and use the system specification during the development process. By formalizing design concepts, system descriptions can be processed to validate their consistency and automate some steps in the development.

Presto. Presto is an architecture specifically defined to support applications in the physical mobile workflow domain. Presto is **a sustainable software architecture**, that is, an architecture that can evolve over time throughout several technological cycles. The architecture elements are defined in a technology-independent fashion and code generation techniques are used to translate the generic architecture into components for each particular mobile platform targetted for development.

Model transformation technologies have been used in this work to translate system specifications based on Parkour into Presto-based mod-

els. Since both, the design and architecture concepts have been defined using modeling technologies, a mapping can be applied between them. We developed a model transformation that takes a Parkour specification as input and produces a first version of the architecture (defined by a Presto model). Then, developers can complete the Presto model and generate the corresponding code to different mobile platforms.

This chapter defines the automation support provided for the development of physical mobile workflows. Our proposal follows an architecture-centric approach: the different modeling layers are defined following a bottom-up approach by starting from the system architecture. First, Presto is defined as a model-based architecture that is capable to generate code. Then the abstraction level is raised by Parkour. Parkour was defined on top of Presto to provide a language that captures the requirements for Presto-based systems. Although Parkour and Presto are used in conjunction in this work, they are not tightly coupled since mappings among them are externally defined. Thus, Presto architecture can be also used in other development methods and Parkour descriptions could be mapped to a different architectures if necessary.

The remainder of the chapter is structured as follows. Section 5.1 introduces the architecture process followed in our approach. Section 5.2 defines the architecture by decoupling the architectural concepts from the mapping to a specific technology. Section 5.3 describes the way in which the development process is automated for the architecture defined. Finally, Section 5.4 concludes the chapter.

5.1 The Architectural Process

In order to offer support to business processes that involve real-world elements, many technological aspects should be considered. These aspects are related to business process management, Auto-ID technologies and mobile platforms. Some examples include Business process execution engines (based on different specifications like WS-BPEL or XPDL), interoperability solutions (such as Web Services or Advanced Message Queuing Protocol) to integrate different systems, middleware to control

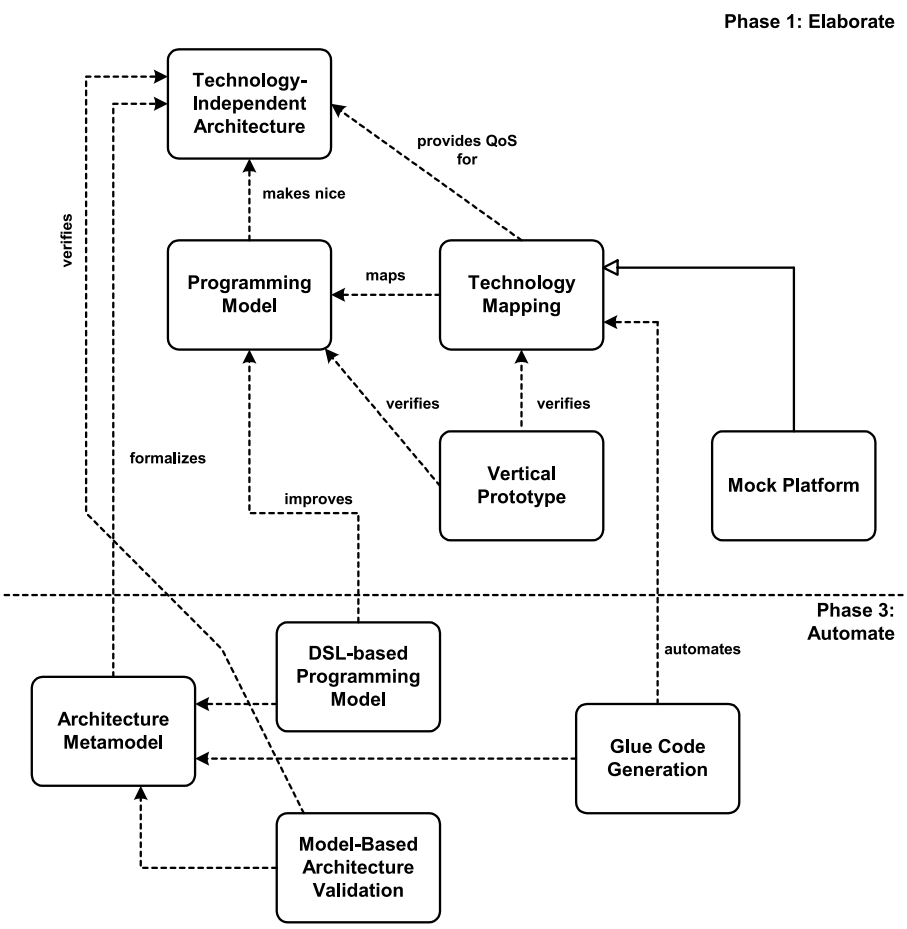


Figure 5.2: Phases of the architectural process (Völter, 2005)

Auto-ID events (such as Fosstrack[1] or Sun RFID[2] middleware), and mobile operating systems (such as Android, iPhone OS, or Symbian). The involved technologies are many in these fields, and new ones are expected to appear.

[1] http://www.fosstrak.org/
[2] http://sun-rfid.dev.java.net/

In order to avoid the the impact of this technology diversity during development, abstraction is used to cope with technological details. Based on the foundations of MDE it is proposed the use of models (abstract technology-independent description of systems) to face the automatic construction of this kind of systems. In order to define an architecture that supports this automatic development paradigm, the architectural process introduced by Völter (Völter, 2005) is followed. Völter proposes an architectural process (see Fig. 5.2) that is composed of the following steps:

1. **Elaboration phase.** The architecture is defined by decoupling the technology independent concepts from the actual technological solutions. The elaboration of the architecture defines the architecture first at a conceptual level. This constitutes a **technology-independent architecture**. Then, usage guidelines for the defined concepts are established in a **programming model**. The **technology mapping** defines how artifacts from the programming model are mapped to a particular technology. A **mock platform** facilitates testing tasks to developers. Finally, the development of a **vertical prototype** helps to evaluate the architecture and provide feedback about non-functional requirements such as testability, maintainability, scalability, etc.

2. **Iteration phase.** This phase consists in putting the architecture to work in order to **consolidate the architecture**. By iterating through the steps of the first phase, feedback is received from its use that can help to make the architecture more mature.

3. **Automation phase.** Finally, the use of the architecture can be improved by avoiding repetitive programming tasks with **automation**. Normally developers cannot completely focus on the domain concepts because they have to consider the conventions and the execution model followed in the architecture. By applying model-driven technologies, a system description can be automatically verified to be consistent with the architecture principles and the code that is required to enforce these principles (e.g., defining getters and setters for the domain objects, descriptors for the

components, etc.) can be automatically generated. In this way, software development for the architecture becomes more effective since developers can avoid to deal with low-level details.

The different steps that form the elaboration and automation phases are detailed in the following sections. Chapter 7 provides detail on the application of the architecture according to the iteration phase.

5.2 Elaboration of the Architecture

This section introduces the steps followed for the elaboration of the architecture. These steps go from the abstract to the concrete in a gradual manner. First, the aspects to be supported by the architecture are detected. Then, the architecture is defined at the conceptual level and mappings to a specific technology are established. The details on the elaboration are provided below.

5.2.1 Architecture requirements

Most of the current solutions for bridging the physical and the digital spaces lack flexibility required to support physical mobile workflows. A common approach for augmenting physical elements with digital services is to include links to the digital services (e.g., URLs or e-mail addresses) in the tags that are attached to physical elements. Since this information is embedded in the tag of the element, it is difficult to add, modify, or customize services without altering this tag (Broll et al., 2008). This coupling between identifiers and services can be assumed in many application contexts where the physical-virtual linkage is static, but it becomes a problem when dealing with physical mobile workflows.

In a physical mobile workflow, physical elements are augmented with services provided by different actors (manufacturers, maintenance companies, retailers, etc.). Users that participate in the process are normally only interested in a subset of these services, which depends on the business process context (i.e. user's role in the process and the activities in which the user is engaged). Each user has a particular point

of view about the real world that is influenced by their role in the process. For example, a shelf in a library can raise different questions in a library member (Is my book there?), the librarian (Is any book misplaced here?) or the maintenance technician (Is any repair required?).

According to these requirements we defined Presto, an architecture to support physical mobile workflows. Our design goal for Presto was to **support the physical-virtual linkage in a modular and extensible manner**. Our approach decouples identification of elements from the services that are provided to each user. In our approach, the physical-virtual linkage is considered to be a subjective relationship that depends on the observer. The services provided to the user depend on the user's role in the business process and his/her pending tasks. Presto provides a set of components with a well-defined functionality that can be composed in a pluggable manner for customizing the way in which the physical-virtual linkage is established. More detail on these components is provided below.

5.2.2 Technology-independent Architecture

In this section we present a technology-independent description of the Presto architecture. For the definition of the architecture, we rely on the *component* concept since this is a well-understood concept that can be implemented in most of the implementation technologies available. Components are the basic software pieces that conform the system. Component functionality is described by means of *interfaces*, and the communication among components is performed asynchronously. We have opted for asynchronous communication since the business processes considered are usually long-running and involve human participation.

Presto has been designed to guide users when fulfilling their tasks in a workflow by starting from the physical or the digital world. That is, users can discover the possible tasks to be performed when a given object is detected in the real-world but they can also decide their next task to perform from the information provided at the digital space. To support this behavior Presto provides two general operation modes: *object-driven* mode and *task-driven* mode.

In the *object-driven* mode, the physical context is sensed first and the

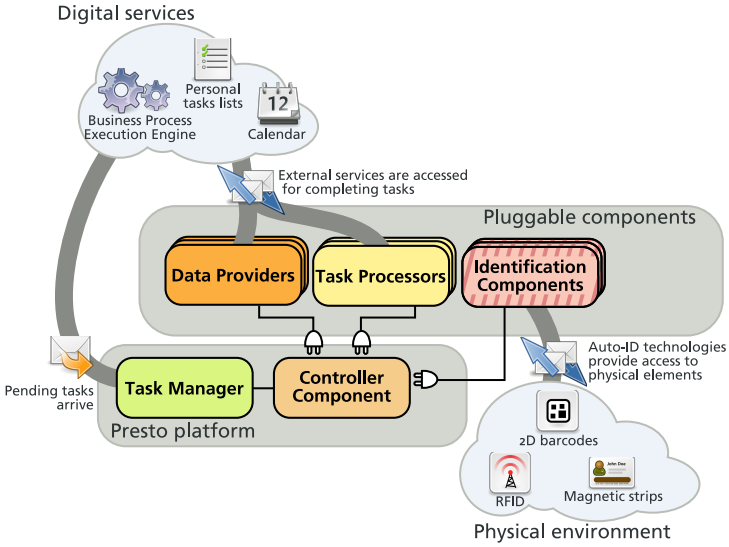


Figure 5.3: Architecture component overview

related tasks are then proposed to the user. For example, a user touches a book with a device and the services associated with this book (such as borrowing it) are presented to the user. In this way, the user finds out what can be done in a given physical context. In the *task-driven* mode, the user explicitly indicates which task he/she is performing. The physical context is then accessed for the purpose of completing this specific task. For example, the user by looking at their pending tasks decides to return a book and then he/she is provided indications to find this book.

In order to support the above operation modes, the system functionality is organized in components that can be composed in a different manner depending on the operation mode used. Presto architecture is composed of a *Task Manager*, a *Controller Component*, and several *Task Processors*, *Identification Components* and *Data Providers*. Fig. 5.3 illustrates the components involved in the Presto architecture and their connection to digital and physical spaces. More detail about these components is given below.

Task Manager. This component acts as a buffer where pieces of pending work are waiting to be completed. It receives messages from external services such as the information system of a library and waits for components to process them. The *Task Manager* can be queried in order to retrieve messages according to certain criteria (e.g., tasks of a certain type, involving a given physical element or targeted to a specific user).

The *Task Manager* has a central role in offering a **flexible physical-virtual linkage** since the information handled by this component (i.e., the business process context) is used to determine the services to be offered to each user. In desktop-based workflow systems, task management is centralized for all the processes under the control of a single organization. Conversely, in a mobile environment users change from task to task (across different organizations) on the go. Thus, it is desirable to allow users to access their pending task from a single application point. The *Task Manager* component provides a unified view for the pending work for all the processes in which the user participates. This distributed schema is common in mobile workflow approaches that keep part of the process state information locally in the mobile device (Hackmann et al., 2006; Pajunen & Chande, 2007).

Task Processors. These components are in charge of supporting the **extensibility of the platform** in terms of business functionality. A *Task Processor* is defined to handle a particular type of task (e.g., borrowing a book from the library). *Task Processors* provide the users with the required information, services and interaction mechanisms for completing the task.

Task-based UIs allow users to arrange their machine's interface architecture to present only those functionality required for a particular purpose (Goth, 2009). In our approach, *Task Processors* are the basic pieces to build a task-based interface that is presented to the user according to the physical elements detected and the workflow state. Considering that a mobile device is normally used by a single user as a personal proxy (Want, 2008), it is feasible to include all and only the functions that each user is

required to perform in a workflow. In this way, it is possible to anticipate what users can do at each moment by activating the corresponding *Task Processor*.

Identification Components. These components are in charge of supporting the **extensibility of the platform** at the technological level. *Identification Components* provide mechanisms for accessing the physical environment and transferring identifiers between physical and digital spaces by means of some Auto-ID technology. These components provide facilities for capturing and generating identifiers with a specific technology. Identification Components provide support to one or several interaction patterns between the mobile device and the real-world such as *touching*, *pointing* or *scanning* (Rukzio et al., 2006).

Data Providers. *Data Providers* are in charge of transforming the identifiers provided by *Identification Components* into information that is relevant for the user. *Data Providers* are in charge of dynamically establishing the connection between physical elements and their virtual counterparts. Thus, they are the basis for achieving a **flexible physical-virtual linkage**. Each *Data Provider* represents a possible projection of a physical element in the digital world. For example, a physical element such as a book can offer the access to services provided by the library, the author who wrote it or its publishers. Each of these perspectives is offered by a different *Data Provider*.

Controller Component. The controller is in charge of handling the **extensibility of the platform** by coordinating the above components. The *Controller Component* transfers the information among the different components in the architecture. The way in which information is transferred depends on the **operation mode** considered.

The controller determines which perspectives are valid for a given business and physical context. A *perspective* is defined in this work by the set of *Task Processors* that are associated with a digital counterpart of a physical element which is provided by some *Data*

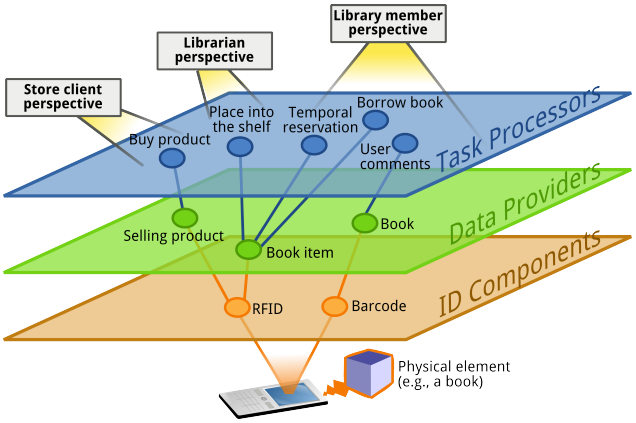


Figure 5.4: Example of the role of identification components, data providers and task processors

Provider. For example, the library perspective can include *Task Processors* that enable users to manage their loans, while an online shop perspective of the book would be associated with *Task Processors* for buying a copy of the book.

As Fig. 5.4 illustrates, the physical-virtual linkage is performed by Presto through different layers. A different perspective can be supported by selecting an appropriate subset of the components of the figure. In the example, two *Identification Components* provide different identifiers by means of RFID and barcode technologies. These identifiers are mapped by *Data Providers* into different digital counterparts. One data provider considers the physical element as a particular book copy that can be borrowed at item level. The other data provider considers the physical book as a representation of a literary work for which different copies exist. Each of these counterparts can be associated with a particular *Task Processor*. A *Task Processor* can either complete a pending task associated with the object or initiate a new task.

The introduced components represent the building blocks that compose any system based on Presto. These components define the technologies to use (Identification Components), the mapping with digital counterparts (Data Providers), and the associated supported tasks

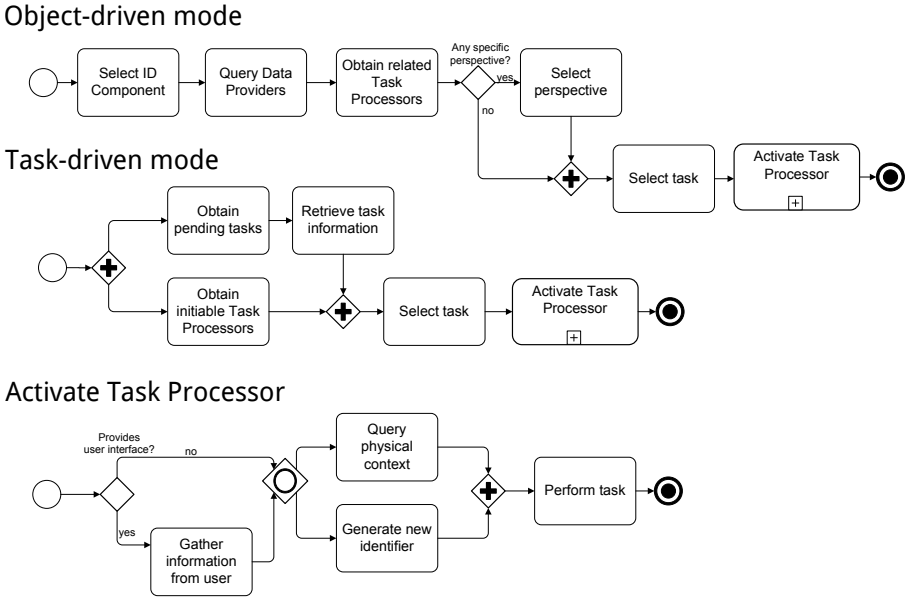


Figure 5.5: Different execution strategies depending on the operation mode used

(Task Processors). The next section introduces usage guidelines for combining these components in the development of Presto-based solutions.

5.2.3 Programming Model

Having defined the technology-independent architecture, we now define how this architecture can be used from the software developer's perspective. Since we have followed a pluggable model for the architecture, building a system consists in the assembling of the required plug-ins together. Thus, the construction of a Presto-based solution consists in (1) determining the plug-ins to be developed (or reused if they are already available) and (2) define how they are integrated in the platform.

Depending on the operation mode followed, the components are co-

ordinated in a different manner during execution. Each Presto plug-in includes some descriptive information that determines how it is integrated in the platform. The *Controller Component* uses this information to determine the behavior of the system when multiple plug-ins coexist. Figure 5.5 illustrates the different execution strategies that the Controller Component follows depending on the operation mode used. The *Activate Task Processor* sub-process which is common for both operation modes is also detailed in the figure. The information required to determine which path to follow in these process diagrams is provided by each plug-in. For example, each *Task Processor* indicates whether it provides a user interface or not. This information is expressed in a declarative manner by means of a set of attributes that is specific for each kind of component.

In the current architecture, *Task Manager* and the *Controller Component* are generic; therefore, they can be used for any application in the domain. For determining the rest of the components, the analysis of the business process is required. The following aspects should be determined: the tasks that compose the business process, the Auto-ID technologies required for identifying the physical elements that are involved in the process, and the information of interest for these objects (see Chapter 4 for more detail on how to capture these aspects). Each of these aspects is supported by a different kind of plug-in in Presto. A *Task Processor* is developed to support each of the tasks that requires user intervention. *Identification Components* are developed for the support of Auto-ID technologies. Finally, *Data Providers* are defined for each information repository required. The properties that are used to describe each kind of plug-in are introduced below.

Defining Task Processors

Task Processors are normally activated by the *Controller Component* as a response to some existing pending task. However, it is also possible to activate them for starting new tasks. In this case, the presence of a pending task is not required for offering functionality of the *Task Processor* to the user. A *Task Processor* is considered to be an *initiator* if it can be activated at any moment for the creation of a new piece of

work. The task-driven execution mode illustrated in Figure 5.5 handles *Task Processors* differently depending on the *initiator* property. This has also has an impact in the way Task Processors are presented to the user as it is illustrated below.

The Task Manager makes use of a dynamic to-do list metaphor to provide access to the different workflows that are supported by the system. Figure 5.6 shows a possible implementation of the dynamic to-do list. The dynamic to-do list represents the user pending tasks and the possible actions in a list:

Pending tasks. Pending tasks are pieces of work that must be performed by the user. Each pending task is associated to a *Task Processor* and contains part of the information that the *Task Processor* needs for handling it (e.g., the physical objects involved in the task). Figure 5.6 (left) shows two pending tasks that correspond to the same *Task Processor* (return book) but are associated with different physical books. When the user is performing a task, the corresponding *Task Processor* is provided the pending task information and the remaining information is provided by the user.

New actions. *Task Processors* that can be executed regardless of the existence of an associated pending task (i.e., those qualified as *initiators*) are presented to the user. In the library example, the *return book* task can only be accessed if the involved book is pending return. However, the *book borrowing* task is not performed in response to any pending work; it is performed by user initiative. *Task Processors* that can initiate a new workflow are shown in the "new action" section in the dynamic to-do list. As it is illustrated in Fig. 5.6, the actions that can be performed depend on the user perspective of the process.

By accessing pending tasks or triggering new actions, users make use of the *task-driven* operation mode. Users can access the *object-driven* operation mode to obtain only the tasks and actions that are associated with the current physical context. The sense button in Fig. 5.6 updates

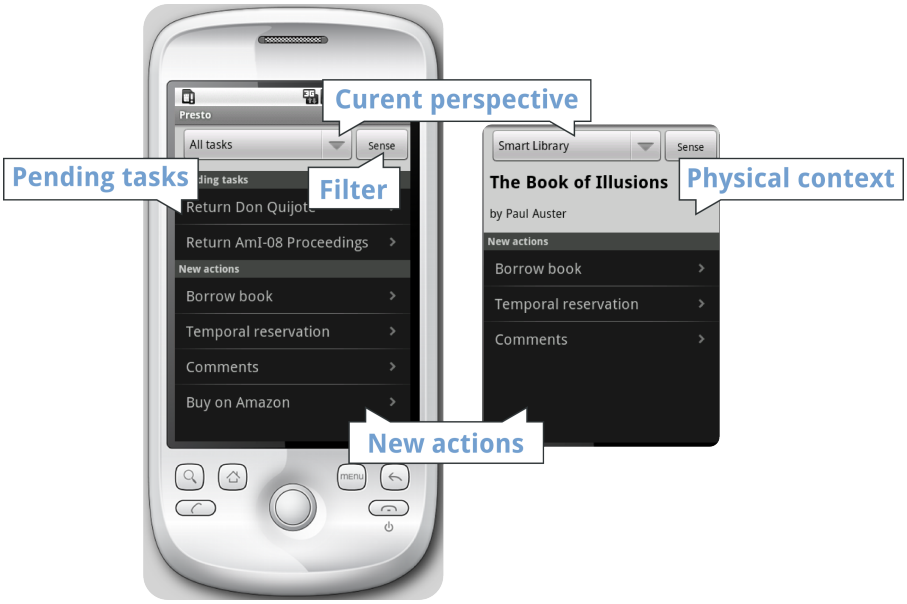


Figure 5.6: A possible implementation for the dynamic to-do list metaphor used in Presto.

the physical context and filters the to-do list in order to show only relevant items. When a physical element is part of several perspectives (i.e., several *Data Providers* provide information regarding a single object) the user can select the desired perspective to be only aware of the tasks that are relevant according to this perspective.

In addition, *Task Processors* can provide a user interface or not provide one. When a *Task Processor* is qualified as *silent*, the actions required for carrying out a task are performed automatically without demanding interaction with the user (see the "gather information from user" step in Fig. 5.5). *Silent Task Processors* can access the physical environment in an unobtrusive manner without disturbing the user.

Defining Identification Components

Identification Components are classified as *capturers* or *minters* according to their function. This classification is based on the identifier resolution mechanisms defined by Kindberg (Kindberg, 2002).

Capturer. *Capturers* are in charge of transferring identifiers from the physical space to the digital space. Barcode readers, or RFID receivers, are some examples. Capturers can offer identification on demand or by subscription. Thus, they should support both kinds of detection when implemented.

Minter. *Minters* are in charge of transferring identifiers from the digital space to the physical one. The creation of identifiers can be performed using the same technology used for capturing (e.g., RFID antenna is used for capturing and minting tags) or a different one (e.g., a barcode printer).

Identification Components can support one or both of the above functions depending on the limitations of the underlying technology and the different design decisions. This information is used to provide the user only with the *Identification Components* that can be used for each task in the process according to the identification needs of the task (detecting or producing identifiers).

In addition, *Identification Components* plug-ins indicate the *technologies* that they support. When locating *Identification Components*, the search can be constrained to a specific technology for tasks where the identification mechanism to be used is already known. This is useful for processes where the labeling technologies used are known in advance and not supposed to evolve in time.

Defining Data Providers

A *Data Provider* is in charge of (1) determining whether an identifier corresponds to an element of a certain kind, and (2) providing related information about this element. Digital services (e.g. EPC Information Services, product databases, etc.) can be accessed for both purposes,

for checking that the identifier belongs to a list of elements of a certain kind and/or for retrieving related information.

Data Providers provide digital information associated with some physical elements that correspond to a given perspective. *Data Providers* indicate the *namespace* (i.e., a unique name for the particular view they are providing) for which they offer their services. In this way, collisions are avoided when multiple *Data Providers* are present. In the case of multiple *Data Providers* covering the same *namespace*, all of them are queried until one of them provides results.

5.2.4 Technology Mapping

Having defined the technology-independent concepts that conform the architecture and the programming model that defines how to use this architecture, we now explain the technology mapping. The technology mapping defines how artifacts from the programming model are mapped to a particular technology. More detail about the **target technology selected** and how **architectural concepts can be mapped to the technological solution** is provided below.

Technology selection

To select an adequate target platform, we analyzed the most relevant aspects that are required for the support of physical mobile workflows according to our architecture: support for user interaction, the extension model provided, and the data abstraction capabilities. For the support of smart workflows, devices of a different nature must be capable of providing rich user interaction (i.e., integrating mobile resources such as touch screens, GPS information, etc.). The implementation platform is required to provide an extension mechanism to describe the components and allow them to be (un)installed at run-time for the system to be adapted. Finally, mechanisms must be provided to interoperate among components and between components and external systems at data level.

We have selected Android as a technological platforms that fit these requirements. Android is an Operating System and application frame-

work that is targeted to mobile devices. We have chosen the Android platform since it provides an open application framework that can be extended in different ways. In addition, several Android devices are available today, and new ones are announced. These devices include mobile phones, netbooks, eBook readers and digital picture frames among others. Thus, physical mobile workflows that span across many kinds of devices are possible with Android. More detail about the reasons for selecting Android as a target technology is provided below.

User interaction. Android provides support for advanced interaction techniques such as gestures or text-to-speech synthesis and easy mechanisms to integrate functionality from different applications. The communication mechanism defined among Android components is based on *Intents*. An *Intent* is an abstract description of a desired action (e.g., obtaining an image) regardless of the component that provides this functionality. Presto makes use of intents to integrate the functionality of applications such as the Barcode Reader that decodes different kinds of barcodes by means of the camera of the device.

Extension model. The Android platform provides loosely-coupled components such as *Service*, *Activity*, and *Content Provider*. A *Service* in Android provides functionality that is executed in the background, and an *Activities* provides the user interfaces from which service functionality can be accessed. A *Content Provider* offers data to other components. This component model allows components that share a common function to be easily replaced.

The Android Manifest is an XML file where the different Android components of an application are declared. For each component different attributes can be set such as the permissions it has, the intents to which it responds to, etc. In addition, the information for each component can be extended with new metadata.

Data abstraction. *Content Providers* are used for sharing data among Android components. *Content Providers* offer their data as a simple table on a database model where each row is a different record. This uniform API allows to access information regardless

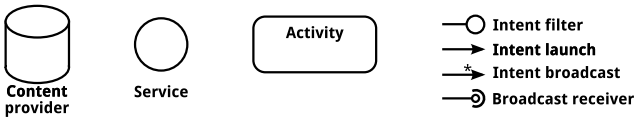


Figure 5.7: Graphical notation used to represent components of the Android application framework

of the internal mechanism used for storing them (file, database, web service, etc.). For the interaction with external services, a REST-based approach is followed. Presto makes use of JSON serialization capabilities in order to exchange information with external services such as business process execution engines.

The above aspects show that the Android platform fits well with the requirements defined for physical mobile workflows at technology level. Other mobile platforms such as the iPhone OS or JavaFX also provide advanced capabilities for the support of physical mobile workflows, but fail to completely cover all the above requirements. On the one hand, the iPhone OS is a closed platform where mechanisms for component interoperability and background processing are very limited. On the other hand, JavaFX mobile is mostly focused on the user interface level (e.g., not considering any extension model). In addition, the mobile profile for JavaFX is at a very early stage of development at the moment with no devices supporting it in the market.

Defining a mapping

In this section it is described how the Presto concepts are projected into the Android platform. Android provides an application framework with different components, and mechanisms to communicate them. This section presents the mapping between the abstract components defined in Presto and the components from the Android application framework. The mapping is described in this section in a visual and textual manner and it is later formalized in Section 5.3.2 in order to automate the development.

Figure 5.7 illustrates the notation used in this section to describe the

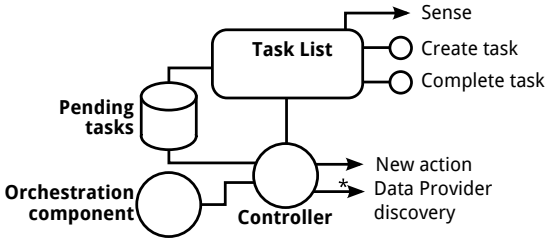


Figure 5.8: Implementation of the Task Manager and Controller Component

way in which each Presto component is mapped into Android. The main components from the Android application framework are represented in a graphical manner. The notation is aligned with other common notations such as BPMN or UML for the sake of intuitiveness.

The intent-based communication mechanism is also represented in our notation (see Fig. 5.7, right) to indicate their possibilities for the components to interact. The way in which components handle intents is depicted in a different manner depending whether we are describing the capabilities of a component to either launch or receive a certain kind of intent. When the broadcast mechanism is used, the previous notations are slightly modified to indicate so. The capability of a component to launch an intent is depicted by means of an arrow. If the broadcast mechanism is used, the arrow is decorated with an asterisk to indicate that it can reach multiple receivers. Since intents are used as abstract descriptions of an action, the target component is not always known at design time. When an arrow connects two components, it describes an explicit intent. However, arrows are not forced to be connected with a target element. In order to indicate that a component can respond to a given intent we make use of the lollypop primitive (used in UML for declaring an exported interface). When the component is a broadcast receiver the lollypop is decorated to resemble an antenna that can receive the broadcast.

The Android implementation of Presto includes two generic components (*Task Manager* and *Controller Component*) that are present in any system developed according to our architecture. Figure 5.8 shows the Android components used for their implementation.

Task Manager. The *Task Manager* is implemented by means of an *Android Activity* and a *Content Provider*. The *Task List* is the *Activity* that provides a user interface to the pending tasks and the new actions that users can perform for the different workflows supported. The *Task List* activity represents the tasks in a list that is divided in two sections (pending tasks and new actions). Users can **filter** the *Task List* (see the *Sense* intent in the figure) in order to access only the pending tasks and new actions that are relevant in the current physical context. When the user selects to perform some task (either pending or a new one), the corresponding intent is launched. This intent (labeled as *new action* in the figure) is created at run-time and the domain-specific information of the task determines the intent parameters.

Pending tasks can be added and removed from the *Task List* by the *Create task* and *Complete task* intents. Thus, the *Task List Activity* from Fig. 5.8 includes two intent filters. A *Content Provider* is used for storing the pending tasks and exposing them to other components such as the *Controller Component*.

Controller Component. The *Controller Component* is implemented as a service that is in charge of coordinating the execution of the different actions in Presto. When a pending task or a new action is selected by the user, the *Controller Component* launches an intent (*new action* intent). The actions that the user can perform depend on the existing perspectives on the physical context. For supporting a flexible physical-virtual connection, the different *Data Providers* are queried by means of a broadcast. The responses obtained for an identifier represent different perspectives on the element. In addition, a service named Orchestration Component is introduced to enable external services (such as orchestration engines) to inform the *Controller Component* about changes in the process state (e.g., new pending tasks).

The listing 5.1 shows an excerpt of the *Android manifest*, an XML file that describes the core package of Presto. In particular, the *Task List* activity is described by declaring the corresponding intents. In addition to the *create* and *complete* intent filters, the *main* intent filter

is used to indicate to Android that this activity is an entry point for the application.

Listing 5.1: Task List Activity described in the Android manifest.

```
<activity android:name=".Presto" android:label="@string/
    app_name" android:launchMode="singleTask">
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.
        LAUNCHER" />
  </intent-filter>
  <intent-filter android:label="Complete Task">
    <action android:name="es.upv.pros.presto.COMPLETE"></
        action>
    <category android:name="android.intent.category.DEFAULT
        "> </category>
  </intent-filter>
  <intent-filter android:label="Create Task">
    <action android:name="es.upv.pros.presto.CREATE"> </
        action>
    <category android:name="android.intent.category.DEFAULT
        "> </category>
  </intent-filter>
</activity>
```

Much of the controller logic for the Android version of the platform is simplified thanks to the flexibility in the intent processing mechanisms provided by Android. Different components can be used in an interchangeable manner as long as they can respond to the same intents. The way in which the pluggable components of Presto (*Identification Component*, *Task Processor*, and *Data Provider*) are mapped to Android components is illustrated in Fig. 5.9 and it is described below.

Identification Component. *Identification Components* in Presto are implemented by means of an Android Sevice. An *Identification Component* is in charge of minting identifiers, capturing identifiers or both. The capabilities for identification are determined by the intent filters defined in the service. Two activities are optionally complementing the identification functionality when user interaction is required to operate or configure the *Identification*

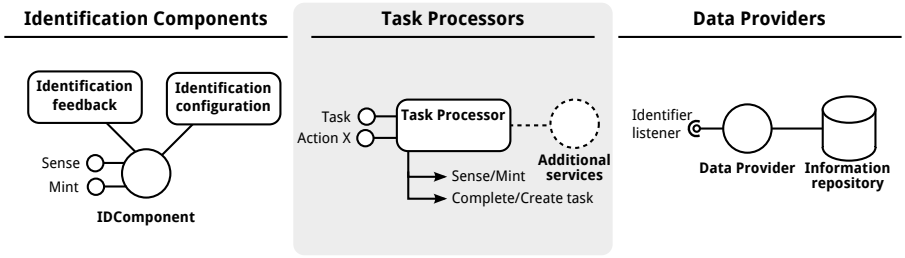


Figure 5.9: Implementation of the Presto pluggable components

Component. On the one hand, some feedback is usually required for guiding the user in the identification process (e.g., showing the video captured in real-time for QRCode detection). On the other hand, some identification mechanisms may accept different configuration parameters.

Task Processor. *Task Processors* provide specific processing for a given task in a business process. They are implemented by means of an Android *Activity*. The activity is defined to react to a specific intent (*Action X* in the figure) that represents its purpose depending on the domain (e.g., borrow book). *Task Processors* can invoke functionality that is provided by different *Identification Components* in order to access the physical world. They can also alter the process state by completing and creating tasks by means of the corresponding intents. In addition, other services can be also accessed in order to offer a better interaction with the user. For example, the notification manager service can be accessed to inform the user about a relevant event by means of the notification bar.

In order to distinguish which Android components are supporting each Presto component, we label Android components with their role according to the Presto architecture. We extend the *Task Processor* activities with the *task* intent filter. This intent indicates that the associated activity is a *Task Processor*. In addition, the category parameter of the intent can be used to determine whether

this *Task Processor* can be initiated by the user or it only handles pending tasks.

Data Provider. *Data Providers* are implemented as Android Services that react to the detection of a physical element by providing the information (if any) that they have regarding this element. This is achieved by means of a broadcast receiver. In order to provide the information associated to the detected element, a *Content Provider* is queried by the service.

By defining the technological mapping as a separate process, it is possible to project the architecture concepts defined into different technologies. In addition, the use of modeling technologies allows to automate the application of this mapping (see Section 5.3.2) and face the development of physical mobile workflows by capturing its requirements (see Section 5.3.3).

5.2.5 Mock Platform

Having decided the technology for the architecture, we now define a mock platform for developers. With a mock platform, developers can run tests locally as early as possible.

The use of loosely-coupled components for the definition of the platform allows for the easy definition of mock-ups that can be later replaced with the functional components. The mock components must respond to a set of intents depending on their function. For example, a mock *Identification Component* based on user input can be later replaced with a component that makes use of a more advanced identification technology such as RFID.

Android also offers different testing capabilities that can be leveraged when developing Presto-based solutions. Android includes a version of the JUnit framework for the definition of unit tests. The *android.test.mock* package provides mocks of various Android framework building blocks. For testing user interfaces, Android provides an instrumentation framework. The instrumentation framework allows to send key events, and make assertions about various UI elements. In addition, the Android Monkey tool is capable of generating pseudo-random

streams of user events such as clicks, touches, or gestures, as well as a number of system-level events. The Monkey tool can be used to stress-test applications in a random yet repeatable manner.

The use of JSON as the data serialization format to communicate with external services also makes it easy to define mock web services. A static text file can be used to simulate a external service without requiring a formal definition of the data schema (as it usually happens with XML). Later, the static text file can be replaced with a functional version of the service that generates the result in a dynamic manner.

5.2.6 Vertical Prototype

To validate the Presto architecture, we have implemented part of the Smart Library case study (more detail on the case study in Chapter 7). This case study is used throughout this work for illustration purposes and describes different business processes in a library. We chose a library context since it is well understood and it was easy to find people that were familiar with it.

In the development of the prototype we followed the guidelines for infrastructure design and evaluation defined by Edwards (Edwards et al., 2003). According to these guidelines we focused on the core infrastructure features, leaving out secondary aspects. While, secondary features such as security, distribution, replication and so on are all key to certain scenarios and even necessary for a "real world" deployment, they are not central to the value that Presto promised to deliver. Thus, we focused on the applicability of the architectural concepts of Presto and its operation modes. In particular, we evaluated to which extent the platform extensibility and the flexibility in the physical-virtual linkage offered by Presto improved the support for book loan process in a library.

In the Smart Library case study, library members can borrow books directly from the shelves where the books are physically located. In addition, we provide users with access to comments about the book and related information.

Figure 5.10 shows an overview of the library scenario. Two mobile devices interact with the library Information System. Both clients are

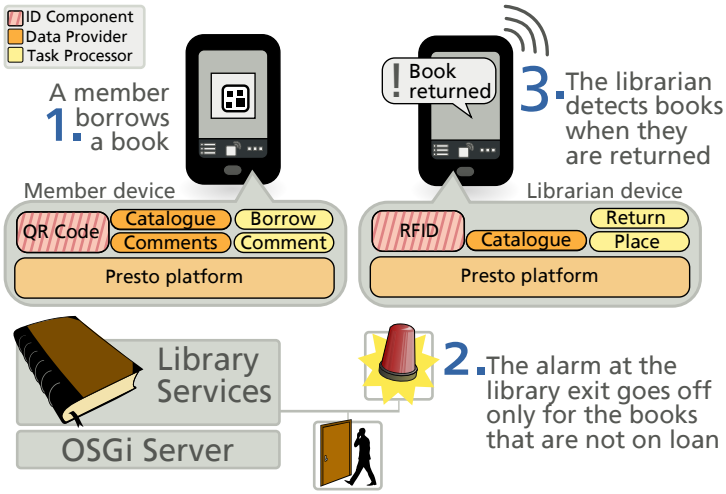


Figure 5.10: Components for the Smart Library mobile clients

based on the Presto platform but they extend it with specific plug-ins. These plug-ins support the specific perspective of each participant in the process. The library services are based on OSGi and we make use of KNX devices to support the detection of books at the exit of the library and the alarm system in a scale environment. An HTC Magic was used as the client device. This is an Android-based mobile device that provides access to WiFi, barcode decoding capabilities using the camera, support for user gesture detection on the touch screen and different embedded sensors (digital compass, GPS, accelerometer, etc.).

The **extensibility of the platform** enabled the creation of custom clients for the different roles in the process just by combining plug-ins. The mobile client for the library members includes a *Task Processor* plug-in for each of the tasks that members perform in the process (e.g., borrowing a book and commenting on books) as well as an *Identification Component* plug-in to support QR Code-based identification. QR Code detection was based on the barcode reader component provided by the ZXing project. The ZXing library provides decoding capabilities for different kinds of one-dimension barcodes and QR Codes. For the QR Code generation the Google chart API was used. This API provides

support for the generation of QR Code tags. In order to make the recognition as fluent as possible, we selected the "H" error correction level, which is the highest correction level defined by the QR Code standard (it allows the recognition of a code that is damaged up to a 30%).

Librarians make use of the *Identification Component* that supports RFID technology to support a different set of tasks (e.g., notify the return of books and place them on their shelves). Due to the lack of RFID capabilities of the mobile device used, Wizard of Oz techniques (Dahlbäck et al., 1993) were applied to simulate the reading of RFID tags for librarians. An operator provided the identification events when a mobile device touched a specific item. If an Android-based mobile device with RFID capabilities were available, RFID capabilities could be leveraged just by replacing the *Identification Component* without affecting the rest of the system.

The book lending process is complemented by services enabling members to share book-related comments. While book lending concerns books at the item level, comments are not specific to the book itself but rather to its content. Thus, two different *Data Provider* plugins were defined to represent these views in order to offer **flexibility in the physical-virtual linkage**.

Figure 5.11 illustrates the interaction with a physical book for a scenario based on the loan process of books in a library. First, Presto shows the tasks that the user can initiate and complete depending on the available *Task Processors*. This list can be filtered using different technologies. In the example, three *Identification Components* are available. Each *Identification Component* supports a different physical mobile interaction (Rukzio et al., 2006) by means of a different technology: QRCode for pointing, RFID for scanning, and a virtual keyboard for user-mediated interaction. Once a physical element is detected, the user can select a perspective (e.g., "Smart Library", "Amazon Store", etc.) to access the appropriate functionality (e.g., borrow the book) according to the perspective. Later, the user is notified when the book is returned (the librarian performs the *return book* task).

Figure 5.11: Book loan process supported by Presto

5.3 Automating the development process

One of the main reasons for following the current architectural process is that it is focused on automation. Business process requirements change quite often, and systems need to evolve accordingly. By automating the development process, the system can adapt to requirement changes without losing quality. With the adequate tool support, changes in requirements can be mapped automatically to the particular technology the system relies on, facilitating its evolution.

Provided that modeling concepts are defined in a precise way, models can be transformed automatically into new models or code by means of model transformation techniques. This enables automation in system development since software artifacts can be derived in a systematic way. Many technologies and standards give support to this development paradigm. The Object Management Group (OMG) defined Model Driven Architecture (MDA) (Miller & Mukerji, 2003) to provide support to these ideas with standards for metamodeling and the definition of model transformations. Either following MDA or any other paradigm based on MDE ideas, software development can be improved by the raise in the abstraction level that the use of models provides.

Figure 5.12 illustrates the tasks that have been automated for the

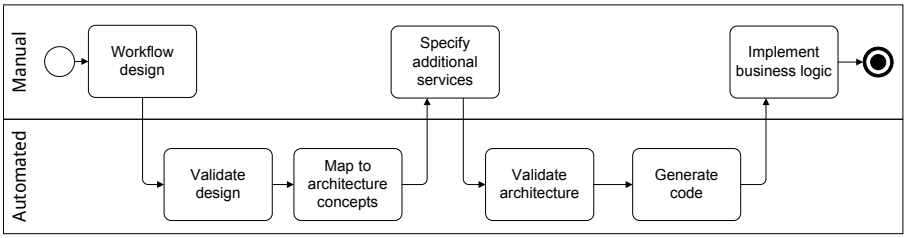


Figure 5.12: Automated and manual steps in the model-driven process

development of physical mobile workflows. The model-driven process defined begins with the specification of the workflow by means of the design concepts defined in Chapter 4. Then, this system specification is automatically validated and transformed into the Presto architecture concepts. Designers can modify the obtained model to specify additional digital services that are required to support a certain task in the workflow. The architecture model is validated and code generation techniques are applied to obtain an initial implementation of the system for a given platform. Finally, the implementation is completed to include business logic and link services with the external entities that provide them.

The following sections describe the way in which the abstraction level has been raised by means of MDE techniques. Presto concepts are formalized in Section 5.3.1 to precisely define the architecture components and the way they can be related to each other. Section 5.3.2 introduces the code generation techniques applied to translate Presto concepts into a software solution. Section 5.3.3 formalizes the design concepts and defines a mapping between them and Presto concepts. Finally, Section 5.3.4 provides an overview of the model validation techniques applied in order to avoid inconsistencies in the models defined during development.

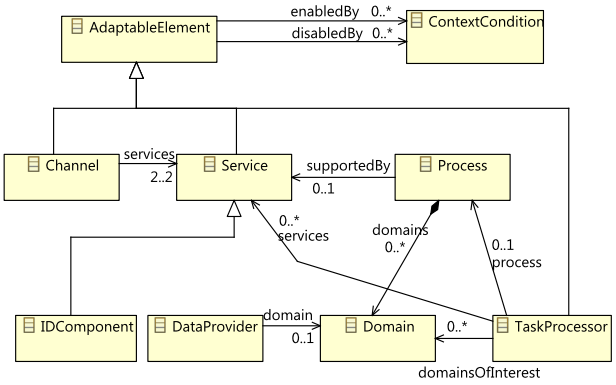


Figure 5.13: Excerpt of the metamodel for Presto architecture

5.3.1 Architecture Metamodel

MDE proposes the use of metamodels to formalize concepts and their relationships. A metamodel defines the constructs that can be used to describe systems and the ways in which these concepts can be combined. Using a metamodel, system descriptions become unambiguous at least at syntactic level. This makes the descriptions machine-processable.

The **architecture metamodel** of Presto captures the concepts defined in Section 5.2.2 such as the *Identification Component* or the *Task Processing Component*, and the constraints for their composition. Figure 5.13 shows a diagram for the architecture metamodel. The complete metamodel can be found at the Appendix A.

The metamodel includes the definition of the pluggable components (*TaskProcessor*, *IDComponent*, and *DataProvider*), some concepts to group them (*Domain* and *Process*) and some concepts to describe adaptation of the workflow (*AdaptableElement* and *ContextCondition*). The pluggable components represent the basic building blocks of any Presto-based system. The metaelements defined allow designers to specify the properties that characterize each component as it was introduced in Section 5.2.3. For example, *Task Processors* can be declared as *initiators*.

Metaelements such as *Domain* and *Process* have not a direct representation in the final software system. The domain represents a particular group of physical elements that is provided by one or more *Data Provider* components. The *Process* element was introduced to organize *Task Processors* according to their support to a specific workflow. This is also expressed as plug-in properties in the corresponding elements. For example, Task Processor plug-ins can indicate the supported process as a text string. Representing these concepts at the modeling level provides benefits in terms of knowledge centralization. Changing the name of a given process is performed in a dingle place instead of modifying all the *Task Processors* associated to it.

Presto systems normally access different kind of services to provide functionality to the user. Services are connected by means of *Channel* elements and can be adapted according to some context conditions. These elements are defined to support the approach introduced in Chapter 6 to adapt the workflow at run-time. We have defined the *Service* metaelement to represent these services. We defined the *Identification Component* as a particular kind of Service since it can make use of other services to achieve identification tasks. The *Identification Component* follows the composite pattern. Thus, *Identification Components* can be used by other *Identification Components*. This is useful for handling containment relationships (e.g., books contained in a return box that can also be identified).

By using the constructs defined in the metamodel, different Presto-based systems can be modeled. Since the concepts used for this description have been formalized, the resulting models can be processed automatically by different MDE tools. For the definition of the architecture metamodel, concepts have been formalized using Ecore. Ecore, part of the Eclipse Modeling Framework[3] (EMF), is a language targeted at the definition of metamodels with precise semantics. EMF provides tool support for the definition of metamodels and the edition of models.

We have defined the Presto metamodel as the first step towards the automation of the development process. The use of EMF enables metamodels to be machine-processable. This allows other EMF-compliant

[3]http://www.eclipse.org/modeling/emf/

tools to manipulate Presto specifications with different purposes (check properties, define graphical editors for the specification, etc.). Thus, Presto becomes also **an extensible platform at tool-level**. In particular, we make use of code generation techniques in this work to automate the technology mapping as it is illustrated in the following sections.

5.3.2 Glue Code Generation

The technology mapping defined in Section 5.2.4 involves several repetitive tasks. For our target technology, the definition of each *Identification Component* requires the definition of two *Android Activities* to interact with the user, an *Android Service* that implements the code to process sensing and/or minting intents, and the communication mechanisms to tight this components together according to the execution strategy defined by Presto. This boilerplate code can be automatically generated by the information captured in system models. In this way, developers can focus on implementing only relevant business-logic.

We provide code generation capabilities for the mapping described in the present work. This mapping considers Android as the target technology, but the approach followed allows developers to define different mappings to target other technological platforms. As it is illustrated in Fig. 5.14, the same Presto-based specification of a system can be used as the target for different platforms depending on the mapping rules applied.

From the description of a system based on Presto metamodel, source code can be generated with model-to-text transformation techniques. Model-to-text generation tools provide mechanisms to traverse models and generate the code associated with them. We applied model-to-text transformations to formalize the mapping defined in Section 5.2.4. Glue code generation has been implemented using XPand templates from the Model-to-Text (M2T) project[4], which is part of the Eclipse Modeling Project. The application of templates to models is similar to the way templates are used to generate dynamic web pages in the web application development area. Model elements can be iterated and pieces of code can be produced instantiating them with values obtained

[4]http://www.eclipse.org/modeling/m2t/

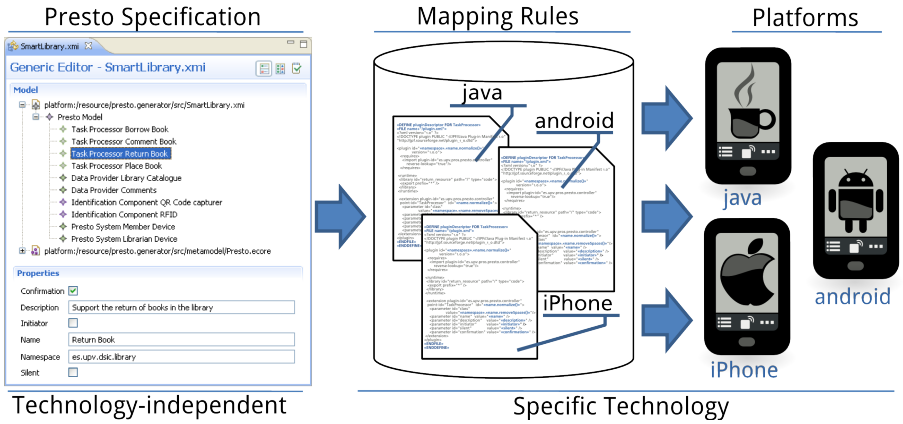


Figure 5.14: Glue code generation strategy.

from the model. XPand is a statically-typed template language with several features that simplify the code generation:

Polymorphic template invocation. Inheritance relationships in the source metamodels can be leveraged when templates are defined. Given a set of modeling elements that are involved in inheritance hierarchy, specific behaviors can be easily defined for the different sub-types. When multiple templates are available for an element, the code generation engine applies the template variant that is more specific to the current kind of element.

Functional extensions. Metamodels can be extended in a non obtrusive manner to obtain derived information easily. This information is accessed as if it were part of the metamodel. However, these extensions do not affect the metamodel since they are only accessible during the transformation. Thus, generation rules are more readable and less dependent on the metamodel structure, which improves the generator maintenance.

A flexible type system abstraction. XPand provides support for some built-in types including simple types (String, Boolean, Integer,

and Real) and collections (List and Set). In addition to built-in types, the type system can be extended with the concepts defined in the different metamodels. Thus, if Presto metamodel is imported we can use a list of a *Task Processors* as if these elements were part of the type system.

Model transformation and validation facilities. In order to ensure that the models that are used for the generation meet certain conditions, they can be analyzed prior to the transformation is applied. By validating the input, we can ensure that the generator does not find unexpected information (e.g., components with the same name that would lead to a nameclash when code is generated). Furthermore, facilities are provided to transform these models in order to fix the problems detected.

Listing 5.2: Excerpt of the code generation template that produces the Android Manifest file.

```
« DEFINE manifest FOR PrestoSystem »
« FILE name+"/AndroidManifest.xml" »
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res
    /android"
      package="« domain.name »"
      android:versionCode="1"
      android:versionName="1.0">
    <application android:icon="@drawable/icon" android:
        label="@string/app_name">
    « EXPAND manifest FOREACH getComponents() »
</application>
    <uses-sdk android:minSdkVersion="3" />
</manifest>
« ENDFILE »
« ENDDEFINE »
```

The listing 5.2 shows the code of one of the templates that is used for generating the *Android Manifest*. The previous excerpt of the transformation declares the *manifest* rule by means of the *DEFINE* keyword. The *manifest* transformation rule is a polymorphic rule that it is used to generate the fragment of the *Android Manifest* that is associated to

each Presto component. In the example, we show the rule that is applied to the whole Presto system. The rule named *manifest* is defined for the *PrestoSystem* element of the Presto metamodel. Generation rules control the creation of new files (e.g., source code, configuration files, resource descriptions, etc.) and the generation of their corresponding content. The *FILE* statement defines the output file for the code generation (in the example, an *AndroidManifest.xml* file is generated into a folder named after the *PrestoSystem*).

The rest of the rule is a code template with static and dynamic parts. The static pats of code are transferred to the generated code directly. In the example template, the static parts represent aspects that are common to any Android manifest, such as the XML header or the application declaration. The dynamic code is calculated for each instance to which the rule is applied. Dynamic expressions (defined between angle quotes) are used for capturing the required information and expressing it according to the target technology. For example, the package name for the *Android Manifest* is obtained from the *domain* attribute defined for the Presto application (e.g., *es.upv.pros.smartlib*).

For the generation of the rest of the *Android Manifest* the *manifest* rule is applied to the different components (see the *EXPAND* command) in a polymorphic manner. In this way, each component contributes to the manifest as the technology mapping of Section 5.2.4 indicates. Listing 5.3 illustrates the definition of the manifest rule when it is applied to *Task Processor* components.

Listing 5.3: Generation rule that produces the Android Manifest fragment corresponding to Task Processors.

```
« DEFINE manifest FOR TaskProcessor »
<activity android:name=".« name.normalize() »" android:
    label="@string/app_name">
  <intent-filter android:label="« name »">
    <action android:name="« task ».« process.name.normalize
        () »">
    </action>
    <category android:name="android.intent.category.DEFAULT
        ">
    </category>
  </intent-filter>
  <intent-filter android:label="task">
```

```
      <action android:name="es.upv.pros.presto.TASK »"></
          action>
       « IF initiator »
         <category android:name="es.upv.pros.presto.LAUNCHER"
             ></category>
       « ELSE »
         <category android:name="android.intent.category.
             DEFAULT"></category>
       « ENDIF »
    </intent-filter>
  </activity>
  « ENDDEFINE »
```

The rule defined in the previous example declares the activity in the manifest and their intent filters according to the mapping defined for *Task Processors* in Section 5.2.4. Two intent filters are defined for the activity. The first intent filter is named after the Task Processor (e.g., "Borrow Book") and it represents the main action for the task. The second intent filter marks the current Android activity as a *Task Processor*. This intent is defined to be part of either the default category or the *es.upv.pros.presto.LAUNCHER* category, depending on the initiator attribute of the *Task Processor* element. This determines whether the current activity can be launched by the user directly or it is executed in response to a pending action. The template of the previous example makes use of conditional statements (see the *IF*, *ELSE*, *ENDIF* instructions) to determine the category for the generated intent.

The code generation support in Presto automates the definition of the *Android Manifest* and the Java classes that are required for the implementation of the different components according to the mapping defined in Section 5.2.4. Intent processing code and method declaration is also generated. Although full code generation is not provided for component implementation, the provided code skeletons let developers focus on the implementation of the business-logic behavior, avoiding to deal with particular details of the target technology. Since the Android specific artifacts are generated, the use of Android application framework is made transparent to the developer, who only has to deal with Java programming.

5.3.3 Using design concepts for development

According to the architecture process followed in this work, once the architecture concepts are mapped to a specific technology, another abstraction step is required to simplify the use of the architecture. The architecture should be programmable by using concepts that are close to the problem domain. In this way, experts in the domain can understand, validate, modify, and define the software solutions by dealing with concepts that are familiar to them.

According to the architectural process defined by Völter, the next step is to define a "DSL-based programming model". A Domain Specific Language (DSL) describe concisely problems of a certain domain (van Deursen et al., 2000). The use of DSLs involves a tradeoff: they reduce their application scope to facilitate their use in these particular situations. Some examples are WS-BPEL for Web Service orchestration, SQL to query data or the Graphviz language for defining graphs.

The modeling primitives introduced for the design of a physical mobile workflow are used to define our DSL. This requires (1) the formalization of the concepts defined in the design method and (2) the mapping between these concepts and the architecture components of Presto. In order to formalize the concepts of the design method into modeling primitives that are machine-processable, a metamodel named Parkour is defined. This metamodel formalizes the concepts defined in our design method. Then, model transformation techniques are used for the generation of an initial model of the architecture for the system.

The Parkour metamodel

The Parkour metamodel defines in a formal manner the modeling concepts used for capturing the physical mobile workflow requirements according to the design method presented in Chapter 4. For the definition of the Parkour metamodel we made use of similar techniques and technologies as the ones used for the definition of Presto architecture metamodel. The main difference with the definition of the previous metamodel is that some of the concepts in the Parkour metamodel were extensions on modeling elements from the BPMN metamodel. The

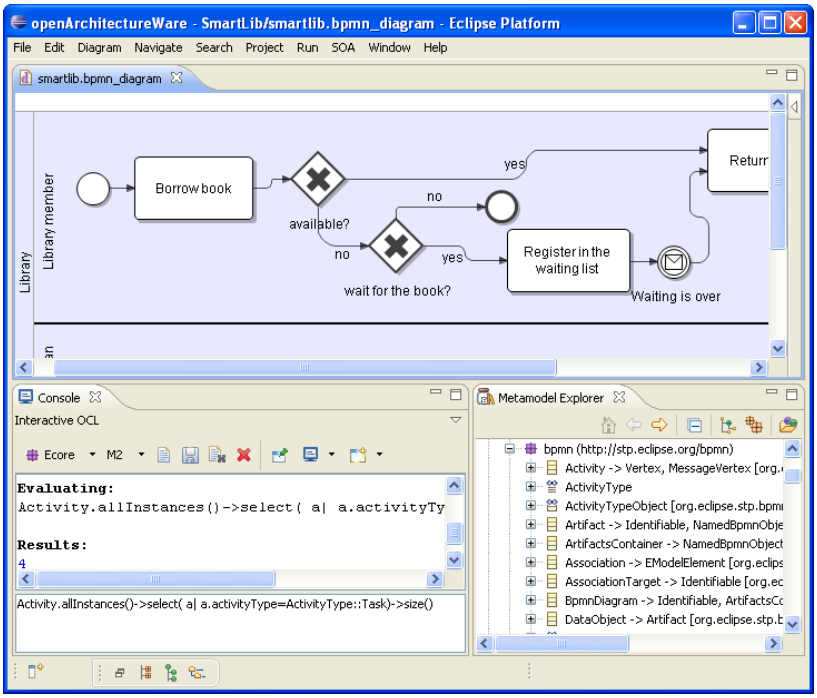


Figure 5.15: Eclipse-based tools for editing and querying business process models

Parkour metamodel integrates the BPMN metamodel in a non-intrusive manner. Since the BPMN metamodel is used as-is, we can make use of the existing tools for business process definition and only complement the required information for the extended elements.

For the specification of business process models we took the BPMN metamodel defined in the SOA Tools Platform Project (STP) as a basis. The STP metamodel has a very complete support for the specification covering almost all BPMN shapes, connections and markers except the layouts and appearance of the lanes inside a pool and the group-artifact. The STP project provides a functional editor for BPMN diagrams (see Fig. 5.15, top) which is integrated with other modeling tools. For example, Object Constraint Language (OCL) can be used to define queried over the model (see Fig. 5.15, bottom-left).

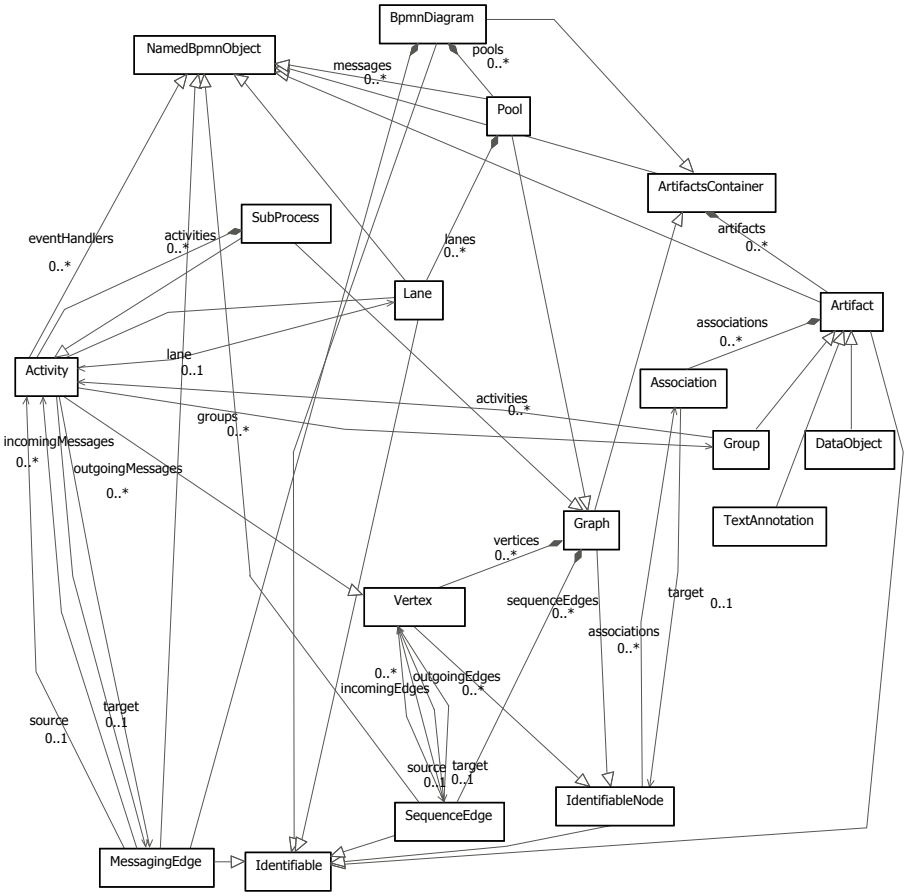


Figure 5.16: BPMN metamodel used as a basis for the business process diagram

The BPMN support in our approach relies on the BPMN metamodel defined by STP. Figure 5.16 shows the different metaclasses in the BPMN metamodel and the relationships among them. The metaclass *BpmnDiagram* normally is used as the root element of BPMN models. We can distinguish four kinds of metaelements in the metamodel. Some metaclasses are used to represent the underlying graph structure of BPMN diagrams such as *Vertex* and *SequenceEdge*. Other

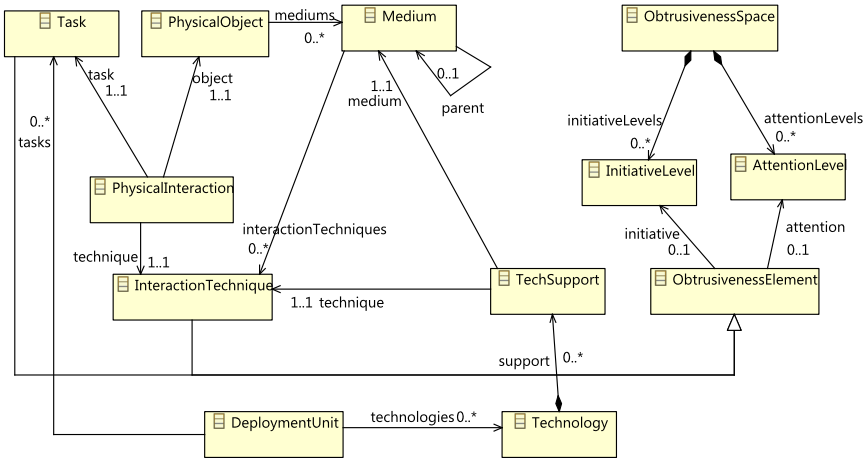


Figure 5.17: Excerpt from the Parkour metamodel

metaclasses are used for the purpose of classification. These classes are not instantiated by final models but used for defining an inheritance hierarchy. This is the case of *NamedBpmnObject*, *Identifiable* and *IdentifiableNode*. Other elements are designed to support the main primitives of BPMN (see Section 2.1.1 for more detail) such is it the case of *Activity*, *Subprocess*, *Lane* and *Pool*. The *Activity* metaelement represents different atomic elements from BPMN such as a task, a gateway or an event depending on its type. Finally, different kinds of artifacts can be associated with model elements as it is the case of *DataObject* as a particular subtype of *Artifact*.

The Parkour metamodel extends *Activity* and *DataObject* from the BPMN metamodel in order to specify the particularities of the physical-virtual linkage. The Parkour metamodel was defined in a separate metamodel in order not to couple the proposal with a specific workflow modeling language. We follow the same approach for the non-intrusive extension of the metamodel we applied in previous works dealing with the BPMN metamodel (Torres et al., 2007c).

Figure 5.17 illustrates an excerpt of the Parkour metamodel that

formalizes the concepts used in our design method. The Parkour metamodel is based on the concepts defined in Chapter 4 and defines how they can be composed to define a system specification. For example, our design method defines a hierarchy of mediums to analyze the possible interaction mechanisms. In the figure the *Medium* concept is represented and a relationship is established with the *InteractionTechnique* element. The *Medium* concept is defined with a reflexive *parent* relationship to allow a hierarchy of inheritance when mediums are defined. However, some of the rules that determine how to build a Parkour model cannot be expressed in the diagram. For example, the medium hierarchy must be acyclic since it represents a taxonomy. In order to enforce this constraint we made use of the modeling validation facilities provided by the Eclipse Modeling tools. In this case we make use of Check constraints that can be used easily in conjunction with XPand templates.

Listing 5.4: Check constraint example

```
import es::upv::pros::parkour
extension parkour::Extensions;

context Medium ERROR "The medium hierarchy must be acyclic.
     Medium '"+ name + "' cannot be set as a child of " +
    parent.name +".":
  this.isAcyclic();
```

The code listing 5.4 shows the definition of a constraint. First it is defined the kind of elements to which the constraint applies by means of the *context* keyword. In addition, the severity (e.g., *error* or *warning*) and the message to be provided to the user when the constraint fails is also defined. Finally, the expression to be validated is specified. The expression in the example makes use of the *isAcyclic* auxiliary function that is defined as an extension of the metamodel. The *extension* statement in the example is in charge of importing the extension of the metamodel that implements the *isAcyclic* operation.

Listing 5.5: Excerpt from the metamodel extension file

```
import parkour;
Boolean isAcyclic(Medium this):
(this.parent != null && this.parent!=this && this.parent.
    isAcyclic({this})) || this.parent==null;
```

```
Boolean isAcyclic(Medium this,Collection[Medium] found) :
  if found.contains(this) then false
  else if this.parent==null then true
  else this.parent.isAcyclic(found.add(this));
```

The *isAcyclic* operation traverses the parent relationship to find whether there are any ancestor that is found more than once. This extension is shown in the listing 5.5. Two overloaded operations are defined to support the search for cycles. The fist represents the base case for the first node explored. This function analyzes the current node and relies on the other function to analyze their ancestors. The second one takes into account the previous results and propagates the execution to the next ancestor until some child element is found or there are not more ancestors. If there is a repeated element, a cycle has been found in the inheritance hierarchy.

More detail on the Parkour metamodel is provided in the Appendix A. The formalization of the concepts defined in our design method allows to face the development of physical mobile workflows from a higher abstraction level. We can make use of modeling tools to define valid workflow specifications according to our method. The following section defines the mechanisms to translate these specifications into a model based on Presto architectural concepts.

From Parkour to Presto

With the definition of the design method concepts in a metamodel, we can express the problem of obtaining a Presto-based solution as the mapping between the Parkour and Presto metamodels. This section provides a mapping between both metamodels that produces a preliminary architecture model. This Presto model contains the basic components that are required to support the workflow that has been described in the Parkour model. The architecture model can be refined and used as an input for code generation techniques in order to obtain a final software solution.

The mapping between the Parkour metamodel and the Presto metamodel is defined by means of a model-to-model transformation. Differ-

ent model-to-model transformation languages exist such as QVT (OMG, 2005), ATL (Jouault & Kurtev, 2006) or RubyTL (Cuadrado & Molina, 2007) . For this transformation Atlas Transformation Language (ATL) was used. ATL is a declarative language with Eclipse-based tools support for the edition and execution of model transformations.

In contrast to the use of general purpose languages for the manipulation of models, model-to-model transformation languages offer different advantages for this purpose:

Domain specific concepts. Transformation languages are defined with the purpose of manipulating models in mind. They provide primitives that fit the need for querying and creating model elements. Most of the model-to-model languages are based on OCL for navigating through models. In the case of ATL, rules can be defined in order to express a mapping between elements of input and output metamodels.

Dependency control. The model resulting from a transformation must be conformant to the output metamodel. This imposes many constraints in terms of the order in which each element can be created. For example, if we are creating a binary association, the elements that are connected by the association must already exist when the association between them is created. If transformations are implemented by means of general-purpose programming languages, the existing dependencies for model creation must be taken into account. Languages such as ATL take care of these dependencies automatically. In ATL, transformation rules are defined regardless of the application order. In order to support this behavior, ATL forbids the input models to be written and the output models to be read. In this way, transformation rules are free from side-effects.

Traceability support. The mapping between elements usually requires information about how other elements were mapped. Model transformation languages allow to be aware of the origin for the generated elements. For example we can determine which Android activity is created as a result of a *Task Processor* component and

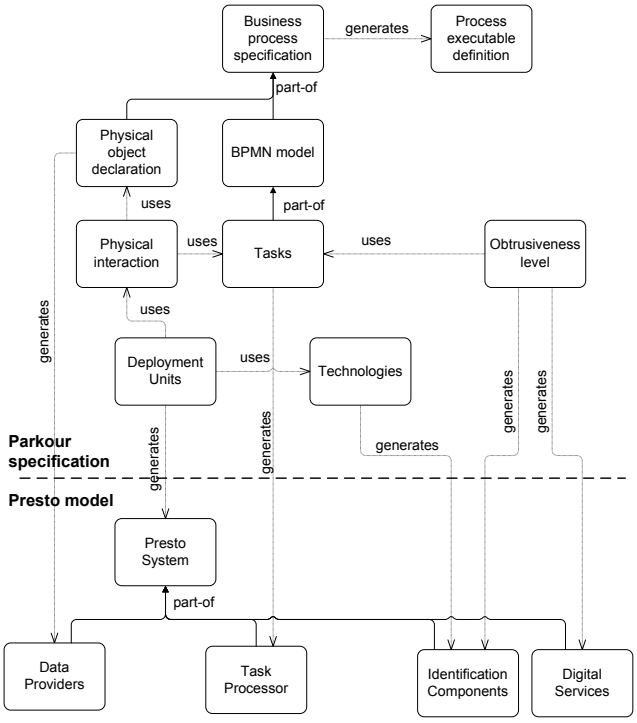


Figure 5.18: Mapping between the design concepts and the architecture components

which ones are created as a result of an Identification Component.

Flexible type system. As it happens with model-to-text transformations, we can extend the type system with metamodels and create virtual extensions of the metamodels to simplify the transformation process. In the case of ATL, these extensions are named *helpers*.

The overall strategy for the mapping is illustrated in Fig. 5.18 by means of a megamodel (Favre, 2006) (i.e. a model that represents modelinging languages and their relationships). The figure shows the relationship between some of the modeling concepts used in this work in terms of dependencies (see the *uses* relationships), inclusion (see the

part-of relationships) and generation(see the *generates* relationship). The mapping is an undirectional mapping from Parkour concepts to Presto concepts. The mapping illustrated is implemented by means of two model-to-model transformations. The first transformation generates an executable version of the business process diagram. This transformation takes a BPMN definition and generates an executable WS-BPEL specification that interacts with the *Task Manager* component of the Presto architecture. The other transformation is in charge of generating the different model elements that form a Presto-based system.

WS-BPEL is a common choice for the implementation of BPMN models. Although the BPMN specification defines a mapping between BPMN and WS-BPEL some details are lacking for obtaining an executable description since BPMN is a more abstract language. Different proposals exist for transforming BPMN into WS-BPEL. These proposals embed certain design decisions in the transformation in order to automate the mapping. We make use of the approach used to develop business process-supporting web applications in a previous work (Torres et al., 2007c). This mapping generates a WS-BPEL specification that delegates the tasks that require user participation to a *Task Manager* component. In the original approach, these tasks are handled by the user through a web interface while in our approach we are using Presto to enable the completion of these tasks by means of the interaction between the user and the physical environment. The use of business process execution engines with BPMN support such as Intalio Business Process Management Suite is a valid alternative to the previous transformation.

In order to complete the transformation process another transformation is defined. This transformation describes the Presto-based system that supports the user participation in the workflow. The mapping for obtaining the different Presto elements is described below:

Presto System. Different systems are normally involved in a business process. A *Presto System* is defined for each *Deployment Unit* that is detected during design. Each unit is implemented in a different device and supports a particular set of tasks. Each Presto system has its own *Controller Component* and *Task Manager*. The

pluggable components can be obtained from the Parkour model. The *Presto System* is complemented with *Task Processors*, *Data Providers*, *Identification Components* and *Services* according to its participation in the workflow as it is indicated below.

Task Processor. Each task in the BPMN diagram is supported by means of a *Task Processor*. The tasks that define the beginning of a business process in the BPMN diagram result in *Task Processors* with the *initiator* attribute set to *true*. The obtrusiveness level for a task also determines the *Task Processor* properties. If the task takes place at the background of user attention, the *Task Processor* is qualified as *silent*.

Data Provider. In order to determine the *Data Providers* required, the physical elements that are involved in each deployment unit are considered. This is calculated by obtaining the tasks supported by the deployment unit and the different objects that participate in these tasks. The *schema* attribute is initially set from the business process domain. Thus, all the physical elements in the process are initially considered to be from the same domain. Later, designers can modify this for specific objects in order to determine which elements are provided by which perspective.

Identification Component. *Identification Components* are defined according to the technologies used in each task. In addition, the interaction pattern used and the function (capturing or minting) are considered for the mapping. The direction of the connection for each physical element and the task is used to determine whether a minting or capturing functionality is required.

Services. The different components involved in a Presto system make use of auxiliary services. Most of these services cannot be anticipated from the information captured in the design models, but others can. In particular we define services for interacting with the user. The tasks that are located in the same region of the obtrusiveness space will use the same service for interacting with the user. Later, designers can provide each service with an appropriate functionality. For example the service used by non-disturbing

tasks can be associated with a notification system of the underlying architecture.

Listing 5.6: Transformation rule for generating the PrestoSystem element

```
module Parkour2Presto;
create OUT : Presto from IN : Parkour;

rule depUnit2PrestoSystem{
 from
   d: Parkour!DeploymentUnit
 to
   p:Presto!PrestoSystem (
    name <- d.name,
    taskProcessors <- d.tasks,
    dataProviders <- Parkour!PhysicalInteraction.
        allInstances()->select(x|d.tasks->includes(x.task))
        ->collect(e|e.object)->asSet(),
    idComponents <- d.technologies
   )
```

The code listing 5.6 shows the rule that defines the mapping between the *Deployment Unit* and the *PrestoSystem* metaelements. The example code includes the header definition of the ATL module. An ATL module (see the *module* keyword) defines a mapping between metamodels and can contain several transformation rules. In the example, Parkour metamodel is defined as the input and Presto metamodel is defined as the output metamodel. The rule *depUnit2PrestoSystem* transforms a *Deployment Unit* from the Parkour metamodel into a *PrestoSystem* element. When the transformation is applied to a Parkour-based model, the rule matches any *DeploymentUnit* element in the model and produces a *PrestoSystem* element according to the assignment instructions described below.

The assignment instructions determine how to generate the different attributes of an element in a model-to-model transformation. For simple types (e.g., the name attribute is copied), ATL copies the value to the resulting module. For complex types, ATL applies a transformation rule defined that matches this kind of elements (e.g., tasks will be transformed into *Task Processors* before they are assigned to the deployment

unit). ATL provides an OCL-like language to navigate elements. In the example, Data Providers are generated from the physical objects that are associated to the tasks supported by the current deployment unit.

The expression defined for the definition of the *Data Providers* navigates the *Physical Interaction* elements from the Parkour metamodel (`Parkour!PhysicalInteraction.allInstances()`). Then, the interactions that support tasks which are part of the current *Deployment Unit* d are selected (`select(x|d.tasks->includes(x.task))`). Finally, the Physical elements involved in thee interactions are gathered in a set to avoid duplicates (`collect(e|e.object)->asSet()`). actually this expression was defined in a helper but we have included it in the previous example for illustration purposes.

By applying the defined transformations, designers can obtain a preliminary version of the Presto system from the requirements captured at design-time. Although many components require further fine-tuning, the overall structure of the application is automatically obtained. In this way, designers are not required to define the Presto components and connect them together in a way that is consistent with the process defined. They can just focus on configuring the obtained services and declaring extra functionality that complements them.

The purpose of the transformation implemented was to illustrate the feasibility of facing the development of Presto-based systems from a higher abstraction level. The purpose was not to provide a full-featured modeling tool since it falls out of the scope of the present work. In order to provide complete tool support, many model-management aspects must be considered such as round-trip engineering support, end-user editors, and model versioning management.

5.3.4 Model-based validation

One of the most important use of models is to **reason about the system** they describe. Model-based verification can ensure that some aspects of the system are valid prior to its construction. This section introduces some of the validation capabilities provided during system specification.

The method proposed for the design of physical mobile workflows promotes separation of concerns. Different aspects are specified in different models in order to better handle complexity. However, this requires to apply validation techniques in order to guarantee that the different aspects specified are consistent.

We have implemented different validations that can be applied automatically to Presto and Parkour models. The validation mechanisms are supported by Eclipse-based tools. The properties to be checked in models are expressed by means of different constraints. The constraint specification is detailed in the Appendix A. This section provides an overview of the questions that can be answered thanks to these constraints:

Can the physical-virtual linkage described be supported in practice? Given a workflow specification it is essential to determine whether the technologies selected can support the physical-virtual linkage in the way it has been described. Designers can determine whether a particular interaction technique can be applied for a task by considering the technologies used for identifying the elements involved.

Designers can foresee the impact of removing, adding or changing a specific identification technology by simply modifying the model. For example, by describing a new identification technology in the medium hierarchy it is immediately detected which tasks can be supported by this technology in terms of interaction techniques and obtrusiveness levels.

Can the workflow be optimized? In addition to the detection of inconsistencies, it is important to detect situations of potential improvement. Constraints have been defined to detect minters that produce an identifier by using a technology that is not used by any other reader, or mediums that do not support any interaction technique.

The above examples do not necessarily constitute an inconsistency since (1) the minters can be used to produce identifiers that are consumed by third parties and (2) mediums can be used with

the purpose to organize the medium hierarchy. Nevertheless, is it good to make the designer to consider whether a specific modeling element (especially in non-common situation) is really useful.

Is the workflow suitable for generation? When abstract concepts are projected into concrete software components they must respect the platform rules. For example, redundant elements can result in a nameclash in the system (e.g., two Java classes in the same package with the same name). For each element in our models it is verified that the names given do no produces any problem when components are generated.

The possible conflicts among components depend on the specific elements considered. For example, two mediums with the same name cannot be defined. But *Task Processors* can exist with the same name provided that they are supporting different processes. This kind of validation is essential to make the underlying platform transparent to developers. Otherwise, designers are required to take some platform-specific rules into account when they are modeling.

These constraints have a twofold goal. On the one hand, they are used during design to provide modelers with immediate feedback about inconsistencies in their specifications. On the other hand, they simplify the definition of the model transformations since transformations can be specified by assuming that the models are consistent.

Figure 5.19 shows a Parkour instance model where the editor has detected some inconsistencies. The editor verifies the constraints each time the model is saved (or by user demand). The errors and warnings detected are integrated in the standard error view provided by Eclipse. In this case, the model contains one error and one warning. The error is produced because the deployment unit is supporting the *borrow book* task by means of RFID while the book was only tagged with a *paper-based identifier* in the example. Thus, the task cannot be supported with the technoloy defined.

It is the task of the designer to evaluate whether to solve this issue by changing the technology used in the task, to replace the current

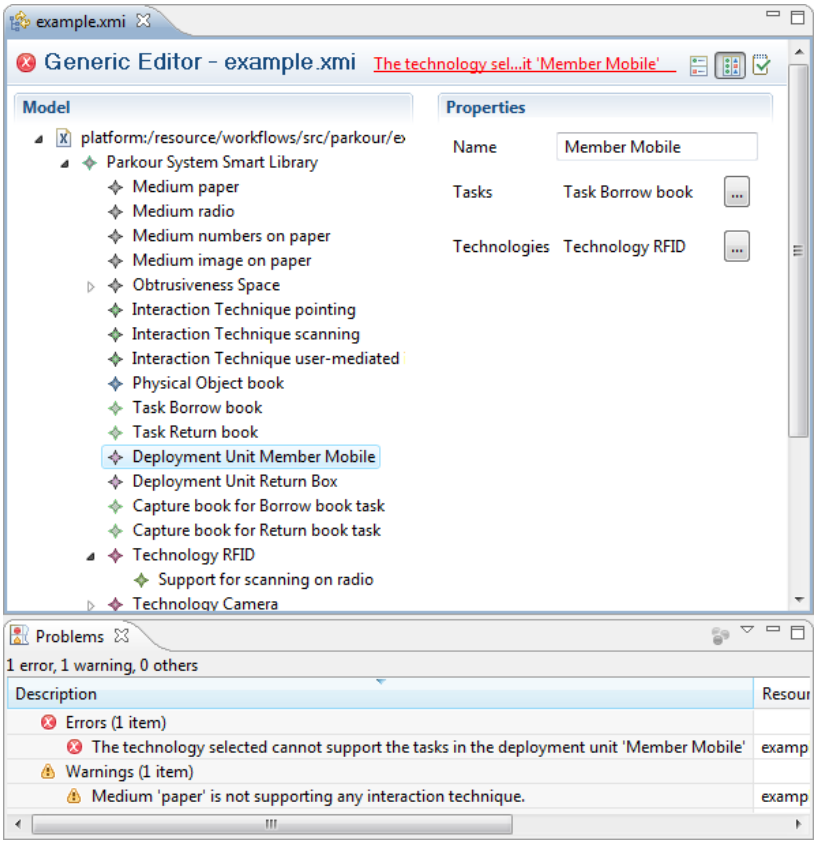


Figure 5.19: Validation for the deployment unit constraint

identifier by RFID, or use RFID in addition to the current technology. Validation support becomes also useful in this process since the implications of each technology can be explored in the model. Fig. 5.19 also shows a warning because the *paper* medium was not associated with any interaction technique. In this case, the specification is correct since the paper medium is used as an abstract medium in order to organize the medium hierarchy.

5.4 Conclusions

This chapter provides mechanisms for automating the development of physical mobile workflows. By applying MDE principles, the requirements captured in the design stage are transformed into a final software solution. In this way, developers do not have to deal with technological details of the target platform. In addition, the definition of the architecture at modeling level, allows the presented approach to be sustainable since it can support the evolution of the system to new technologies.

Much of the complexity in software development is due to the rules that the different computing frameworks require but are not enforced by the programming language in use. Platforms such as Android, OSGi, Java ME, Enterprise Java Beans, or Google Web Toolkit, make use of the Java language but they require to follow different programming models that are not always simple to understand. By avoiding developers to deal with these technological constrains they can focus on business logic. Hiding this complexity is the main goal of our approach for the automation of physical mobile workflows. For example, the developers completing the generated code in our approach do not require to know how the intent mechanisms works in Android, since the intent processing code is already generated for them.

In order to experiment the easy projection of the architecture concepts into different technologies we developed some prototypes based on JavaFX and Service Component Architecture. The concepts defined for Presto-based solutions were proven easy to express in these technologies.

Chapter 6

Adapting obtrusiveness at run-time

Imagine a future in which your fridge announces to you the recipes that can be prepared with the available goods, your TV tells you that your favorite program is beginning, the book you want to start reading is suggesting you try other similar books; and all of this is happening at the same time. Clearly living in such an environment on a daily basis would be annoying. On the other hand, if services behave in a completely automatic manner (without requiring human input or informing the user), users can feel that their environment is out of their control, which is also undesirable.

According to the Considerate Computing vision (Gibbs, 2004), user attention is a primary resource to be considered by software systems. Although any kind of software system can benefit from being aware of the user attention level, this is especially relevant in the Internet of Things (IoT) paradigm which promotes a natural interaction between the user and the environment (which is full of embedded services). The design method proposed in Chapter 4 takes the obtrusiveness aspect into consideration for the design of physical mobile workflows. In this

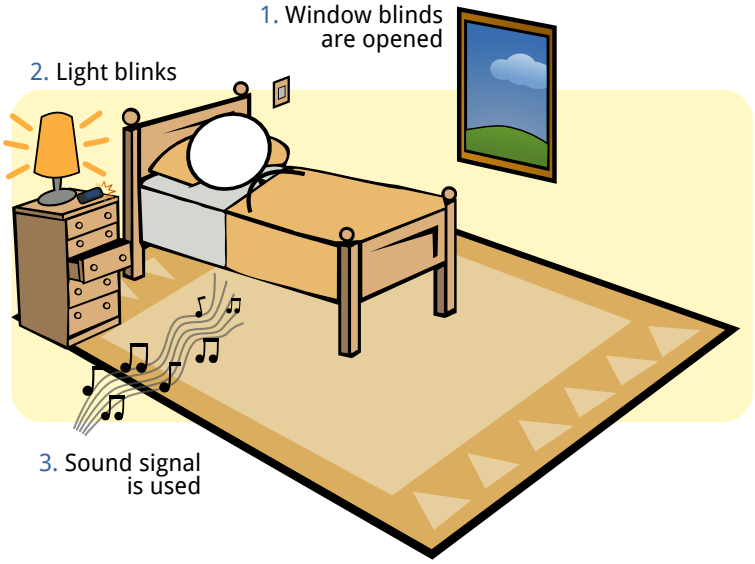


Figure 6.1: Example scenario where the obtrusiveness level is adjusted gradually

chapter we also consider this aspect for driving the workflow adaptation at run-time. In this way, the services that are involved in the workflow can be executed in a way that is considerate to the users according to their changing context.

In Chapter 4 we considered the implications of the obtrusiveness design for the development of physical mobile workflows. By considering this aspect at design-time, support can be provided for executing each task in the workflow at a different level of automation. As it has been argued, it is important to consider obtrusiveness aspects early in the development process since changes in the obtrusiveness level usually have a deep impact on the system architecture (they require actions such as object re-tagging, technology replacement, etc.). In our approach, the obtrusiveness requirements are considered when selecting the appropriate technologies and deployment options for a given business process. However, sometimes the obtrusiveness level cannot be completely determined at design-time and it must change during run-time. For some tasks the obtrusiveness level varies dynamically according to changes in

the context. For example, waking up a hotel guest can be done in a gradual manner[1] (see Fig. 6.1). To achieve this, the attention demand can be increased as the user remains in bed: first, the blinds are opened to increase the level of light in the room, a subtle blinking of lights is performed, music is played; and finally, a loud sound is produced by the guest's mobile device if the guest is still in bed.

In terms of obtrusiveness the previous scenario can be defined as a *weak up* activity that is performed in a proactive manner by the system. Each time it is performed, it requires a different degree of user attention. As a consequence, each time the task is executed a different set of resources are used to support the task at the appropriate obtrusiveness level. These resources provide the same functionality but at a different level of obtrusiveness.

This chapter describes how run-time reconfiguration techniques can be used to address this kind of scenarios. In particular, we have applied model-based reconfiguration techniques to allow the obtrusiveness level of a physical mobile workflow to be dynamically adapted to context conditions. Model-based techniques are applied to (1) describe by means of models the way in which the obtrusiveness level depends on context conditions and (2) propagate the changes in the obtrusiveness level into the system architecture components.

The remainder of the chapter is structured as follows. Section 6.1 introduces the technique used to describe the obtrusiveness adaptation for physical mobile workflows. Section 6.2 defines the mechanisms used to update the architecture according to what is described in the adaptation models. Section 6.3 describes the requirements that our technique has for the developers. Finally, Section 6.4 concludes the chapter.

6.1 Adapting the obtrusiveness level

As we have shown in our approach, thanks to the use of models, software for supporting physical mobile workflows can be defined by working with

[1] This scenario is based on techniques such as "dawn simulation" which are used for Seasonal Affective Disorder treatment, but also for increasing the comfort of waking up in general (Terman & Terman, 2006).

abstract concepts. We can deal in a centralized way with aspects such as obtrusiveness that are much harder to deal with in the final software system since they are spread across different parts of the code. In this chapter we make use of modeling techniques to drive the obtrusiveness adaptation of a process.

Changes in the obtrusiveness level of a workflow require the rapid reorganization of the system resources. For providing software systems with these capabilities in this work we have applied the principles of Autonomic Computing (Kephart & Chess, 2003). Autonomic Computing envisions computing environments that evolve without the need for human intervention. A system with autonomic capabilities installs, configures, tunes, and maintains its own components at runtime. In particular, we have provided physical mobile workflows with self-adapting capabilities. The adaptation capabilities provided in our approach are based on the models used at design for specifying obtrusiveness. In this way, the modeling effort made at design time is not only useful for producing the system but also provides a richer semantic base for autonomic behavior during execution.

Since the deployment of a system in the IoT requires set-up actions that cannot be changed at run-time (such as labeling objects with a certain technology or installing readers in certain places) our approach for adaptation consists on selecting from the available resources the ones that provide an obtrusiveness level that is as close as possible to the one desired. When the obtrusiveness level varies the architecture is re-targeted to make use of these resources in an automated fashion. Our proposed approach has two main aspects:

Reuse of design knowledge to achieve adaptation. We reuse the knowledge captured at design to describe the evolution of the system in terms of obtrusiveness. In response to changes in the context, the system itself can query these models to determine the necessary modifications to its architecture.

The conceptual framework that is used for describing obtrusiveness at design has been extended to allow the specification of adaptation rules that define the way in which obtrusiveness is affected by context changes.

Reuse of existing model-management technologies at runtime. In order to guide the adaptation of the system, we leverage models at runtime without modification (i.e., we keep the same model representation at runtime that we use at design time). This avoids the need for technological bridges, making it possible to apply the same technologies used at design-time to manipulate models at run-time.

We used the Model-based Reconfiguration Engine (MoRE) (Cetina et al., 2009a) to implement model-management operations. These operations determine how the system should evolve and the mechanisms for modifying the system architecture accordingly. Thus, systems use the knowledge captured by obtrusiveness models as if they were the policies that drive the system's autonomic evolution at run-time.

Different initiatives exist to make use of models for driving system adaptation. Several research efforts have successfully produced self-adapting systems using Petri-net-based (Zhang & Cheng, 2006) models and adaptation models (Morin et al., 2008). Due to the fact that these approaches generate decisions algorithmically and only on demand, they may give rise to undesirable behavior, impede users to understand system behaviors, and also, reduce the predictability of the system. Conversely, our approach takes the entire development process into account. Since the system specification is available at design time, we are able to conduct thorough analysis of the specifications for the purpose of validation. In this way, we are able to guarantee deterministic reconfigurations at runtime, which is essential for reliable systems.

The following subsections introduce the use of the obtrusiveness concept as the adaptation space for a system, the techniques used to manage the contextual information that can impact on the obtrusiveness level, and the modeling concepts used to describe the response to context changes when the system is adapted.

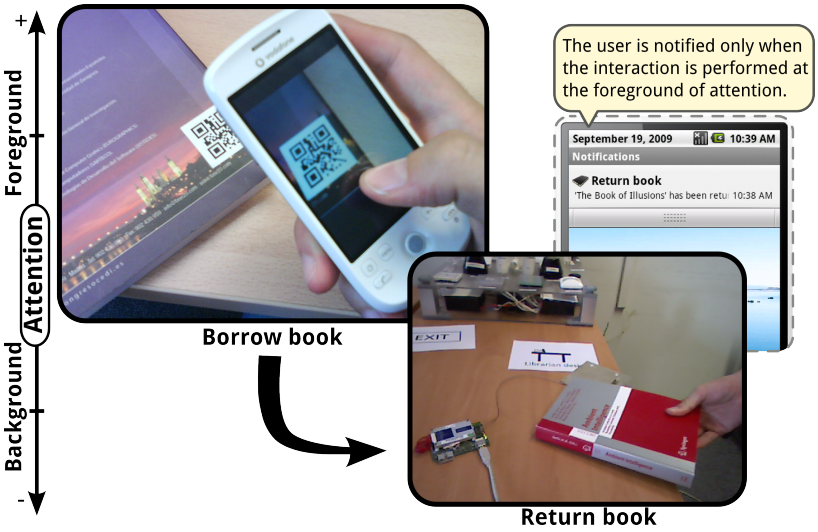


Figure 6.2: Example scenario where books are loaned and returned by means of different technologies at different obtrusiveness levels

6.1.1 The obtrusiveness adaptation space

Adaptive systems adjust their behavior to different situations. The adaptation space for a system is defined by all the possible system variants into which the system can evolve. The adaptation space is normally defined in terms of some relevant properties for the system. For example, some user interface adaptation approaches define their adaptation space in terms of the *environment* and the *platform* properties (Calvary et al., 2001). When this criteria is followed, different variants of the system are provided to the user depending on whether a desktop or a mobile device (platform) is used; or the system is used in a noisy or a quiet room (environment). When some of these conditions change (e.g., the user enters into a noisy environment), the appropriate system variant is provided to the user. Our approach defines the adaptation space for the system in terms of obtrusiveness.

The notion of obtrusiveness used for run-time adaptation in our approach is the implicit interaction framework that was used in the design method that was introduced in Chapter 4. The difference is that

in the case of adaptation, tasks in the workflow can be defined at two or more different obtrusiveness levels. Figure 6.2 shows a scenario based on the Smart Library case study, where different tasks are performed at different attention levels. The *borrow book* task is always performed at the same attention level, but the *return book* task can dynamically adapt the attention level. When the return of books is performed at the *foreground* of user attention the notification mechanism is used. When the return of books is performed at the *background* of user attention, the system silently registers the return of the loan without notifying the user.

The conceptual framework used in our approach considers *initiative* and *attention* as the orthogonal aspects that define obtrusiveness and defines mechanisms to move from one point of the space to another. The adaptation space is defined by formalizing the *initiative* and *attention* concepts as the axes that define a two-dimensional space. For supporting self-adaptation, we introduce an order in the values that define these factors. In this way, the system can compute the initiative and attention levels required for adapting the system that is at a specific obtrusiveness level to a new one.

The ordering in the obtrusiveness axes is defined as follows. On the one hand, the extreme values for the *attention* axis are *Background* and *Foreground*. Since this axe represents user attention demands, we could order these values as $Background < Foreground$ to indicate that *Foreground* interactions require more attention than *Background* interactions. On the other hand, the *initiative* axe is related to automation, so we consider that the *Reactive* value provides a lower degree of automation than the *Proactive* value (i.e., $Reactive < Proactive$).

Since we are using the same conceptual framework to define adaptation, the analysis performed at design to describe the obtrusiveness space can be reused for specifying adaptation. The only additional requirement is to define each level with a particular order and avoid intersections. This is not a strong limitation, and much of the divisions of the obtrusiveness space such as the ones used in this work already meet these requirements from the beginning. As a consequence of introducing this ordering, it is possible to express changes in the obtrusiveness level as increments and decrements in the different axes. This idea is

exploited in next sections.

6.1.2 Defining context conditions

Mobile information systems are characterized by frequent changes in the context of the user. Different kinds of context can be considered when a mobile system is developed (Krogstie et al., 2004): *spatio-temporal context*, *environment context*, *personal context*, *task context*, *social context*, and *information context*. The *spatio-temporal context* describes aspects such as time, location, direction and speed. The *environment context* captures the entities that surround the user. The *personal context* describes the user state by considering information such as pulse, blood pressure and weight, as well as mood, expertise, and preferences. The *task context* describes what the user is doing (either defined as explicit goals or the tasks and task breakdown structures). The *social context* describes information about friends, neighbors, coworkers and relatives. The *information context* describes the information space that is available at a given time.

According to the previous classification of contextual information, physical mobile workflows deal directly with information related to the *environmental context* and the *task context*. However, the adaptation of the obtrusiveness level can be the result of changes in other sources of contextual information. For example, the presence of a close friend (social context) in the nearby area (spatial context) can be notified to the user at a *foreground* level of attention compared to the presence of other people with a further distance in their social network which can be queried on demand by the user (i.e. reactive behavior is used). An ontology is used to capture and reason with a broad kind of context information.

In the Artificial Intelligence literature, an ontology is a formal, explicit description of concepts in a particular domain of discourse. It provides a vocabulary for representing domain knowledge and for describing specific situations in a domain. The ontology-based context model used in this work is based on the Web Ontology Language[2] (OWL). OWL is an ontology markup language that was defined by the World Wide

[2] http://www.w3.org/standards/techs/owl

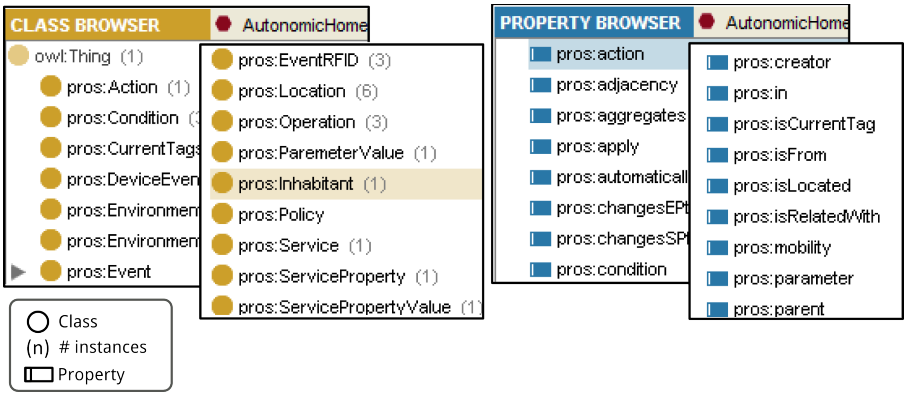


Figure 6.3: OWL Ontology for Smart Environments.

Web consortium (W3C) to support the Semantic Web approach that envisions a web where all their resources are precisely described in a machine-processable format.

An ontology-based approach for context modeling lets us describe context information semantically and share common understanding of the structure of contexts among users, devices, and services. The main benefit of this model is that it enables a formal analysis of the domain knowledge, such as performing context reasoning using first order logic.

The ontologies used in this work are described in OWL as a collection of triples based on the Resource Description Framework (RDF). Each statement is in the form of *(subject, predicate, object)*. The *subject* and *object* are the ontology objects or individuals and the predicate is a property relation defined by the ontology. For instance, *(John, Location, Garden)* means that John is located in the garden. Figure 6.3 shows the ontology defined to handle context information in a Smart Home environment. Different services are provided in the Smart Home of the example and some of them can be coordinated to support physical mobile workflows.

Information regarding the process execution state, the physical elements detected and the state of different devices and sensors around the user environment are represented in the ontology. For a detailed

description of this ontology see (Serral et al., 2008).

The context processing mechanism that is in charge of keeping contextual information consistent with the real world, is based on rules. These rules aggregate and filter low level information and generate meaningful events. For example, the detection of an RFID tag (*pros:RFIDEvent* class in Fig. 6.3, left) can trigger the change of the location information of another entity (*pros:isLocated* in Fig. 6.3, right). By aggregating and filtering context events we can obtain context events that are relevant for adaptation. The next section defines the mechanism used for adapting the obtrusiveness level in response to context changes.

6.1.3 Defining transitions

During execution workflow tasks are performed in only one of the obtrusiveness levels that are defined in the obtrusiveness space. Nevertheless, this obtrusiveness level can be modified in response to changes in the context information. Designers can define *transitions* that link context events with changes in the obtrusiveness level. Each transition is composed by a *condition* and an *action*. When a condition is fulfilled, the obtrusiveness level is modified by changing the attention level, the initiative level, or both, as defined by the *action*.

The transitions define the changes to be performed in the obtrusiveness space. More detail regarding the definition of conditions over the context information and the actions performed for adaptation is provided below.

Conditions on context information

When a task from a workflow can be performed at different obtrusiveness levels, it is important to consider the factors that determine such decision. Conditions are specified in order to indicate which context events require the system to behave at a specific obtrusiveness level. Conditions are expressed as queries for context information. In order to detect a change in the context information SPARQL is used.

SPARQL is a query language based on (triple) patterns. Using SPARQL the user may issue a query of the form *(John isA ?role)* where

?role denotes a variable. The query engine checks through the data and retrieves the value of *Librarian* as a possible value for *?role* which constitutes a possible answer to the query. The dataset may also contain the triple *(Jonh isA Person)* (e.g., as a result of an inference); in this case *Person* is also a possible value for *?role*. By providing several triple patterns, complex queries can be created and used by the application.

In order to allow the definition of transitions, the corresponding context information must be included in the context ontology. Information can be either provided as explicit data or as derived data. The inclusion of explicit data requires the creation of new classes and instances in the ontology. However, for the definition of derived information, derivation rules must be defined. Depending on the information management needs, designers can chose the most appropriate mechanism.

Adaptation actions

Adaptation actions produce an impact on the obtrusiveness level of the system. When a context condition of a given transition is fulfilled, the corresponding action is performed. An action can change the obtrusiveness level in either a relative or absolute manner. A *relative* action is specified as an increment or decrement of the obtrusiveness level for one or both axes (e.g., an increment of two levels for *attention* and a decrement of one level for *initiative*). An *absolute* action represents the transition to a particular value for one or both axes. Relative modifications are specified by indicating the *increment* either positive or negative while absolute actions are specified by indicating the *destination* level.

The use of relative or absolute actions depends on the specific semantics that designers are using. Nevertheless, the relative actions allow to specify changes in the obtrusiveness level that are sensitive to the current obtrusiveness level. In the wake-up example, we can define a *relative* action that produces an increment in the *attention* axis each time the hotel guest is required to wake up. If the workflow is defined in such a way that the *wake up* task is activated repeteadly at certain intervals until the guest finally gets up, services demanding a greater degree of user attention will be used each time.

In the case that multiple conditions are fulfilled at the same time, contradictory actions may be triggered, i.e., a movement in the obtrusiveness space performed in different (possibly opposite) directions. To resolve this conflict, actions are aggregated before they are applied. This process involves the following steps:

1. Absolute actions are expressed as relative actions. This is done by calculating the increment that is required to reach the desired destination from the current obtrusiveness level. For example, if the attention axis is divided in three parts and the current task is performed at the *Background* level, an action that requires attention to be at the *Foreground* level is represented as an increment of 2 units for this axis.

2. Actions are aggregated. The median is calculated for all the relative increments of the different actions for each axis to obtain an average increment. The rationale behind this is to obtain the average movement in the obtrusiveness space. We use the median instead of the mean in order to avoid extreme results to affect the changes in the obtrusiveness level.

3. The resulting action is applied. The current initiative and attention levels are modified according to the increment obtained by aggregating the different actions. In this way, only a single obtrusiveness change is propagated to the system architecture.

Once the transitions among certain obtrusiveness levels are defined, the obtrusiveness levels are mapped to the architecture components that support the underlying system. This is illustrated in the following section.

6.2 Reconfiguring architecture components

The set of transitions defined for the obtrusiveness space capture the criteria to be followed during adaptation. Once these criteria has been defined by means of models, we need to establish links between these models and the underlying architecture that supports the system. When

a change is produced in the obtrusiveness level, the architecture must be updated accordingly. In order to enable the obtrusiveness level of a system to be changed at run-time, we make use of the Model-based Reconfiguration Engine (MoRE).

MoRE is a reconfiguration engine that is based on the principles of Autonomic Computing. Designers can provide adaptation rules to MoRE in order to indicate when the architecture should be reconfigured. If there is a context change that triggers a rule, MoRE performs a set of changes to the architecture. Then, MoRE computes which components must be disabled and which ones must be enabled. Finally, it performs the corresponding actions to obtain the desired state. The reconfiguration is performed in a transactional manner, enabling the rollback of actions if the reconfiguration fails.

We chose MoRE since it is a generic engine that can be customized by means of models. This make it feasible to integrate it in our approach by leveraging existing models of the system. In order to use MoRE for a a particular application, designers must provide the adaptation rules and the architecture description by means of models. The reconfiguration infrastructure defined by MoRE and the technique used for defining the adaptation rules are described below.

6.2.1 Model-based reconfiguration

To achieve autonomic computing, MoRE is based on the IBM reference model for autonomic control (IBM, 2003). The overall reconfiguration steps performed by MoRE are outlined in Figure 6.4. A Context Monitor uses the run-time state as input to check context conditions. If any of these conditions is fulfilled, then MoRE queries the models at run-time about the necessary modifications to the architecture. Given the model response, MoRE elaborates *reconfiguration actions* which modify the components of the system architecture and maintain the consistency between the architecture models and the actual architecture components.

The above model-based version of IBM's reference model for autonomic control makes an intensive use of models. Context events and system architecture are represented by models. Context events are rep-

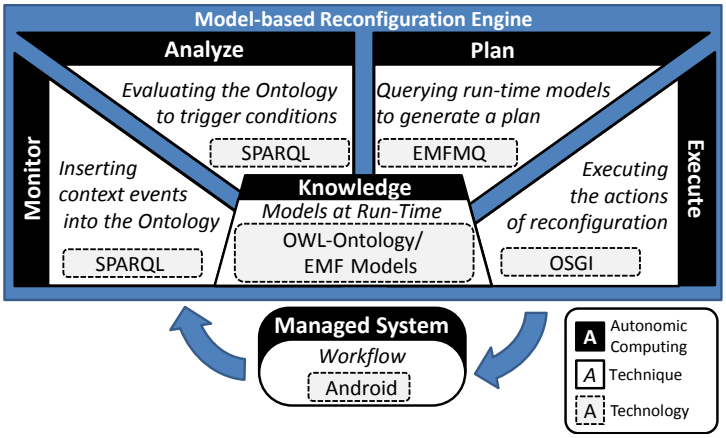


Figure 6.4: Components for supporting model-based reconfiguration

resented by means of OWL ontologies, and system architecture is also captured by means of models. For performing the system reconfiguration, information is extracted from these models. Different model query technologies are used at run-time by MoRE. MoRE uses SPARQL for OWL manipulation and Eclipse Model Query (EMFMQ) for variability model manipulation. SPARQL was already introduced for defining conditions on context information. EMFMQ provides an API to construct and execute query statements in a SQLlike fashion. These query statements can be used for discovering and modifying model elements that are based on the Eclipse Modeling Framework (EMF).

The reconfiguration of the system is performed by executing reconfiguration actions that deal with the activation/deactivation of components. The component life-cycle followed during reconfiguration is based on a Decentralized Control System Reconfiguration pattern (Gomaa & Hussein, 2007). This model defines different states for a component in order to activate and deactivate components gradually without breaking the transactions in which they are involved. The states defined are *Active*, *Passive*, *Quiescent*, *Passivating* and *Waiting*. Figure 6.5 show a state machine to represent the life-cycle of a component according to this pattern. By following this approach the following properties are achieved for the resulting architecture: (1) Non interference with those

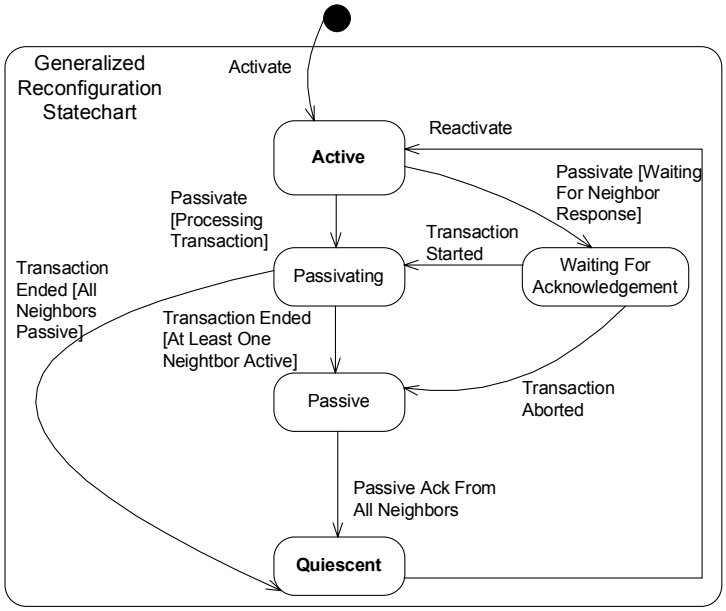


Figure 6.5: Reconfiguration state machine model (Gomaa & Hussein, 2007)

parts of the application that are not impacted by the reconfiguration, and (2) during reconfiguration, impacted components must complete their current computational activity before they can be reconfigured.

MoRE makes use of the OSGi framework for implementing the reconfiguration actions. The managed system is manipulated by MoRE in order to update its components according to the desired configuration. For the purpose of the present work, the managed system is a system that supports physical mobile workflows by following our approach and receives signals from MoRE when some components must be updated.

6.2.2 Reconfiguration policies specification

In order to allow a flexible reconfiguration, architecture models in MoRE are based on different components and communication channels that connect them. We classify these components into two categories: *Services* and *Resources*. Architectures following this pattern provide easy

reconfiguration since communication channels can be established dynamically between the components, and these components can dynamically appear or disappear from configurations (Hallsteinsen et al., 2008).

Figure 6.6 shows the part of the system architecture that supports the *wake up* and *prepare coffee* tasks from the example scenario. Services are represented by a circle, and resources are represented by a square. Finally, the channels among services and resources are represented by lines. The components that are required for supporting a task in the different obtrusiveness levels are indicated in the figure. For example, the *blinds* that regulate the light for the *wake up* task are used when the task is performed at the lowest level of the attention axis. The *wake-up* task is obviously performed in a proactive manner with regard to the *initiative* axis since the user is sleeping when this action is started. This mapping between the *blinds* and the obtrusiveness levels for the *wake-up* task can be expressed as $WakeUp(invisible, proactive)$, where the first coordinate refers to the *attention* and the second one to the *initiative* axis (numerical values have been used in Fig. 6.6 for the sake of brevity).

In order to specify which components and channels support a certain task for a given *obtrusiveness level*, the *Superimposition operator* ($\odot$) is defined. The Superimposition operator takes a task and an obtrusiveness level and returns the set of components and channels required for the task. Some examples of the relationship between the obtrusiveness level for the *prepare coffee* task and its mapping with system components (see Fig. 6.6) are as follows[3]:

$$\begin{aligned}\odot PrepareCoffee_{(aware,*)} = \{&notification\_service,\\ &mobile\_display, coffee\_preparation,\\ &coffee\_machine, mobile\_sound\}\\ \odot PrepareCoffee_{(*,proactive)} = \{&motion\_sensors,\\ &coffee\_preparation, coffee\_machine\}\end{aligned}$$

As the above example illustrates, the wildcard $*$ can be used to

[3]Communication channels among components are also defined in the superimposition operator, but they are omitted in the example for clarity.

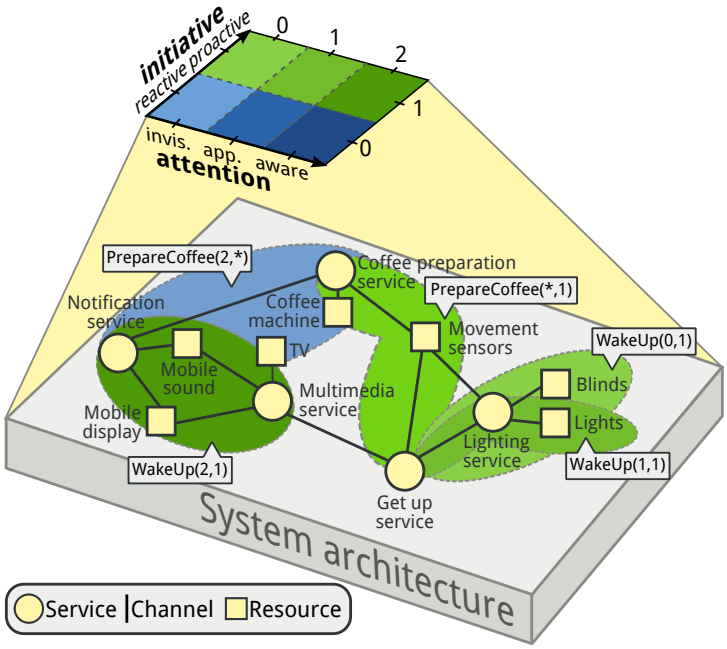


Figure 6.6: Mapping between architecture components and obtrusiveness aspects

match any value for a given axis. This is convenient for indicating that some components will be active for a specific level on a given axis, regardless of the value of the other axis. If the *prepare coffee* task is provided in a *proactive* manner (the *user-awareness* level defined above) at the *foreground* of user attention, both of the above superimposition operations ($PrepareCoffee_{(aware,*)}$ and $PrepareCoffee_{(*,proactive)}$) are composed. According to this configuration, motion sensors are used to trigger the coffee preparation automatically and the mobile device is used for notification purposes in order to make the user aware of the interaction.

MoRE is in charge of updating the system components as a reaction to the adaptation rules defined. MoRE keeps a model of the system architecture (the system components are described by means of services, channels, and resources). When a certain context condition is met, some

architecture fragments can be enabled or disabled. MoRE is in charge of querying the architecture model to determine the resulting architecture and update the actual components from the underlying architecture. MoRE does not impose restrictions in the definition of particular criteria for specifying the architecture fragments or the context conditions. In our approach, adaptation rules and the architecture fragments are specified according to the desired obtrusiveness level.

6.3 Development of reconfigurable components

MoRE is a generic reconfiguration engine that manipulates an external system to augment it with reconfiguration capabilities. MoRE determines how the architecture of an operated system must be updated in any moment according to some adaptation policies described by models. However, the system that is being operated by MoRE must fulfill a set of properties in order to interoperate with MoRE.

From the developer perspective several considerations must be taken into account when the workflow that is being developed requires adaptation capabilities. These considerations are related to the functionality to be adapted, the provision of context information sources for triggering the adaptation, the infrastructure required for the communication between MoRE and the system, and some efficiency considerations. The following subsections provide detail on all of these aspects.

6.3.1 Develop alternative components

MoRE provides adaptation capabilities to a system by activating/deactivating its components. Typically, a system that follows this approach includes all the components that can be potentially activated when the system is developed. Thus, multiple variants must be considered during the development of certain components in the system. This section provides clues on how to develop these variants by following our development method.

Since the physical mobile workflows developed in this work are based on Presto platform, components of the system are isolated by the defi-

nition of plug-ins. This makes plug-ins easy to activate and deactivate at run-time. The simple approach to make components adaptable is to develop a plug-in for each of the variants of the system. However, it is usual that plug-in variants share most of their functionality. In this case services can be defined to represent the common functionality and different channels can be established between the component variants that use them.

The degree in which functionality overlaps among component variants varies from case to case depending on the domain. For example, in the Smart Library case study the *return book* task can be performed at different levels of attention. However, the functionality provided at the foreground of the user attention is the same that is provided at the background with the exception that the user is notified about the book return. Thus, the common functionality that deals with registering the return and the notification mechanism can be defined as different services that are activated independently. The benefit of this approach is that once code generation is applied, developers only need to provide the common functionality in a single point.

6.3.2 Connect sources of contextual information

The reconfiguration technique applied provides adaptation capabilities that are defined in terms of context events. As it has been introduced in our approach, the context information must be described by means of an ontology in order to adapt to a certain kind of context changes. In addition, it is required to keep MoRE informed about the changes in this information at run-time. Thus, developers should provide mechanisms for integrating the different sources of context information that are relevant for a workflow.

For the Smart Library case study we integrated contextual information from different sources. On the one hand, the task context was provided by querying the business process execution engine. On the other hand, environmental context was provided by integrating both the elements detected by mobile devices and a pervasive system in a Smart Hotel Room system. The integration of such context sources are described below.

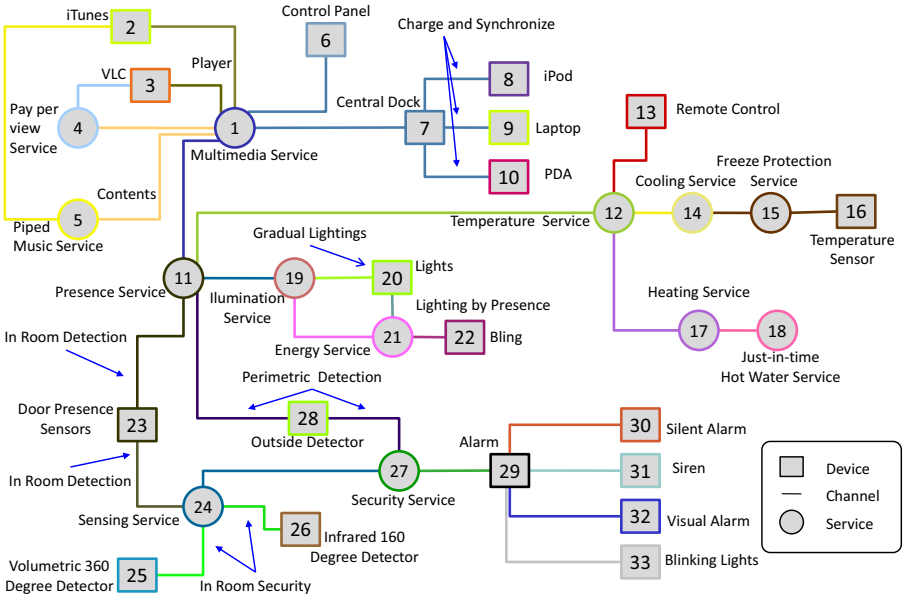


Figure 6.7: Smart Hotel architecture model

To support the execution of the business process, we make use of the business process execution engine provided with the Intalio Business Process Management Suite[4]. The Intalio suite supports the definition of business processes by means of BPMN diagrams (which are internally transformed into WS-BPEL definitions). The process execution keeps track of the process context (e.g., pending tasks for each user, priority of the tasks, etc.). A service was developed to monitor changes in the business process execution. This service makes use of the API provided by Intalio for accessing the process execution state (process variables, result of conditional expressions, etc.). When a change is detected, this component updates the ontology accordingly. For business process execution engines that do not provide such capabilities, some additional tasks can be added to the process to inform external components such as MoRE.

[4]http://bpms.intalio.com/

In addition to Intalio execution engine, a scale environment of a Smart Hotel Room was integrated to make the system sensitive to additional kinds of context events. Figure 6.7 depicts the architecture components defined for the Smart Hotel. The scale environment includes different sensors and actuators to provide services such as temperature and illumination control, presence detection, or controlling a simulated coffee machine. The different devices involved in the system (e.g., volumetric and infrared detectors) are controlled by the PervML framework (Muñoz & Pelechano, 2005). Since PervML is based on high level descriptions of pervasive services, the changes that are relevant for the user can be easily detected and transferred into an ontology (Serral et al., 2008). The previous approach makes use of the PervML models to provide updated information of the current system state by means of an OWL ontology.

6.3.3 Extend the infrastructure

With respect to the operated system, MoRE follows the Dependency Injection pattern (Fowler, 2004). It sends messages to the different services in order to enable them, disable them or inject a dependency (e.g., use service X for notification instead of using service Y). For integrating each system with the reconfiguration engine, an infrastructure component is required to perform the operations required by MoRE. This component acts as a local broker and it must support the following operations in behalf of MoRE:

Activation and deactivation of services. If adaptation is desired, it is convenient to select a platform that allows components to be easily managed. For example, Android allows the dynamic activation and deactivation of services by means of the *PackageManager* (from the *android.content.pm* package). *PackageManager* provides general management capabilities for the components of the framework. Components can be queried, enabled and disabled. One important aspect to take into account is that MoRE works with a model of the architecture that does not have a one-to-one correspondence with the underlying system architecture (e.g. in Android). Thus, when the architecture mapping is defined, a mechanism must be provided to identify the implementation

components that support each service in the architecture model. For the mapping developed to Android, components are tagged by means of an *intent filter* that identifies the abstract component (e.g., a Task Processor) they are representing. In this way, component query capabilities can be used to retrieve all the components supporting this intent.

Dependency changes. In addition to components (de)activation, communication channels can be also re-organized. Since there is not direct support in Android for communication channels among components, we have introduced an indirection mechanism based on intent routing to support them.

Intents are used in Android as a loosely-coupled communication mechanism among components. In order to dynamically control which component is capable of responding to a given intent, we have introduced an intermediate level that handles intents. In this case, the broker intercepts a given intent and translates it to a more specific one that matches the architecture configuration model that is handled by MoRE. For instance, if MoRE indicates that the *Lamp Light* resource is used for the purpose of notification by the service S (i.e., the architecture model contains a communication channel that connects the service S with the *Lamp Light* resource), the intents triggered by S for notification would be routed to the *Lamp Light* resource. When a notification intent is launched by service S, the broker replaces the general intent with one that contains the same data but a different action description. This action description would be the action corresponding to the Android *Service* controlling the *Lamp Light* device (e.g. "es.upv.pros.Illumination.LampLight"). By applying this technique, communication among components can be reorganized by MoRE.

6.3.4 Consider efficiency aspects

Adaptation capabilities provide a greater degree of flexibility for workflow execution. By following an adaptation approach based on models we can adapt the system in terms of concepts of a high abstraction level. The introduced model-based reconfiguration is subject to the same efficiency requirements as the rest of the system because the execution of the reconfiguration impacts the overall system performance.

As opposed to the application of models at design, the use of models at run-time incorporates latency in the system that is determined by (a) *the model manipulation frameworks*, (b) *the model population* and (c) *the metamodel* (which defines the model schema). In this section, we evaluate the performance of manipulating models at run-time using EMF and EMF Model Query. Specifically, we demonstrate the feasibility of using at run-time the models introduce in our approach.

For evaluation, randomly generated models were used. These models started with one element and they were populated with two hundred new elements each iteration. After the model population, the following model operation were performed: *Current Configuration*, *Superimposition*, *Architecture Increment*, *Architecture Decrement*, and *Apply Resolution Action*. These operations are in charge of querying the current configuration (*Current Configuration* operation), defining the mapping with the architecture components (*Superimposition* operation), calculate which components to update (*Architecture Increment* and *Architecture Decrement* operations) and apply the corresponding reconfiguration actions (*Apply Resolution Action*).

Our approach requires the model operations to be efficient enough to gather the necessary knowledge from the models without drastically affecting the system response. Figure 6.8 shows the milliseconds that take performing each one of the model operations. Even with a model population of 45000 elements in each model, the model operations provide a fast time response ($<$ 200 milliseconds) at least for the kind of applications we are addressing. Designers should take into account these results in order to determine whether this technique can be applied in a particular domain according to its performance constraints.

6.4 Conclusions

In the same way that a musical orchestra requires a conductor to indicate who should play and the tempo to be followed, we consider that service orchestration in the IoT environment should not only determine which services must be provided to the user but also how much noticeable the interaction with them should be.

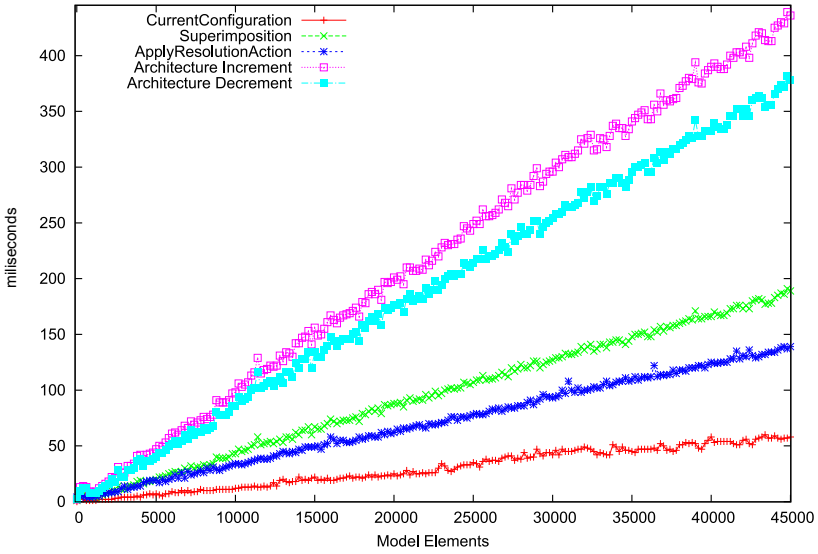


Figure 6.8: Performance of the model handling operations

This chapter makes use of model-based reconfiguration techniques to adapt the system architecture to cope with high-level directives. With the increase in the capabilities of computing devices, the bottleneck for the system can be on the human resources such as attention. Thus, it is important to manage these resources in an effective manner. The mechanisms provided for adapting the obtrusiveness level at run-time allow the system to be considerate with the user. By considering the user context, users can participate in the IoT in a more comfortable manner.

Chapter 7

Validation of the proposal

This chapter describes the application of our approach in practice. Our method was applied in order to support different business processes in the library of our Faculty by improving the physical-virtual linkage in this context. The number of users of the faculty library varies very much depending on the year period. In certain periods (e.g., during exams) it is very hard for the library personnel to attend the user demands in a reasonable time. Since most of the tasks of the librarians require to cross the physical-virtual gap, this becomes a good setting for applying our approach.

By supporting the library services with physical mobile workflows, users are able to access library services in a more fluent manner. We defined the Smart Library case study in order to improve business process by means of a better physical-virtual connection. Based on this case study, we validate whether the method defined in this work can cope with physical mobile workflow requirements. In particular, in this chapter we are verifying the following aspects:

Design method. The information captured at design should describe the aspects that are relevant to capture physical mobile workflow

requirements. Our research results show that the models defined are useful for designers to discuss about system requirements.

Iterative design. Requirements must be captured in a way that it is feasible to validate them with fast iterations. In this way, continuous feedback can be obtained to improve the designs. Our research results show that fast-prototyping techniques can be applied in a way that reproduce the user experience of the final system.

Implementation. It must be feasible to obtain an implementation that fits with the requirements described. Code generation techniques can be applied to simplify the development.

The approach followed for validating the proposal is focused on the previous aspects. First, we wanted to verify that the design method was appropriate for describing physical mobile workflows. In order to respond to this, we modeled the new workflow for the library. Then, we make use of fast-prototyping techniques in order to re-design the process. On the one hand, feedback was gathered from end-users in order to verify how the design method could deal with the changes iteratively. On the other hand, we evaluated to which extent the prototypes provided are representative of a final system. This determined the usefulness of the fast-prototyping technique in providing quality feedback. Finally, a software solution based on Presto was developed following the development process defined in Chapter 5.

The remainder of this chapter is structured as follows. Section 7.1 introduces the design of the workflow. Section 7.2 describes the validation with end-users by means of early stage prototypes and the redesign of the system to solve the issues detected. Section 7.3 provides information regarding the implementation of the final software solution. Finally, section 7.4 concludes the chapter.

7.1 Designing the smart workflow

For the design of the Smart Library case study we applied the design method defined in Chapter 4. According to this method, different aspects of a physical mobile workflow should be captured in models. The

detail of the information captured in each modeling perspective includes the *user activities* detected for the business processes in the library, the requirements for *the interaction with physical elements*, the *technology analysis* to select the most appropriate technologies, and the *deployment configuration* that determines how the system functionality is distributed in the physical space. All these aspects are detailed bellow.

7.1.1 User activities

Books are the central resource in the library. Most of the activities that are performed in the library involve books in a certain way. Books are manipulated by different people in the library context in order to fulfill their goals. The Smart Library scenario involves people playing the following roles:

Library user. The members of the library are allowed to borrow books. Their goal is to find the book they need. This requires to perform different tasks depending on the situation such as deciding among several possible alternatives or physically find the book in the library. The inability to find the appropriate book and the need for waiting are their main concerns.

Librarian. Their mission is to keep the library organized and to assist the library users in their tasks. However, the number of librarians is much reduced compared to the number of students. Thus, the tasks performed by this role are potential bottlenecks for the process.

Security personnel. Since the system cannot completely control what is happening in the real world, security is sometimes needed to enforce a correct use of the physical resources. In the library scenario, security staff is in charge of avoiding books from being stolen.

The roles defined above perform different activities in order to fulfill their goals. For the Smart Library scenario, we focused on defining services that library users require while they are exploring the library

in-situ. We defined the services using our previous experiences in libraries and by observing user behavior at the university library. After a brainstorming session, the services defined were the following:

Loan process. Users are enabled to directly borrow books in the place where the books are physically located, without requiring a librarian to be present. For the return, books are just dropped into the return box to complete the process. The need to return the books is reminded to the user when the loan is close to expire.

Related information. Given a book, we considered the possibility of informing the users about similar books and comments from other users. In this way, users can better determine whether or not the book is worth reading.

Book blocking. We observed that many users carry books around the library with them while they are deciding whether or not to borrow them. Their intention is to prevent others from borrowing a book which they might be interested in. We provide the possibility of digitally blocking books for fifteen minutes. During this time, only the user who blocked the book can borrow it.

Security. Only the books that are on loan can exit the library. The security service must be provided in a way that does not disturb the users. Users which performed their loans correctly must be able to exit the library without stopping at the security control.

By providing library services closer to the place where they are needed, users can perform their tasks more fluently in comparison to their experience in the library. In their previous experience, loans were performed by the librarian in the library desk, information about books could be only gathered from two computers that were located far from the shelves and the temporal reservation was only feasible by physically carrying the books with them. In practice, the physical barriers for accessing the information affected negatively the user experience. With the system proposed, users can perform these tasks by interacting directly with the physical object of their interest using their mobile

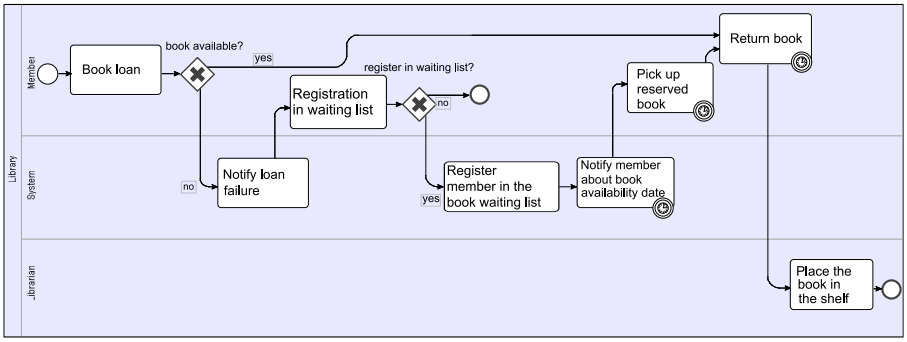


Figure 7.1: Business process mode for the book loan in the Smart Library case study.

device. Services are accessed in a more autonomous manner, reducing the dependency between library users and the other roles.

The book loan process is the process that involves more activities. Figure 7.1 shows a BPMN diagram that defines the temporal dependencies for the different tasks involved in the book loan process. Initially, a library member picks up the book in which she is interested and access the associated information by means of her device. Then, the library loan application running in her device checks the state of the book and informs the member about it. If the book is available, then the loan is performed. If not, the member is notified and asked to register in its waiting list. At this point, the member decides either to reject registering in the waiting list or accepting to be included in it. If so, when the available date is reached, the library loan application notifies the member and she can pick up the book at the library. If the expiry date approaches, the member is reminded. Then, within a predefined period, the member returns the book simply by leaving the book in the library "return box". The use of the return box concept is common in many libraries (see Fig. 7.2). A member (or someone in her behalf) can drop the book inside the box in order to return it. In this case, Auto-ID technologies can be user to automatically detect when a book has been returned.

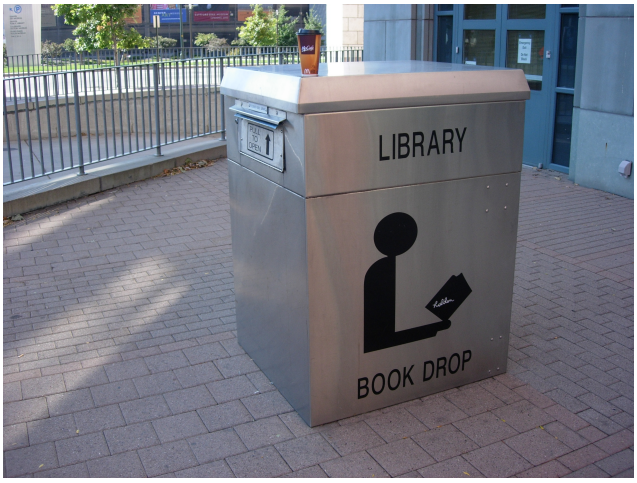


Figure 7.2: Book return mechanisms at the Denver Library

Thanks to the use of Auto-ID, library members are not required to directly interact with librarians. Queues are avoided provided that library members could access the system with their own mobile devices. Finally, the librarian personnel is in charge of placing again the book in its corresponding shelf. In this case, the librarian is guided by her device to the proper location.

7.1.2 Requirements for physical interaction

Books are the only kind of physical element involved in the Smart Library case study. Books participate in the business process from different perspectives. For some services a book is a unique item while other services are not specific to the particular book instance but the literary work it represents. For example, the book loan process is performed at an item level. A member borrows and returns a particular book copy. Conversely, comments about the book are performed by different users commenting the same literary work regardless the particular copy they have.

In order to support the multiple perspectives of the book element, it is represented in our data model at different abstraction levels. On the

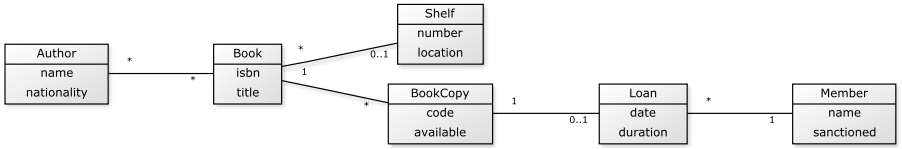


Figure 7.3: Class diagram for the data model used in the Smart Library case study

one hand, the *BookCopy* element is defined to represent the book at the item level. On the other hand, the *Book* element is defined to represent the literary work. Other levels could be considered in order to reflect the different editions of the book. Since this work is not focused on data modeling, we opted for a simple model that was capable of supporting the case study needs. Figure 7.3 illustrates an excerpt of the data model used for the case study. The model also includes information regarding the library users and heir loans.

Since books are individually identifiable and they can participate only once for some of the processes involved in the Smart Library, they can be used for process correlation. In particular, books are used for correlation in the *book return task*, *book blocking* and *security* activities. For example, when a book is returned it can be found the active loan that contains the book in order to process it.

Once we have defined the participants in the process, the tasks they perform and the physical elements involved, we define how the process is integrated in the physical environment. The way in which is performed the tasks in terms of obtrusiveness and physical interaction is detailed below.

Obtrusiveness

Each task of the workflow can be provided at different obtrusiveness levels. Figure 7.4 shows some of the most representative tasks of the Smart Library case study and their obtrusiveness level. The obtrusiveness space in this case was defined by dividing each axis in different parts as it was illustrated in Chapter 4. The *attention* axe is divided in

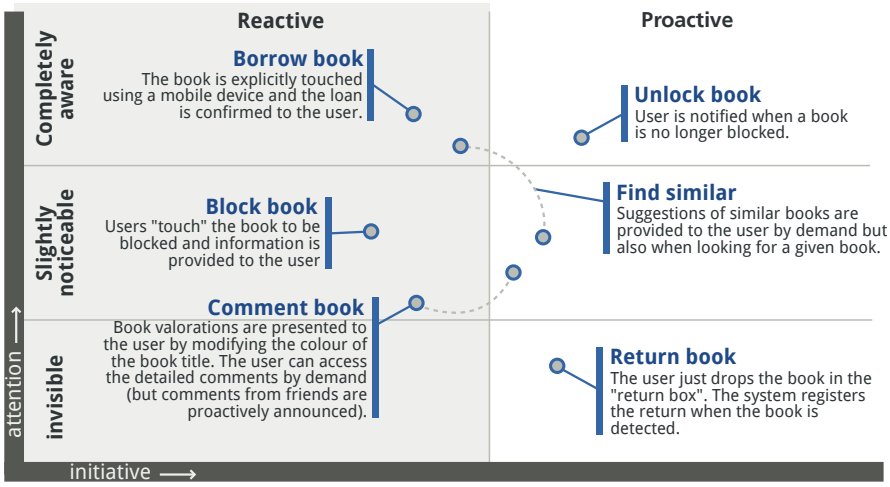


Figure 7.4: Obtrusiveness level defined for each task in the Smart Library scenario

three levels depending whether the interaction should be *invisible* to the user, *slightly noticeable*, or *completely aware* for the user. The initiative axe is divided in two parts that represent interactions initiated by the user (*reactive*) and interactions initiated by the system (*proactive*).

During analysis we decided the appropriate obtrusiveness level for certain tasks, but several levels were considered for others. The later would require to be adapted at run-time according to what is discussed in Chapter 6. The obtrusiveness level for the different processes in the Smart Library are detailed below.

Loan process. Borrowing books is performed in an explicit manner by users. Users initiate the interaction (*reactive*) and they are informed about the loan (*completely aware*). The reminder to the user is performed in a proactive manner in terms of initiative but the attention level required is dynamically increased as the deadline approaches (three days before the date, the functionality is provided at the *completely aware* level). The return of books is performed in a completely unobtrusive manner, the user leaves

the book in the *return box* and the system initiates the return process without notifying the user.

Related information. Not all the information that is associated with a book is relevant for the library users at anytime. For the tasks that provide complementary information regarding a book we provide the information in a noticeable manner when it is considered to be relevant. We provide the interaction at a different obtrusiveness level depending on the relevance of the information. User comments about the book that are made by friends are considered to be more relevant that those made by people that is not part of the user social network. Thus, comments from friends are provided in a proactive manner to the user. When a book is detected, the user is announced that some friends commented this book without the user to express the intent of accessing comments. When the user is looking for similar books to a given one, the books that are physically close to the user position are considered more relevant. Although the primary criteria for choosing similar books is the proximity of their content not their position, by emphasizing the books that are physically closer to the user, users can make more efficient routes when exploring the library. For close books, information is provided in a proactive manner, but it is only shown as a hint (*slightly noticeable*) (see Fig. 7.4). Thus, it can be later implemented as a soft vibration or some non-intrusive mark on the screen to indicate that additional information exists.

Book blocking. Book blocking is performed explicitly by library users. When it is performed, it is "suggested" in order to increase fluency (i.e., at the *slightly noticeable* level of attention). For example, when a book is blocked, a small number indicating the remaining minutes appears close to the book title. When a book is no longer blocked, this is announced to the users that are waiting for the book in a *proactive* and more notorious manner.

Security. When the user exists the library. The system checks in a proactive manner whether the user is carrying any book that is not on loan. The library personnel are warned if someone is taking

out a book. On the one hand, if the book is on loan the picture of the member is displayed (*slightly noticeable*) in order to allow the personnel to verify whether the user carrying the book is its legitimate owner. On the other hand, if the book is not on loan an alarm is raised (*completely aware*).

In order to support the behavior described above for the tasks performed in the Smart Library case study, different interaction techniques can be applied. The mechanisms used for interacting with the environment in the Smart Library are described below.

Interaction technique

According to the previous requirements, different interaction techniques are used by following the guidelines defined in (Rukzio et al., 2007). The interaction techniques used to access the services in the Smart Library are detailed below.

Touching. This technique is used for explicit interactions where the user is nearby the object. Users can touch the book with their device and access the services available for this particular book. This is the case when borrowing a book and gathering related information explicitly (proactively at the foreground of user attention).

Pointing. For objects that are a bit far from the user, pointing technique can be applied. In the Smart Library, when the user faces a shelf, the relevant books are suggested. In his case the location of the user is used to determine the shelf to which she is pointing. Pointing can also act as a replacement technique in the case touching is not available.

Scanning. For the tasks that must be performed in an unobtrusive manner scanning can be used. Tasks such as book return or the detection of book at the library exit must be as unobtrusive as possible to avoid queues. Users exiting the library and books that are left in the return box are automatically processed when they are detected.

These interaction techniques can be supported by different technologies. In the Smart Library case study, we selected a set of technologies that support the appropriate interaction techniques and could be reproduced in practice either in a real environment, in a scale environment or simulating some aspects. The following section provides detail on the technologies used for supporting each interaction technique.

7.1.3 Technological analysis

In order to choose the most adequate technologies, we first define them in an abstract manner by detecting the possible *mediums* used for identification. The mediums defined are described below.

Radio. Identification by means of radiofrequency enables the application of *scanning* and *touching* interaction techniques. Since radio does not require direct-line of sight, no artificial elements are perceived by the user during interaction. Conversely, since an augmented item looks identical to a non-augmented one, some mechanism should be user for announcing that the element has digital services associated to them.

Image on paper. The use of visual marks is an interaction method that offers a lower degree of automation than the identification based on radio but it is less technologically demanding. Much of the mobile devices nowadays have an on-board camera that allow the recognition of visual markers.

Position. The identification by position is useful for objects that are in the same place most of the time. In the case of the Smart Library, each book is assigned to a particular shelf. When the book is not on loan, its position is static. Thus, the position can be used as an identifier for services that involve only available books. For example, it is user to prioritize book suggestions that are close to the user when the user is exploring the library. For the support of pointing interaction technique, some technology such as digital compass may be required to provide the user orientation. Otherwise, only *scanning* technique can be applied.

Once we have defined the possibilities for identification that could support the desired interaction techniques, we select specific technologies to support them. Multiple identification technologies have been selected for this case study. The technologies considered in this case are described below.

RFID. RFID is used for supporting the *touching* and *sensing* interaction techniques. Depending on the technique supported a different range is required. Short range antennas are enough for touching, but scanning require an antenna with a large range.

QR Codes. QR Codes provide support to the *pointing* interaction technique. However, since the visual mark must be orthogonal in a direct line-of-sight with respect to the device, it cannot be used to point at several books put on a shelf. QR Codes are used in the Smart Library as an alternative mechanism to RFID for *touching-based* interactions. The technology required for decoding QR Codes is more affordable. Thus, most of the current mobile devices are capable of using it. This allows users to use their own mobile device when participating in the process. Interactions based on *scanning* cannot be easily supported by QR Codes.

Bluetooth. In order to determine the shelf to which the user is close to, bluetooth is used. Bluetooth was chosen by considering the limitations of location systems such as GPS for working indoors. Bluetooth range allows to have enough precision to determine the library zone where the user is. Positioning is far from perfect, but knowing a coarse grained position if it is complemented with orientation information could be good-enough for the task.

According to the previous selection, books are tagged using both RFID tags and visual markers, and library shelves are tagged with a bluetooth beacon to broadcast its identity. Table 7.1 shows a summarized view of the technology analysis results performed for the tasks where the physical-virtual linkage is critical. The table shows for each task, the obtrusiveness levels at which it can be performed, the interaction techniques that can be applied and the technology that must be

Task	Obtrusiveness	Technique	Technology
Borrow book	(reactive, aware)	touching	RFID, QRCode
Return reminder	(proactive, slightly) (proactive, aware)	–	–
Return book	(proactive, invisible)	scanning	RFID
Comments	(reactive, aware) (proactive, slightly)	touching, –	RFID, QRCode
Similar books	(reactive, aware) (proactive, slightly)	touching, pointing	RFID, QRCode, Bluetooth
Block book	(reactive, slightly)	touching	RFID, QRCode
Book released	(proactive, aware)	–	–
Security check	(proactive, invisible)	scanning	RFID

Table 7.1: Technology analysis for the Smart Library

used to support them. Some tasks are lacking interaction technique since they are triggered by some event not related with identification. Tasks such as the return book reminder and the book released are time-based. When a certain moment arrives, the system proactively informs the user.

7.1.4 Deployment configuration

In order to support the different tasks in the smart workflow, the functionality is divided into different deployment units. These deployment units are supported by a mobile device, although other kind of device can be also used as it is the case of the return box. Each deployment unit is operated by a different role to fulfill their tasks. The deployment units defined for the Smart Library case study are defined below.

Member mobile. Library users can perform most of their tasks in the library through their own mobile device. They are provided support to *borrow* and *blocking books*, and accessing *associated*

information. For these tasks, their mobile device must be capable of supporting either RFID or QRCode technology. Optionally, bluetooth support would be also desirable in order to prioritize the related information depending on the books that are physically close to the user. In addition, library users use their mobile device to receive different notifications (return reminders and available books) in their mobile device.

Librarian mobile. Librarians make use of RFID to determine where must be placed a book that has been returned. RFID is the preferred mechanism for librarians to interact with books since it supports better the *touching* interaction. Librarians are notified when books are returned in the return box. They can also explicitly return books if the user prefers interacting with the librarian or the return box is not available.

Return box. RFID capabilities are required to detect the books that have been returned in an unobtrusive manner. In order to reduce the risk for a book not being detected, the return box can make use of additional sensors (e.g., detecting the weight of the content).

Security door. RFID is used to detect books at the exit of the library. For users that behave according to the rules, they do not perceive that the security system is inspecting the books they are carrying. Security personnel can validate whether the person exiting the library is the owner of the detected book or not since an image of the library member is shown to them. An alarm service is accessed in case, the books detected are not on loan.

Library shelf. Library shelves are provided a digital identity that is broadcasted by means of Bluetooth. This is useful to provide location-aware information to the library participants. Librarians can be warned if they approach the wrong shelf when returning a book, and users can organize their books of interest according to their location.

The requirements captured following our method describe the physical mobile workflow to support the Smart Library scenario. Since the

design method was followed, the different decisions are organized in different layers and consistency is guaranteed among them. Although the workflow defined is consistent, this does not guarantee that the system described is wel accepted by users. Once the system is designed, the following section detects whether the design decisions taken were appropriate.

7.2 Early-stage evaluation

The MDE techniques defined in Chapter 5 can be applied to the previous requirements to obtain a software solution to support the system. However, there is still place for fast-prototyping. In the case of physical mobile workflows, deployment efforts are high if real hardware is used. Designers need certain guaranties that the business process defined will work when it is finally deployed.

Fast-prototyping techniques have been applied in order to immerse the user in the workflow designed without actually implementing it. HTML interface mock-ups and Wizard of Oz techniques are used to simulate the process. From the device perspective, the only requirements for applying the technique are wireless connectivity and HTML rendering capabilities. Since we had access to our department library the physical context was easy to reproduce.

Once the user is immersed in the simulated environment, the **user experience in terms of usability and productivity** is evaluated. To evaluate these aspects, we use MoBiS-Q (Vuolle et al., 2008), a questionnaire to measure mobile business service experience. MoBiS-Q evaluates user experience combined with enhancements in work productivity. Three dimensions are evaluated with the questionnaire: *perceived usability of a mobile business service*, *fit for mobile working context*, and *perceived impact on mobile work productivity*. The first dimension focuses on the functionality provided to the user; the second one focuses on the use of the service through the mobile device; and finally, the third dimension is about improvements in the fluidity of work.

For the evaluation of the Smart Library prototype, we defined different scenarios that users must follow and put into practice at the

department library. In-situ evaluation was possible since the technique does not require a complex infrastructure (e.g., tagging books or installing RFID readers) is not required. An iPhone was used to interact with the Smart Library services. We chose this device because of its HTML rendering capabilities and the availability of frameworks such as iUI[1] for the development of native-like web applications. The use of these capabilities helps to obtain a realistic look and feel with little effort .

The prototype developed for the evaluation took a single student two days to prepare. Even though the development required little effort, the user immersion that was achieved was excellent. When the users evaluated the prototype, they were not told that it was a non-functional prototype. After the evaluation, when they were told that it was not a final functional system, more than a third of the participants confessed that they thought that it was. This means that **it is possible to anticipate the feedback that could be obtained from the final system with minimal effort**.

The experiment was performed by 34 users (26 men, 8 women; between the ages of 20 and 60), 7 were familiar with the mobile device used. We applied a Likert scale (from 1 to 5 points) to evaluate the items defined in the MoBiS-Q questionnaire in different scenarios that comprised several alternate, simultaneous and repetitive tasks. They were required to borrow books (waiting for them to be available in the case they were reserved), make temporal reservations while looking for similar books with a better rating. Figure 7.5 shows a summarized table of the obtained results[2].

More than 80% of the people highly (4 points) or totally agreed (5 points) about each of the service-related questions. The offered functionality was perceived as useful and the way of presenting it to the user (following the dynamic to-do list metaphor) was considered intuitive for navigating and accessing the desired functionality. One user described the service as being a "universal remote control providing the information needed at each moment".

[1] http://code.google.com/p/iui/

[2] The complete results can be found in the Appendix B

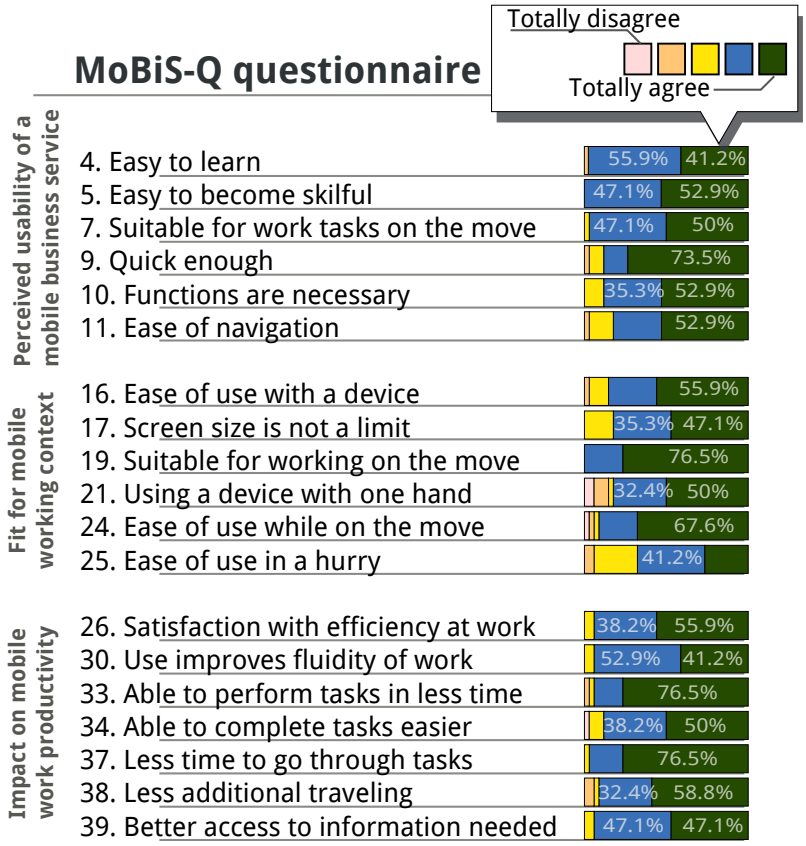


Figure 7.5: Summarized results

With regard to the *fit for mobile working context*, the results were also positive, but more dispersion was found in them. This was due to the different levels of expertise in the use of mobile devices for accessing applications that go beyond the typical phone capabilities. Specifically, using the device with only one hand (question 21 of MoBiS-Q) was considered hard to do by 14.6% of the users, while the rest found it highly (32.4%) or totally (50%) feasible.

With regard to the *perceived impact on mobile work productivity*, more than 90% of the users positively rated the increase of productivity

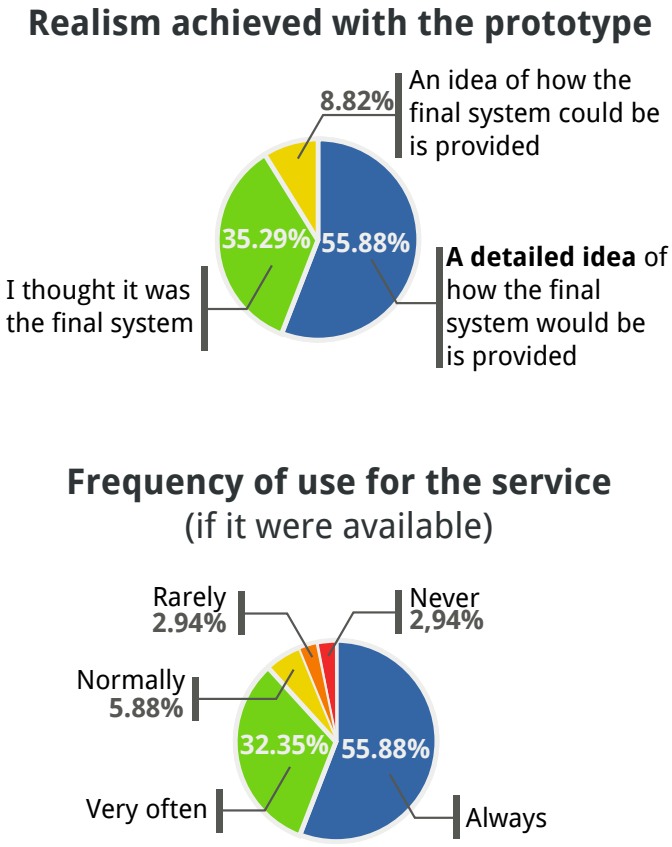


Figure 7.6: Relevance of the feedback obtained

experienced when they compare the service provided with the libraries they normally access. The users indicated a strong belief that tasks could be performed more quickly (questions 30, 33 and 37). However, despite the general improvement in process efficiency, 11.7% of the users considered that tasks were not actually simplified since they were assuming some tasks that are traditionally performed by librarians.

Finally, we included in the evaluation some questions to determine the relevance of the feedback obtained (see Fig. 7.6). One of the ques-

tions was related to how much the prototype provided a detailed idea of the final system. All the participants considered the prototype to be helpful for developing the final system. Of these, 55.88% considered that it provided a very detailed view of what the system should be, and 35.29% reported that they even believed that it was the final system. We also asked how often they would use the service if it were available in their library. A total of 88.23% of the users were willing to use the service *always* or *very often*, with the only complaints being the limited availability of Auto-ID-enabled devices to properly access the service.

7.2.1 Workflow re-design

The feedback from the users was used to re-design the smart workflow iteratively. Some concerns resulted in minor modifications of the mock-up interfaces (e.g., using graphical metaphors and bigger buttons). Other suggestions were more relevant to the process redesign since they involved changing the *level of obtrusiveness* for some tasks or the inclusion of new tasks in the process.

More detailed capabilities were requested for *book blocking*. Initially, the user was informed by means of notifications. However, users wanted to take the *initiative* and access this information on demand. Some concerns were also raised about the possibility of vandal users abusing the block feature. As a consequence, a limited-size list of blocked books was provided for users to manage.

Book location was the functionality most demanded. When looking for similar books, users wanted to physically locate them. Thus, we extended the process model to include this task as the next step for the *similar book finding* task.

The obtrusiveness level for the *return of books* was also problematic. In the first iterations of the design, the return of books was completely unobtrusive for the users (just dropping a book into a return box). However, users were not sure whether the return had actually taken place or not. Thus, we adapted the *return book task* to provide a notification message. Therefore, the interaction is still reactive in terms of *initiative*, but it is performed in the foreground of user *attention*.

7.3 Obtaining a final implementation

After redesigning the business process, the case study was finally implemented using the Presto platform. The requirements captured in Section 7.1 were specified according to the Parkour metamodel. In this way, requirements become machine-processable and can be used to guide the development by automating some tasks. From these requirements, the initial version of the system was obtained. This version defines a different Presto system for each deployment unit defined above.

Each deployment unit resulted in a Presto system that includes a *Task Processor* to support each tasks of the user, *Identification Components* to support the different identification technologies, and *Data Providers* to represent the different perspectives provided over books. More detail regarding the process for obtaining a functional software solution is provided in Chapter 5. This section provides detail on the manual development that was required to obtain the Smart Library prototype based on Presto.

7.3.1 Task support

The tasks supported by the smart workflow are not complex in their business logic. *Task Processors* are in charge of presenting some information to the user that they extract from the pending task, and complete this information with some input from the user. Due to the intrinsic I/O limitations of mobile devices, user interfaces have been designed to be simple and provide predefined options when it is possible since selection is preferred to text input.

In order to support different obtrusiveness levels, additional services have been added to the original system. In particular the notification service is defined to support proactive notifications by the system. It is implemented by wrapping the *Notification Manager* component from Android. This is used for notifications such as the *return reminder* and the *book released*.

Alternative Task Processors have been developed for tasks that can be performed at multiple obtrusiveness levels. For example, the *similar books* task is supported by two alternative plug-ins. One makes use

of the notification service and the other does not. These components can be activated and stopped at run-time without problems since both support the same intents and the application state is mainly kept in the *Task Manager*.

In order to validate the functionality provided by the Task Processors developed, the scenarios defined for the non-functional prototypes were used as a reference. Since the non-functional prototype defined for the early-stage evaluation obtained positive results in terms of usability and process performance, the *Task Processors* were implemented to define a user experience as close as possible. We validated that the obtained system could reproduce the scenarios supported by the prototype in the same way.

7.3.2 Integrating identification technologies

The HTC Magic mobile device was used for supporting each Presto system. The device lacks RFID capabilities but it is provided with QRCode decoding capabilities based on the ZXing library. The *Identification Component* developed, wraps the functionality provided by the barcode application from the ZXing project. Thanks to the use of intents in the Android platform, functionality from different applications can be easily integrated.

For the QRCode generation, the Google Chart API was used for the production of tags. This API provides support for the generation of QR Code tags. In order to make the recognition as fluent as possible, we selected the "H" error correction level, which is the highest correction level defined by the QR Code standard (it allows the recognition of a code that is damaged up to a 30%).

Since no current android device was provided with RFID capabilities, we provided some alternatives for its simulation. Two different *Identification Components* were defined to support RFID. One plug-in receives the identification events from a remote web service to simulate RFID behavior. This allows Wizard of Oz techniques to be applied. The other plug-in receives the events from an external RFID reader. Since Identification Components can be easily replaced, once it were available an Android device with RFID an appropriate Identification

Component could be developed to make use of it.

For bluetooth identification, the bluetooth device name was used as an identifier. Different bluetooth-enabled devices were put in different shelves to provide a coarse-grained notion of user location. Since library shelfs are normally located at different sides of the user, the orientation of the device was used to filter the detected shelves to consider only the ones in front of the user. Since this feature is only intended to prioritize additional information, it was not implemented to provide a high level of precision.

For tasks that require a the *scanning* interaction technique was required, an external Caen A828DKEU RFID reader was used. These tasks require a completely autonomous detection of multiple physical elements. This RFID reader allows multiple tags to be read but the reading distance is shorter than half a meter which avoids its use in a real environment. Thus, the library exit and the return box were supported on a scale environment. For a real deployment, RFID readers with a larger range will be required. For the library alarm, the system makes use of KNX devices that were controlled by means of an OSGi server.

7.3.3 Communication among systems

Intalio BPMS was used to orchestrate the different services in the Smart Library scenario. The orchestration engine provided the pending tasks to each participant in the system according to the process definition.

MoRE supported the reconfiguration aspects. Different kinds of context were supported for adaptation: temporal, social and spatial. The context ontology was designed to represent the remaining time for loans, the frienship relationships of users and the position of the shelves and users in the library. Depending on these factors, MoRE activates and deactivates the appropriate Task Processors.

Information in the Smart Library was supported at the server level by means of Restlet[3]. Restlet is a framework for providing and accessing REST-based services. Restlet is designed by following the principles of

[3] http://www.restlet.org

the web and provides support to flexibly represent digital resources. Restlet is available for Java and Android platforms among others. Data Providers exposed this information to the Presto application. A custom Content Provider was defined in Android to expose information in a homogeneous manner for each Data Provider. In particular, two *Data Providers* were developed. One provides information about books at the item level (e.g., used to control loans). The other provides information regarding books at the literary level (e.g., used to provide comments).

7.4 Conclusions

Modeling is about abstractions and the conceptualization of the system to be built. However, for a problem to be completely understood analysis hypotheses must be validated with the end-users of the system. René Thom, a French mathematician, stressed the relevance of connecting abstraction with reality:

> "If you want to know what happens when you throw a stone into a pond, it is infinitely better to make a trial and film it than to attempt to theorize about it."

In this chapter, we put in practice the design method defined. Our research results show that the method is capable of producing systems with a high level of user acceptance, the evaluation method provides valuable feedback and a final software solution can be obtained from what it is captured at requirements level.

Chapter 8

Concluding remarks

The present work has introduced a model-driven development method for the construction of business process supporting systems in the context of the Internet of Things. Facing the development of such systems from a specification perspective has resulted innovative and different contributions were produced from this work. In addition, the research line in which this work is aligned is by no means completed here. Further work can complement and extend this thesis.

This last chapter introduces the conclusions of the work developed in this thesis. First, Section 8.1 presents the main contributions to both the Business Process Management and the Internet of Things communities. Section 8.2 provides an overview of the publications that have emerged from this work. Finally, Section 8.3 outlines the ongoing and future work that can extend this research line.

8.1 Contributions

The main contribution of this work is a development process for the construction of business process supporting systems that integrate the

digital and the physical worlds. The development process comprises from architectonic to methodological aspects. Thus, the work provides the following contributions:

Architecture for implementation. An architecture supporting the integration of real-world elements in business process has been defined. In order to obtain a sustainable architecture, **architectonic concepts** have been detected in a technologically-independent fashion. **Mappings to a particular technological solution** have been also established to obtain a system that meets the demanded requirements.

DSL for specification. Modeling primitives have been defined to facilitate the specification of identification aspects in business processes. BPMN has been extended to cope with the integration of physical elements in process definitions. Separation of concerns and metamodeling techniques have been applied to **capture and organize formally the concepts** that conform this specification language.

Method for development. A development method has been defined to guide the developer in the construction of business process-supporting solutions for physical mobile workflows. The method comprises from specification to the final implementation. To promote separation of concerns, different development roles are defined for applying the method.

8.2 Publications

The research activity presented in this work has resulted in a total of **23 research publications**. Table 8.1 compiles all the published works indicating the author position, the type of publication and the conference or journal where it was published. The author position is used as an indicator of the degree of contribution made by the author of the present work in each one of the publications.

In addition, **three senior theses** were co-directed in the context of this work to explore some concepts and related technologies. A more

Publication	Pos.	Type	Published at
(Giner et al., 2010)	1st	Int. Journal	**IEEE Pervasive Computing**
(Giner et al., 2008d)	1st	Int. Journal	IEEE Latin America Transactions
(Giner et al., 2009b)	1st	Int. Journal	ERCIM News
(Giner et al., 2009c)	1st	Int. Conference	**Mobiquitous 2009**
(Giner & Pelechano, 2008)	1st	Int. Conference	**AmI 2008**
(Giner et al., 2008a)	1st	Int. Conference	**ICEIS 2008**
(Giner et al., 2008b)	1st	Int. Conference	UCAmI 2008
(Giner et al., 2009a)	1st	Int. Workshop	IWAAL 2009
(Giner et al., 2008c)	1st	Workshop	DSDM 2008
(Giner et al., 2007b)	1st	Conference	JISBD 2007
(Giner & Torres, 2007)	1st	Int. Workshop	IDEAS 2007
(Giner et al., 2007c)	1st	Demo Session	JISBD 2007
(Giner et al., 2007a)	1st	Int. Workshop	**MDWE 2007**
(Cetina et al., 2009a)	2nd	Int. Journal	**IEEE Computer**
(Cetina et al., 2009b)	2nd	Int. Conference	ICAS 2009
(Cetina et al., 2008a)	2nd	Int. Workshop	MRT 2008
(Torres et al., 2007a)	2nd	Int. Workshop	**CAiSE Forum 2007**
(Cetina et al., 2008b)	2nd	Workshop	WASELF 2008
(Torres et al., 2008)	2nd	Workshop	JSWEB 2008
(Torres et al., 2007b)	2nd	Workshop	PNIS 2007
(Torres et al., 2007d)	3rd	Int. Journal	IEEE Latin America Transactions
(Torres et al., 2007c)	3rd	Int. Journal	ERCIM News
(Torres et al., 2006)	3rd	Conference	JISBD 2006

Table 8.1: Summary of Publications

detailed description of the different contributions in these works is exposed below.

8.2.1 Detail of the publications

One of the key points of this work is to identify the aspects to be modeled when a physical mobile workflow is designed. A formal framework to define the concepts that are involved in the physical-virtual linkage was introduced in (Giner et al., 2008a). More detail about how to connect these aspects with architecture components and tool support is provided in (Giner et al., 2009c). Metamodeling techniques have been used to capture the concepts that define a physical mobile workflow (Giner et al., 2008c) and their integration with business process engineering approaches (Torres et al., 2007a). Methodological aspects were considered in several works (Giner & Torres, 2007; Giner et al., 2007b, 2008d) where the requirements for the development of business processes in the AmI area are detected.

Considering the conceptual foundations described above, our approach for the development of physical mobile workflows was developed. The techniques applied for the development of physical mobile workflows and the results from their application were reported in IEEE Pervaive Computing (Giner et al., 2010). An overview of the proposal that included the obtrusiveness adaptation at run-time was introduced in (Giner et al., 2009b). Most of the run-time reconfiguration mechanisms that are applied in our approach were introduced with MoRE (Cetina et al., 2009a).

Regarding the technical infrastructure that supports our approach, it has been published the mechanisms used for integrating with business process execution engines (Giner & Pelechano, 2008; Giner et al., 2007a), the integration with a reconfiguration engine (Giner et al., 2008b, 2009a), and the access to physical mobile workflows through mobile devices (Giner et al., 2009c; Torres et al., 2007a).

The present work, build upon a previous research that had the goal of automatically building business process supporting applications (not integrating physical elements). The research results and tools obtained resulted in four publications (Torres et al., 2007b, 2006, 2007c,d). Es-

sential knowledge for the development of the present work in business process-related languages (BPMN and WS-BPEL) and modeling techniques was obtained from this collaboration.

8.2.2 Relevance of the publications

This sectionprovides some information about the relevance of some of the journals and conferences where different aspects of this work have been published.

IEEE Pervasive Computing. It is recognized as one of the most important journals in pervasive and ubiquitous computing area. It is included in the top quartile of JCR (impact factor 2008: 2.615, position 12/99 in the "Computer Science/Information System" category). According to its mission statement:

> IEEE Pervasive Computing delivers the latest peer-reviewed developments in pervasive, mobile, and ubiquitous computing to developers, researchers, and educators who want to keep abreast of rapid technology change. With content that's accessible and useful today, the quarterly publication acts as a catalyst for realizing the vision of pervasive (or ubiquitous) computing, described by Mark Weiser nearly a decade ago.

The article published in this journal is part of an special issue entitled "Labeling the World" which is focused on topics that are central to this thesis. The guest editors for this special issue were Tim Kindberg, Thomas Pederson and Rahul Sukthankar, relevant researchers in the Internet of Things area.

IEEE Computer. It is recognized as one of the most important journal in computer science, and it is edited by the IEEE Computer Society since 1970. It is included in the top quartile of JCR (impact factor 2008: 2.093, position 16/86 in the "Computer Science/Software Engineering" category). According to its mission statement:

> Computer, the flagship publication of the IEEE Computer Society, publishes peer-reviewed technical content that covers all aspects of computer science, computer engineering, technology, and applications. Computer is a resource that practitioners, researchers, and managers can rely on to provide timely information about current research developments, trends, best practices, and changes in the profession.

The article published in this journal is part of an special issue entitled "models@run.time" which is focused on techniques for leveraging modeling techniques at run-time. These techniques are applied in Chapter 6.

Mobiquitous. The *International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services* provides a forum for practitioners and researchers to interact and exchange experiences about the design and implementation of mobile and ubiquitous systems. According to the CORE conference ranking, Mobiquitous is a *Tier-A* conference. This classification defines tier-A as follows:

> Publishing in a Tier A conference would add to the author's respect, showing they have real engagement with the global research community and that they have something to say about problems of some significance. Attending a Tier A conference would be worth travelling to if a paper was accepted. Typical signs of a Tier A conference are lowish acceptance rates and a program committee and speaker list which includes a reasonable fraction of well known researchers from top institutions (as well as a substantial number from weaker institutions), and a real effort by the program committee to look at the significance of the work.

The paper presented in this conference was part of the Work in Progress (WiP) track. The Mobiquitous conference acceptance ratio was 20% (32% including the works in the WiP track).

Ambient Intelligence. The AmI conference has an important role for the social cohesion of the Ambient Intelligence community. The conference is organized by Emile Aarts, who defined the *Ambient Intelligence* term (Aarts et al., 2002). The conference has a major impact in industry with the participation of relevant corporations in the AmI area such as Nokia, Philips, NTT DOCOMO or SAP.

International conferences and workshops. In addition to the above mentioned conferences, different parts of the work have been published in international events such as ICEIS (tier-B conference according to CORE) or ICAS (a reference in the Autonomic Computing area). Initial development of the work have been accepted in workshops from relevant conferences such as Models, CAiSE or ICWE. This has helped to achieve diffusion for the work. The paper introducing the model-to-model transformations for BPMN has 4 references, and led to a contribution to the Babel tools[1], a project from the *Queensland University of Technology*.

8.3 Future work

The research presented here is not a closed work and there are several interesting directions that can be taken to provide the proposal with a wider spectrum of application. The following list summarizes the research activities that are planned to continue this work.

End user development. The use of modeling techniques to formalize concepts allows for the automation of software development. In the present work, the generation of an initial version of the system that hides infrastructure issues is obtained automatically. The next big step is to provide developers and end-users with tools to intuitively develop their physical mobile workflows easily. In order to do so, the primitives that capture the requirements for the system should be provided adequate tool suport not only to make feasible the development but also to provide an optimal user experience for developers.

[1] http://www.bpm.fit.qut.edu.au/projects/babel/tools/

Location-based services. Automatic identification is central in connecting the physical and virtual spaces. Many of the applications that take advantage of this connection are also sensible to the user location. Our current proposal can cope with the identification of places but some relationships between places could be taken into account such as their proximity, inclusion and connectivity which are not explicitly considered by the method. Thus, future extensions can further explore these aspects.

Adaptive interactions. Run-time adaptation has been considered in the present work at architectural level. Components are enabled/disabled to provide services at the appropriate obtrusiveness level. A more fine-grained adaptation at interface level can be convenient in order to better adapt to the user needs and reduce duplication of efforts during development when multiple platforms are targetted.

Bibliography

Aarts, E., Harwig, R., & Schuurmans, M. (2002). Ambient intelligence. *The invisible future: the seamless integration of technology into everyday life*, (pp. 235–250).

Abowd, G. D., & Mynatt, E. D. (2000). Charting past, present, and future research in ubiquitous computing. *ACM Trans. Comput.-Hum. Interact.*, *7*(1), 29–58.

Alves, A., Arkin, A., Askary, S., Barreto, C., Bloch, B., Curbera, F., Ford, M., Goland, Y., Guízar, A., Kartha, N., Liu, C. K., Khalaf, R., König, D., Marin, M., Mehta, V., Thatte, S., van der Rijn, D., Yendluri, P., & Yiu, A. (2007). Web services business process execution language version 2.0. OASIS Specification.

Ballagas, R., Rohs, M., Sheridan, J. G., & Borchers, J. (2004). Byod: Bring your own device. In *In Proceedings of the Workshop on Ubiquitous Display Environments, Ubicomp*.

Bencina, R., & Kaltenbrunner, M. (2005). The design and evolution of fiducials for the reactivision system. In *Proceedings of the 3rd International Conference on Generative Systems in the Electronic Arts (3rd Iteration 2005)*. Melbourne, Australia.

Bézivin, J., & Gerbé, O. (2001). Towards a precise definition of the omg/mda framework. In *ASE '01: Proceedings of the 16th IEEE international conference on Automated software engineering*, (p. 273). Washington,

DC, USA: IEEE Computer Society.

Bornhövd, C., Lin, T., Haller, S., & Schaper, J. (2004). Integrating automatic data acquisition with business processes experiences with sap's auto-id infrastructure. In *vldb'2004: Proceedings of the Thirtieth international conference on Very large data bases*, (pp. 1182–1188). VLDB Endowment.

Brock, D. L. (2001). The physical markup language. Tech. rep., Auto-ID Center.

Broll, G., Haarländer, M., Paolucci, M., Wagner, M., Rukzio, E., & Schmidt, A. (2008). Collect&drop: A technique for multi-tag interaction with real world objects and information. In *AmI*, (pp. 175–191).

Buxton, B. (1995). Integrating the periphery and context: A new model of telematics. In *Proceedings of Graphics Interface*, (pp. 239–246).

Calvary, G., Coutaz, J., & Thevenin, D. (2001). Supporting context changes for plastic user interfaces: A process and a mechanism.

Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., & Vanderdonckt, J. (2003). A unifying reference framework for multi-target user interfaces. *Interacting with Computers*, *15*(3), 289–308.

Carter, S., & Mankoff, J. (2004). Challenges for ubicomp evaluation. Tech. Rep. CSD-04-1331, University of California, Berkeley.

Cetina, C., Giner, P., Fons, J., & Pelechano, V. (2008a). A model-driven approach for developing self-adaptive pervasive systems. Models@run.time Workshop. Springer LNCS.

Cetina, C., Giner, P., Fons, J., & Pelechano, V. (2008b). Using variability models for developing self-configuring pervasive systems. Workshop on Autonomic and SELF-adaptive Systems (WASELF 08).

Cetina, C., Giner, P., Fons, J., & Pelechano, V. (2009a). Autonomic computing through reuse of variability models at runtime: The case of smart homes. *Computer*, *42*(10), 37–43.

Cetina, C., Giner, P., Fons, J., & Pelechano, V. (2009b). Using

feature models for developing self-configuring smart homes. In *Fifth International Conference on Autonomic and Autonomous Systems (ICAS 2009)*, (pp. 179 – 188).

Chakraborty, D., & Lei, H. (2004). Pervasive enablement of business processes. In *PERCOM '04: Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom'04)*, (p. 87). Washington, DC, USA: IEEE Computer Society.

Cuadrado, J. S., & Molina, J. G. (2007). Building domain-specific languages for model-driven development. *IEEE Softw.*, *24*(5), 48–55.

da Silva, P. P., & Paton, N. W. (2003). User interface modeling in umli. *IEEE Softw.*, *20*(4), 62–69.

Dahlbäck, N., Jönsson, A., & Ahrenberg, L. (1993). Wizard of oz studies: why and how. In *IUI '93: Proceedings of the 1st international conference on Intelligent user interfaces*, (pp. 193–200). New York, NY, USA: ACM.

Davis, J. (2009). *Open Source SOA*. Manning Publications Co.

den Bergh, J. V., & Coninx, K. (2005). Towards modeling context-sensitive interactive applications: the context-sensitive user interface profile (cup). In *SoftVis '05: Proceedings of the 2005 ACM symposium on Software visualization*, (pp. 87–94). New York, NY, USA: ACM Press.

DiMarzio, J. (2008). *Android a Programmers Guide*. McGraw-Hill Osborne Media.

Dumas, M., & ter Hofstede, A. H. M. (2001). Uml activity diagrams as a workflow specification language. In *UML'01: Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools*, (pp. 76–90). London, UK: Springer-Verlag.

Edwards, W. K., Bellotti, V., Dey, A. K., & Newman, M. W. (2003). The challenges of user-centered design and evaluation for infrastructure. In *CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems*, (pp. 297–

304). New York, NY, USA: ACM.

Fano, A., & Gershman, A. (2002). The future of business services in the age of ubiquitous computing. *Commun. ACM*, *45*(12), 83–87.

Favre, J.-M. (2004). Foundations of model (driven) (reverse) engineering : Models – episode i: Stories of the fidus papyrus and of the solarus. In J. Bezivin, & R. Heckel (Eds.) *Language Engineering for Model-Driven Software Development*, no. 04101 in Dagstuhl Seminar Proceedings. Dagstuhl, Germany: Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.

Favre, J.-M. (2006). Megamodelling and etymology. In J. R. Cordy, R. Lämmel, & A. Winter (Eds.) *Transformation Techniques in Software Engineering*, no. 05161 in Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany.

Federal Trade Commission (2005). Radio frequency identification: Applications and implications for consumers.

Fleisch, E. (2001). Business perspectives on ubiquitous computing. M-Lab Working Paper No. 4.

Fleisch, E., & Tellkamp, C. (2003). The challenge of identifying value-creating ubiquitous computing applications.

Floerkemeier, C., Roduner, C., & Lampe, M. (2007). Rfid application development with the accada middleware platform. *IEEE Systems Journal, Special Issue on RFID Technology*.

Fowler, M. (2004). Inversion of control containers and the dependency injection pattern.

Gershenfeld, N., Krikorian, R., & Cohen, D. (2004). The internet of things. *Scientific American*, *291*(4), 46–51.

Gibbs, W. W. (2004). Considerate computing. *Scientific American*, *292*(1), 54–61.

Giner, P., Albert, M., & Pelechano, V. (2008a). Physical-virtual connection in ubiquitous business processes. In *Proceedings of the 10th International*

Conference on Enterprise Information Systems, vol. 2, (pp. 266 – 271). Barcelona (Spain).

Giner, P., Cetina, C., Fons, J., & Pelechano, V. (2008b). A framework for the reconfiguration of ubicomp systems. In J. M. Corchado, D. Tapia, & J. Bravo (Eds.) *3rd Symposium of Ubiquitous Computing and Ambient Intelligence 2008*, vol. 51 of *Advances in Soft Computing*, (pp. 1 – 10). Springer Berlin.

Giner, P., Cetina, C., Fons, J., & Pelechano, V. (2009a). Building self-adaptive services for ambient assisted living. In *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living, IWANN 2009, Part II*, vol. 5518 of *LNCS 5518*, (pp. 740 – 747). Springer.

Giner, P., Cetina, C., Fons, J., & Pelechano, V. (2009b). Orchestrating your surroundings. *ERCIM News*, (77), 40 – 41.

Giner, P., Cetina, C., Fons, J., & Pelechano, V. (2009c). Presto: A pluggable platform for supporting user participation in smart workflows. In *Proceedings of the Sixth Annual International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous)*. Toronto, Canada.

Giner, P., Cetina, C., Fons, J., & Pelechano, V. (2010). Developing support for mobile business processes in the internet of things. *IEEE Pervasive Computing*, *9*(2).

Giner, P., Fons, J., & Pelechano, V. (2008c). A domain specific language for the internet of things. V Taller sobre Desarrollo de Software Dirigido por Modelos (DSDM'08).

Giner, P., & Pelechano, V. (2008). An architecture to automate ambient business system development. In E. Aarts (Ed.) *European conference on Ambient Intelligence (AmI 08)*, (pp. 240–257). Springer LNCS.

Giner, P., & Torres, V. (2007). Una propuesta basada en modelos para la construcción de sistemas ubicuos que den soporte a procesos de negocio. In F. Losavio, G. H. Travassos, V. Pelechano, I. Diaz, & A.Matteo (Eds.) *X Workshop Iberoamericano de Ingeniería de Requisitos y Ambi-*

entes de Software. Isla Margarita (Venezuala).

Giner, P., Torres, V., & Pelechano, V. (2007a). Bridging the gap between bpmn and ws-bpel. m2m transformations in practice. In *Proc. of the 3rd International Workshop on Model-Driven Web Engineering (MDWE 2007)*. Como, Italy. ISSN 1613-0073.

Giner, P., Torres, V., & Pelechano, V. (2007b). Building ubiquitous business process following an mdd approach. In *XII Jornadas de Ingeniería del Software y Bases de Datos*. Zaragoza.

Giner, P., Torres, V., & Pelechano, V. (2007c). Generation of business process based web applications. XII Jornadas de Ingeniería del Software y Bases de Datos 2007 (Demo).

Giner, P., Torres, V., & Pelechano, V. (2008d). Building ubiquitous business process following an mdd approach. *IEEE Latin America Transactions*, *6*(4), 347–354.

Gomaa, H., & Hussein, M. (2007). Model-based software design and adaptation. *Software Engineering for Adaptive and Self-Managing Systems, International Workshop on*, *0*, 7.

Goth, G. (2009). The task-based interface: Not your father's desktop. *IEEE Software*, *26*(6), 88–91.

Greenfield, A. (2006). *Everyware: The Dawning Age of Ubiquitous Computing*. Berkeley, CA: New Riders Publishing.

Hackmann, G., Haitjema, M., Gill, C. D., & Roman, G.-C. (2006). Sliver: A bpel workflow process execution engine for mobile devices. In *ICSOC*, (pp. 503–508).

Hallsteinsen, S., Hinchey, M., Park, S., & Schmid, K. (2008). Dynamic software product lines. *Computer*, *41*(4), 93–95.

Hansmann, U., Nicklous, M. S., & Stober, T. (2001). *Pervasive computing handbook*. New York, NY, USA: Springer-Verlag New York, Inc.

Heil, A., Moradi, I., & Weis, T. (2006). Lcars: the next generation programming context. In K. Mihalic (Ed.) *CAI*, (pp. 29–31). ACM Press.

Henricksen, K., & Indulska, J. (2005). Developing context-aware pervasive computing ap-

plications: models and approach. In *Pervasive and Mobile Computing, In*. Press, Elsevier.

Hofreiter, B., Huemer, C., & Klas, W. (2002). ebxml: Status, research issues, and obstacles. In *Proc. of 12th Int. Workshop on Research Issues on Data Engineering (RIDE02)*, (pp. 7–16).

Horvitz, E., Kadie, C., Paek, T., & Hovel, D. (2003). Models of attention in computing and communication: from principles to applications. *Commun. ACM*, *46*(3), 52–59.

IBM (2003). An architectural blueprint for autonomic computing. Tech. rep., IBM.

Iftode, L., Borcea, C., Ravi, N., Kang, P., & Zhou, P. (2004). Smart phone: an embedded system for universal interactions. In *Distributed Computing Systems, 2004. FTDCS 2004. Proceedings. 10th IEEE International Workshop on Future Trends of*, (pp. 88–94).

Jamali, B., Sharp, E., Thorne, A., & Cole, P. (2007). Technology selection for identification applications. Tech. rep., Auto-ID Labs.

Jouault, F., & Kurtev, I. (2006). Transforming models with atl. In *Satellite Events at the MoDELS 2005 Conference, LNCS 3844*, (pp. 128–138). Springer. ISBN=0302-9743.

Ju, W., & Leifer, L. (2008). The design of implicit interactions: Making interactive systems less obnoxious. *Design Issues*, *24*(3), 72–84.

Kagal, L., Finin, T., & Joshi, A. (2003). A policy language for a pervasive computing environment. In *POLICY '03: Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*, (p. 63). Washington, DC, USA: IEEE Computer Society.

Kanjo, E., Bacon, J., Roberts, D., & Landshoff, P. (2009). Mobsens: Making smart phones smarter. *Pervasive Computing, IEEE*, *8*(4), 50–57.

Kephart, J. O., & Chess, D. M. (2003). The vision of autonomic computing. *Computer*, *36*(1), 41–50.

Kindberg, T. (2002). Implementing physical hyperlinks using ubiquitous identifier resolution. In *WWW '02: Proceedings of*

the 11th international conference on World Wide Web, (pp. 191–199). New York, NY, USA: ACM Press.

Kindberg, T., Barton, J. J., Morgan, J., Becker, G., Caswell, D., Debaty, P., Gopal, G., Frid, M., Krishnan, V., Morris, H., Schettino, J., Serra, B., & Spasojevic, M. (2002). People, places, things: Web presence for the real world. *MONET*, *7*(5), 365–376.

Köhler, A., & Gruhn, V. (2004). Analysis of mobile business processes for the design of mobile information systems. In *EC-Web*, (pp. 238–247).

Kramer, J. (2007). Is abstraction the key to computing? *Commun. ACM*, *50*(4), 36–42.

Krogstie, J., Lyytinen, K., Opdahl, A. L., Pernici, B., Siau, K., & Smolander, K. (2004). Research areas and challenges for mobile information systems. *Int. J. Mob. Commun.*, *2*(3), 220–234.

Lampe, M., Strassner, M., & Fleisch, E. (2004). A ubiquitous computing environment for aircraft maintenance. In *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, (pp. 1586–1592). New York, NY, USA: ACM.

Langheinrich, M., Coroama, V., Bohn, J., & Rohs, M. (2002). As we may live – real-world implications of ubiquitous computing. Technical Report.

Li, Y., & Landay, J. (2008). Activity-based prototyping of ubicomp applications for long-lived, everyday human activities. In *ACM Conference on Human Factors in Computing Systems (HCI'08)*, (pp. 1303 – 1312).

Loke, S. W. (2003). Service-oriented device ecology workflows. In *ICSOC*, (pp. 559–574).

Loke, S. W. (2009). Building taskable spaces over ubiquitous services. *IEEE Pervasive Computing*, *8*(4), 72–78.

Mao, J.-Y., Vredenburg, K., Smith, P. W., & Carey, T. (2001). User-centered design methods in practice: a survey of the state of the art. In *CASCON '01: Proceedings of the 2001 conference of the Centre for Advanced Studies on Collaborative research*, (p. 12). IBM Press.

Mayer, R. J., Painter, M. K., & DeWitte, P. (1992). Idef family of methods for concurrent engineering and business reengineering applications. College Station, TX: Knowledge Based Systems, Inc.

Miller, J., & Mukerji, J. (2003). MDA Guide Version 1.0.1. Tech. rep., Object Management Group (OMG).

Mori, G., Paternò, F., & Santoro, C. (2002). Ctte: Support for developing and analyzing task models for interactive system design. *IEEE Transactions on Software Engineering*, *28*(8), 797–813.

Morin, B., Fleurey, F., Bencomo, N., Jézéquel, J.-M., Solberg, A., Dehlen, V., & Blair, G. (2008). An aspect-oriented and model-driven approach for managing dynamic variability. In *MoDELS '08: Proceedings of the 11th international conference on Model Driven Engineering Languages and Systems*, (pp. 782–796). Berlin, Heidelberg: Springer-Verlag.

Muñoz, J., & Pelechano, V. (2005). Building a software factory for pervasive systems development. In *CAiSE*, (pp. 342–356).

Neely, S., Stevenson, G., Kray, C., Mulder, I., Connelly, K., & Siek, K. A. (2008). Evaluating pervasive and ubiquitous systems. *IEEE Pervasive Computing*, *7*(3), 85–88.

OMG (2005). MOF QVT Final Adopted Specification.

OMG (2006). Business Process Modeling Notation (BPMN) Specification. OMG Final Adopted Specification. dtc/06-02-01.

Ouyang, C., Dumas, M., Ter, & van der Aalst, W. M. P. (2006). From bpmn process models to bpel web services. *Web Services, 2006. ICWS '06. International Conference on*, (pp. 285–292).

Pajunen, L., & Chande, S. (2007). Developing workflow engine for mobile devices. In *EDOC '07: Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference*, (p. 279). Washington, DC, USA: IEEE Computer Society.

Pastor, O., & Molina, J. C. (2007). *Model-Driven Architecture in Practice: A Software Production Environment Based on Conceptual Modeling*. Secau-

cus, NJ, USA: Springer-Verlag New York, Inc.

Radi, H., & Mayrhofer, R. (2008). Towards alternative user interfaces for capturing and managing tasks with mobile devices. In *Proc. MoMM 2008: 6th International Conference on Advances in Mobile Computing and Multimedia*, (pp. 272–275).

Recker, J., & Mendling, J. (2006). On the translation between bpmn and bpel: Conceptual mismatch between process modeling languages. In *18th International Conference on Advanced Information Systems Engineering.*.

Rellermeyer, J. S., Duller, M., Gilmer, K., Maragkos, D., Papageorgiou, D., & Alonso, G. (2008). The software fabric for the internet of things. In *IOT*, (pp. 87–104).

Roduner, C., & Langheinrich, M. (2007). Publishing and discovering information and services for tagged products. In *Proceedings of CAiSE 2007, Trondheim, Norway, 11-15 June, 2007*, LNCS. Berlin Heidelberg New York: Springer.

Römer, K., Schoch, T., Mattern, F., & Dübendorfer, T. (2004). Smart identification frameworks for ubiquitous computing applications. *Wirel. Netw.*, *10*(6), 689–700.

Roussos, G., Tuominen, J., Koukara, L., Seppala, O., Kourouthanasis, P., Giaglis, G., & Frissaer, J. (2002). A case study in pervasive retail. In *WMC '02: Proceedings of the 2nd international workshop on Mobile commerce*, (pp. 90–94). New York, NY, USA: ACM.

Rukzio, E. (2007). *Physical Mobile Interactions: Mobile Devices as Pervasive Mediators for Interactions with the Real World*. Ph.D. thesis, Faculty for Mathematics, Computer Science and Statistics.

Rukzio, E., Broll, G., Leichtenstern, K., & Schmidt, A. (2007). Mobile interaction with the real world: An evaluation and comparison of physical mobile interaction techniques. In *AmI*, (pp. 1–18).

Rukzio, E., Leichtenstern, K., & Callaghan, V. (2006). An experimental comparison of physical mobile interaction techniques: Touching, pointing and scanning. In *8th International Conference on Ubiquitous Comput-*

ing, UbiComp 2006. Orange County, California.

Rukzio, E., Pleuss, A., & Terrenghi, L. (2005a). The physical user interface profile (puip): modelling mobile interactions with the real world. In *TAMODIA '05: Proceedings of the 4th international workshop on Task models and diagrams*, (pp. 95–102). New York, NY, USA: ACM.

Rukzio, E., Wetzstein, S., & Schmidt, A. (2005b). A framework for mobile interactions with the physical world. In *Wireless Personal Multimedia Communication (WPMC) conference*. Aalborg, Denmark.

Ruokonen, A., Pajunen, L., & Systa, T. (2008). On model-driven development of mobile business processes. *Software Engineering Research, Management and Applications, ACIS International Conference on*, *0*, 59–66.

Sandner, U., Leimeister, J. M., & Krcmar, H. (2005). Business potentials of ubiquitous computing. Proceedings of the Falk Symposium No. 146. Innsbruck, Austria,.

Sarma, S. (2004). Integrating rfid. *Queue*, *2*(7), 50–57.

Schmidt, D. C. (2006). Guest editor's introduction: Model-driven engineering. *Computer*, *39*(2), 25–31.

Serral, E., Valderas, P., & Pelechano, V. (2008). A model driven development method for developing context-aware pervasive systems. In *UIC '08: Proceedings of the 5th international conference on Ubiquitous Intelligence and Computing*, (pp. 662–676). Berlin, Heidelberg: Springer-Verlag.

Siau, K. (2003). *Advances in Mobile Commerce Technologies*. Hershey, PA, USA: IGI Publishing.

Smith, H., & Fingar, P. (2003). *Business Process Management: The Third Wave*. Meghan-Kiffer Press.

Spieß, P., Bornhövd, C., Lin, T., Haller, S., & Schaper, J. (2007). Going beyond auto-id: a service-oriented smart items infrastructure. *Journal of Enterprise Information Management*, *20*(3), 356 – 370.

Strassner, M., & Schoch, T. (2002). Today's impact of ubiq-

uitous computing on business processes. In F. Mattern, & M. Naghshineh (Eds.) *Short Paper Proc. International Conference on Pervasive Computing*, (pp. 62–74). Pervasive2002.

Tedre, M. (2008). What should be automated? *interactions*, *15*(5), 47–49.

Terman, M., & Terman, J. S. (2006). Controlled trial of naturalistic dawn simulation and negative air ionization for seasonal affective disorder. *Am J Psychiatry*, *163*(12), 2126–33.

Torres, V., Giner, P., & Pelechano, V. (2007a). Modeling ubiquitous business process driven applications. In J. Eder, S. L. Tomassen, A. L. Opdahl, & G. Sindre (Eds.) *CAiSE Forum*. ISSN: 1503-416X.

Torres, V., Giner, P., & Pelechano, V. (2007b). Web application development focused on bp specifications. I Taller sobre Procesos de Negocio e Ingeniería del Software (PNIS 2007).

Torres, V., Giner, P., & Pelechano, V. (2008). From bpmn to bpel4people: A mde approach. In A. V. y. A. R.-C. Jose Manuel López Cobo (Ed.) *IV Jornadas Científico-Técnicas en Servicios Web y SOA*, (pp. 29–41).

Torres, V., Pelechano, V., & Giner, P. (2006). Generación de aplicaciones web basadas en procesos de negocio mediante transformación de modelos. In J. Riquelme, & P. Botella (Eds.) *Ingeniería del Software y Bases de Datos*, (pp. 443 – 452).

Torres, V., Pelechano, V., & Giner, P. (2007c). Building business process driven web applications based on the service oriented paradigm. *ERCIM News*, *70*, 54–55. ISSN: 0926-4981.

Torres, V., Pelechano, V., & Giner, P. (2007d). Generación de aplicaciones web basadas en procesos de negocio mediante transformación de modelos. *IEEE Latin America Transactions*, *5*, 245 – 250.

Urbanski, S., Huber, E., Wieland, M., Leymann, F., & Nicklas, D. (2009). Perflows for the computers of the 21st century. *Pervasive Computing and Communications, IEEE International Conference on*, *0*, 1–6.

van Deursen, A., Klint, P., & Visser, J. (2000). Domain-specific languages: an annotated

bibliography. *SIGPLAN Not.*, *35*(6), 26–36.

Völter, M. (2005). Software architecture patterns – a pattern language for building sustainable software architectures.

Vredenburg, K., Mao, J.-Y., Smith, P. W., & Carey, T. (2002). A survey of user-centered design practice. In *CHI '02: Proceedings of the SIGCHI conference on Human factors in computing systems*, (pp. 471–478). New York, NY, USA: ACM.

Vuolle, M., Tiainen, M., Kallio, T., Vainio, T., Kulju, M., & Wigelius, H. (2008). Developing a questionnaire for measuring mobile business service experience. In *MobileHCI '08: Proceedings of the 10th international conference on Human computer interaction with mobile devices and services*, (pp. 53–62). New York, NY, USA: ACM.

Wang, F., & Liu, P. (2005). Temporal management of rfid data. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, (pp. 1128–1139). VLDB Endowment.

Want, R. (2008). You are your cell phone. *Pervasive Computing, IEEE*, *7*(2), 2–4.

Want, R., Fishkin, K. P., Gujar, A., & Harrison, B. L. (1999). Bridging physical and virtual worlds with electronic tags. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, (pp. 370–377). New York, NY, USA: ACM Press.

Weinstein, R. (2005). Rfid: a technical overview and its application to the enterprise. *IT Professional*, *7*(3), 27–33.

Weiser, M. (1991). The computer for the 21st century. *Scientific American*, *265*(3), 66–75.

Wieland, M., Kaczmarczyk, P., & Nicklas, D. (2008). Context integration for smart workflows. In *PerCom*, (pp. 239–242).

Zhang, J., & Cheng, B. H. C. (2006). Model-based development of dynamically adaptive software. In *ICSE '06: Proceedings of the 28th international conference on Software engineering*, (pp. 371–380). New York, NY, USA: ACM.

Appendix A

Metamodels

The metamodels that describe the design concepts and the architecture used for supporting physical mobile workflows are described in this appendix. The Parkour metamodel formalizes the concepts that were introduced in Chapter 7.1 and the Presto metamodel presents the architectural concepts defined in Chapter 5. Each metamodel has been formalized by means of modeling technologies to describe the concepts involved the relationships among them and the constraints for their composition.

The metamodels have been defined by means of Ecore which is an implementation of the Essential MOF specification that is widely used in the Eclipse community. In this way, these metamodels represent the abstract syntax of a modeling language that can be used for (1) capturing requirements for physical mobile workflows and (2) describe the components used for their implementation. Eclipse-based tools can be used for defining models based on the previous metamodels and assist the designers to avoid inconsistencies. The following sections presents Presto and Parkour metamodels in more detail.

A.1 Parkour metamodel

The Parkour metamodel defines a set of concepts that are useful to characterize how a business process is integrated with physical elements. Figure A.1 depicts the metaelements defined and their relationships by means of a class diagram. These elements are briefly described below.

Named Element. This is an abstract metaelement that is extended by the elements with name.

Parkour System. Represents the system that is supporting a physical mobile workflow. It represents a single system at the requirements level but is projected into multiple software systems when it is translated into the technology domain. This element is normally the root element of Parkour-based specifications.

Task. This element extend the *Activity* element from the BPMN metamodel in order to allow other elements in the Parkour metamodel to be linked to them. In the metamodel it is defined to inherit the properties of both *Named Element* and *Obtrusiveness Element*.

Physical Object. The physical elements involved in the process are represented by means of this element. For each element of the *Physical Object* kind, it is defined the mediums used for representing its identifier.

Physical Interaction. A physical interaction is defined to represent the participation of a *Physical Object* in a process *Task*. The direction of the participation is specified to indicate whether the physical object is minted or captured (see the *Interaction Kind* enumeration defined).

Medium. Mediums can be defined in a hierarchy to define a hierarchy. Each Medium supports a set of interaction techniques (see *Interaction Technique* element).

Technology. The available Auto-ID technologies are described by means of this element. It indicates its Auto-ID capabilities for capturing, minting or both. The support provided for mediums and interaction techniques is described by the *support* feature.

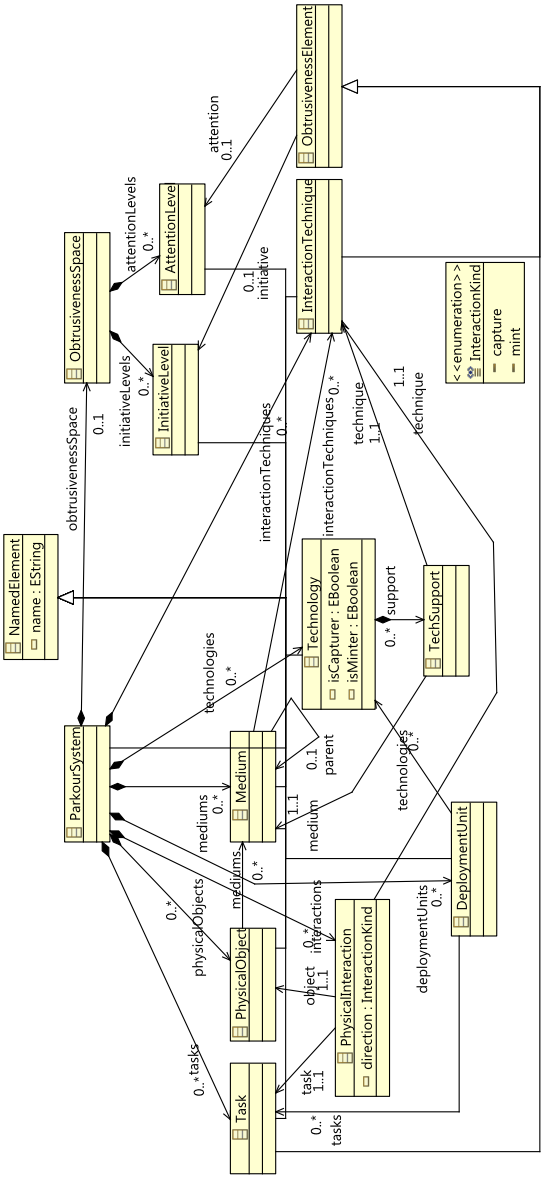


Figure A.1: Parkour metamodel

Tech Support. These metaelements are defined to connect *Medium* elements with *Interaction Technique* elements in order to describe the support provided by each technology.

Deployment Unit. Deployment units represent the support provided for several process tasks by a set of technologies. Each deployment unit can be supported by a different software system.

Obtrusiveness Space. This element describes the obtrusiveness space defined for the process. The ordering is considered for the *initiative* and *attention* levels that are represented.

Obtrusiveness Element. It is an abstract metaclass to represent the elements that can be mapped to the obtrusiveness space described.

Initiative Level. Levels defined by the designers to describe the different degrees in which the system or the user can take the initiative for interacting.

Attention Level. The levels defined by the designers to describe the different degrees in which the system can require the user attention.

The metaelements defined can be used to describe systems in Parkour terms. The following section introduces the constraints defined on these elements to limit their possible combinations.

A.1.1 Constraints

The introduced metamodel limits the possible concepts to use, the information they contain and the way they can be connected to each other. However, for some specific conditions the use of the metamodeling constructs that Ecore provides is not enough. In this section we introduce additional constraints to the metamodel. The constraints have been defined by means of the Check language which complements OCL-like expressions with descriptive messages. In order to simplify the expressions that define the constraints we defined auxiliary functions by means of the extension mechanism provided by the Eclipse modeling tools.

Element name unicity

Modeling elements that inherit from *NamedElement* must be unique in their kind. For example, there cannot exist two *Mediums* with the same name. In addition, the empty string is not considered a valid name. A constraint is defined for each metaelement in order to allow the verification of their siblings. In order to allow the access from one element to their siblings, the container for some elements must be accessible. The listing A.1 shows the extensions over different metaelements in order to access their parent model element. We defined the *parentModel* auxiliary function that provides the immediate container of the element to which it is applied. For all the descendants of *NamedElement*, the parent element is a *ParkourSystem* element. However, this is not the case for the different obtrusiveness levels defined. Thus, the *parentModel* function is overloaded for the *InitiativeLevel* and *AttentionLevel* elements in order to return the proper type (*ObtrusiveneeSpace*).

Listing A.1: Extensions for validating name unicity.

```
ParkourSystem parentModel(NamedElement this) :
  eContainer;

ObtrusivenessSpace parentModel(InitiativeLevel this) :
  eContainer;

ObtrusivenessSpace parentModel(AttentionLevel this) :
  eContainer;
```

Based on the previous extension, constraints that validate the correct naming for the different elements are defined. Listing A.2 shows the different constraints defined.

Listing A.2: Constraints for validating name unicity.

```
context NamedElement ERROR "Invalid name for a "+this.
    metaType.name+" element":
  name.length >= 0;

context Task ERROR "Task '"+this.name+"' name must be
    unique":
  !(this.parentModel().tasks.exists(x|x!=this && x.name ==
      this.name));
```

```
context PhysicalObject ERROR "Physical Object '"+this.name+
    "' name must be unique":
  !(this.parentModel().physicalObjects.exists(x|x!=this &&
      x.name == this.name));

context Medium ERROR "Medium '"+this.name+"' name must be
    unique":
  !(this.parentModel().mediums.exists(x|x!=this && x.name
      == this.name));

context InteractionTechnique ERROR "Interaction Technique '
    "+this.name+"' name must be unique":
  !(this.parentModel().interactionTechniques.exists(x|x!=
      this && x.name == this.name));

context Technology ERROR "Technology '"+this.name+"' name
    must be unique":
  !(this.parentModel().technologies.exists(x|x!=this && x.
      name == this.name));

context InitiativeLevel ERROR "Initiative Level '"+this.
    name+"' name must be unique":
  !(this.parentModel().initiativeLevels.exists(x|x!=this &&
       x.name == this.name));

context AttentionLevel ERROR "Attention Level '"+this.name+
    "' name must be unique":
  !(this.parentModel().attentionLevels.exists(x|x!=this &&
      x.name == this.name));
```

The first constraint forbids the definition of empty names. The rest provide error messages for the different metaelements that require unique names if this condition is not met. Given a modeling element, the *parentModel* function is used to access the container of the current element and the appropriate association is navigated to reach all the elements of this kind to find another with the same name as the original one.

Medium hierarchy must be acyclic

The *parent* relationship in the *Medium* metaclass is intended to represent an inheritance hierarchy. Thus, cycles are not allowed. The *isAcyclic* operation is defined in order to validate that a medium is not found more that once when the parent relationship is traversed. The *isAcyclic* operation is defined as an extension of the Medium element as it is illustrated in the code listing A.3.

Listing A.3: Extensions for validating acyclic hierachy of mediums.

```
Boolean isAcyclic(Medium this):
(this.parent != null && this.parent!=this && this.parent.
    isAcyclic({this})) || this.parent==null;

Boolean isAcyclic(Medium this,Collection[Medium] found) :
  if found.contains(this) then false
  else if this.parent==null then true
  else this.parent.isAcyclic(found.add(this));
```

The *isAcyclic* operation traverses the parent relationship to find whether there is any ancestor that is found more than once. This extension is shown in the listing 5.5. Two overloaded operations are defined to support the search for cycles. The fist represents the base case for the first node explored. This function analyzes the current node and relies on the other function to analyze their ancestors. The second one takes into account the previous results and propagates the execution to the next ancestor until some child element is found or there are not more ancestors. If there is a repeated element, a cycle has been found in the inheritance hierarchy.

The Check constraint defined in listing A.4 makes use of the *isAcyclic* operation to provide an error message.

Listing A.4: Constraint for validating acyclic hierachy of mediums.

```
context Medium ERROR "The medium hierarchy must be acyclic.
     Medium '"+ name + "' cannot be set as a child of " +
    parent.name +".":
  this.isAcyclic();
```

Interaction techniques supported by the medium

For each interaction with the real world an interaction technique must be selected. However, we must take into account that the interaction technique selected must be applicable to the physical element involved in the interaction. This is determined by the mediums used for the identification of the *Physical Object*. Physical interactions defined over an element should make use of one of the techniques that these medium supports. For example, it makes no sense to define a *pointing*-based interaction over a book that is labelled using the *numbers on paper* medium.

Considering that the mediums can be part of an inheritance hierarchy, determining the supported interaction techniques for a medium requires to explore the ones supported by their ancestors. Listing A.5 shows the *supportsTechnique* operation. This operation traverses the medium hierarchy to gather all the interaction techniques that are associated with a given medium.

Listing A.5: Extensions for validating that a medium supports the required interaction techniques.

```
Boolean supportsTechnique(Medium this, InteractionTechnique
    inter):
  this.interactionTechniques.contains(inter) || (this.
    parent != null? this.parent.supportsTechnique(inter):
     false);
```

Based on the previous extension, we define a constraint (see listing A.6). This constraint explores the different mediums that are supported by the PhysicalObject that is involved in the interaction and checks whether some of this mediums supports the desired interaction technique.

Listing A.6: Constraint for validating acyclic hierachy of mediums.

```
context PhysicalInteraction WARNING "Interaction technique
    not supported by the mediums used for '"+this.object.
    name+"' object":
 this.object.mediums.exists(e|e.supportsTechnique(this.
    technique));
```

We defined this constraint as a warning. The severity of the constraint was lower because the business process described can be still supported despite of the detected problem. The warning tells the designer that it is not possible to apply the particular interaction pattern he desired and a different one will be used instead. This problem may be imposed by the use of a certain technology, so we believe that it is a good design decision to reflect the desired situation in order to detect whether a new technology can improve the current business process to be performed as the designers defined.

Technology support for the deployment unit

The technologies that support a deployment unit must be capable of supporting the tasks that are assigned to this deployment unit. This constraint is useful to determine whether a given mobile device (considering the technologies it includes) can be used to support a given set of tasks from the business process.

The listing A.7 defines different auxiliary operations to determine the compatibility between the tasks and the set of technologies associated with a *DeploymentUnit* element.

Listing A.7: Extensions for validating deployment unit consistency.

```
List[PhysicalInteraction] interactions(Task this):
 this.parentModel().interactions.select(i|i.task==this);

Boolean canBeSupportedBy(PhysicalInteraction this, List[
    Technology] techList):
  let tech = this.direction==InteractionKind::capture?
      techList.select(e|e.isCapturer): techList.select(e|e.
      isMinter) :
  tech.support.exists(s|this.object.mediums.contains(s.
      medium) && s.technique == this.technique);

Boolean canBeSupportedBy(Task this, List[Technology]
    techList):
  this.interactions().forAll(e|e.canBeSupportedBy(techList)
      );
```

The first operation (*interactions*) is used to traverse the relation-

ship between the *PhisicalInteraction* and the *Task* metaelements in the opposite direction. The *interactions* operation defined over the *Task* metaelement makes use of the *parentModel* extension (previously defined in listing A.1) to obtain all the interactions that are associated with the current task.

The *canBeSupported* operation is defined over *PhysicalInteraction* and *Task* metaelements to determine whether a list of technologies (received as a parameter) is supported by the element. In the case of *PhysicalInteraction* elements, technologies are filtered according to their support for capturing or minting identifiers. Then, the remaining technologies are evaluated to determine whether the supported medium and interaction technique match the *PhysicalInteraction* element. in the case of *Task* elements, the *canBeSupported* operation is applied to all the physical interactions that are involved in the current task (which is obtained by applying the operations previously defined).

The constraint defined in the listing A.8 is defined for the *DeploymentUnit* element and makes use of the *canBeSupported* operation in order ot validate that the tasks supported by the deployment unit can be supported by the technologies assigned to the unit.

Listing A.8: Constraint for validating deployment unit consistency.

```
context DeploymentUnit ERROR "The technology selected
    cannot support the tasks in the deployment unit '"+this
    .name+"':
  this.tasks.forAll(e|e.canBeSupportedBy(this.technologies)
      );
```

Inconsistencies in the obtrusiveness level

It is desirable that all the interactions that support a given task are performed with a technology that is capable to provide the obtrusiveness level required by the task. However, due to technology limitations it is not possible to reach the desired level of obtrusiveness. Thus, we qualified this constraint as a warning in order to allow the development of sub-optimal solutions but keeping explicit in the model the rationale that enables further improvements. The listing A.9 shows the expression

defined.

Listing A.9: Constraint for validating the consistency for the obtrusiveness level.

```
context Task WARNING "Inconsistencies in the obtrusiveness
    level":
  this.interactions().technique.forAll(e|
    (e.initiative==null || this.initiative==null || e.
        initiative==this.initiative) &&
    (e.attention==null || this.attention==null || e.
        attention==this.attention));
```

This constraint makes use of the *interactions* extension defined in the listing A.7. In the case that no obtrusiveness level is defined, the constraint is not enforced.

A.1.2 Tool support

The use of Eclipse Modeling technologies for the formalization of the Parkour metamodel allows the use of different modeling technologies. The Eclipse Modeling Project provides a customizable editor that allows to create model instances based on the defined metamodel. The constraints can be applied either in a batch mode (to validate an existing model) or in the editor.

The generic edition capabilities suffer from some usability problems compared to ad-hoc editors. However, we have defined a set of additional constraints to improve the tool support for the creation of Parkour models.

Undefined properties

Some modeling elements are not much useful to the overall system is some information is lacking. For example, a Medium that does not support any interaction technique is not very useful for bridging the physical and virtual spaces. However, it can be useful to organize some sub-elements in the medium hierarchy as it is the case of the paper element. Warning the designer about these elements can avoid some aspects to be left unspecified by mistake.

The listing A.10 shows the *supportsTechnique* operation defined for the *Medium* metaelement. This operation determines whether a medium is supporting any interaction technique.

Listing A.10: Extensions for validating undefined properties.

```
Boolean supportsTechnique(Medium this):
  (!this.interactionTechniques.isEmpty) || (this.parent !=
     null? this.parent.supportsTechnique(): false);
```

The listing A.11 shows different constrains that detect undefined information that can potentially represent a mistake. All these constraints raise a warning to the designer.

Listing A.11: Constraint for validating undefined properties.

```
context Medium WARNING "Medium '"+this.name+"' is not
    supporting any interaction technique.":
  this.supportsTechnique();

context Technology WARNING "Technology '"+this.name+"' is
    not supporting any interaction technique.":
  !(this.support.technique.isEmpty);

context Technology WARNING "Technology '"+this.name+"' is
    not supporting any medium.":
  !(this.support.medium.isEmpty);
```

The first constraint makes use of the *supportsTechnique* operation previously defined in order to detect Mediums that are not supporting any interaction technique. The second constraint detects a *Technology* that is not supporting any interaction technique. Finally, the third constraint detects technologies that do not provide support to any medium.

Usability changes

In order to improve the default edition capabilities for Parkour models, we defined the way in which the element labels are rendered for some elements. By default, the name is used for all the elements that inherit from *NamedElement*. However, other elements represent connections between elements and their identity depends on the state of related

elements. The listing A.12 shows the expressions used to obtain the label for *TechSupport* and *PhysicalInteraction* metaelements.

Listing A.12: Label definition for some elements.

```
import parkour;

String label(TechSupport this):
if this.technique!= null && this.medium !=null then
  "Support for "+this.technique.name+" on "+this.medium.
      name
else
  "Support for...";


String label(PhysicalInteraction this):
let action=
  if this.direction==InteractionKind::capture then
    "Capture"
  else "Mint":
if this.task!=null && this.object!=null
then action+" "+object.name+" for "+task.name+" task"
else action;
```

The *TechSupport* metaelement represents the support of a given technology for an interaction technique by using a medium. For example, an RFID reader is capable to support the scan interaction mechanism over the radio medium. This support is labeled as "Support for scan on radio".

For the *PhysicalInteraction* metaelement, the direction of the interaction is reflected in the label in addition to the object and the task being supported. For ecample, the interaction performed for identifying a book for performing a loan is labeled as "Capture book for borrow book task".

A.2 Presto metamodel

The Presto metamodel provides a definition of the architectural concepts for supporting the user participation in physical mobile workflows. Figure A.2 depicts the metaelements defined and their relationships by

means of a class diagram. These elements are briefly described below.

Named Element. This is an abstract metaelement that is extended by the elements with name.

Presto Model. Represents a set of Presto-based systems that support different processes. This element is normally the root element of Presto-based specifications.

Presto System. This metaelement represents a system based on the Presto platform. It represents a single system that can be deployed to a computing device.

ID Component. A pluggable component of the architecture. It represents Identification Components and describe their capabilities for identification. Identification Components inherit from the *Service* element.

Data Provider. A pluggable component of the architecture. Data Providers are organized in different domains and provide a reference to a schema that captures the structure of the data they provide.

Task Processor. A pluggable component of the architecture. It provides support to a task from a specific process. *Task Processors* can be qualified as silent or initiator depending on the way that provides its functionality.

Process. Processes are used to group *Task Processors* and organize *Domain* elements. the service which is in charge of the process orchestration is indicated in the *supportedBy* reference.

Domain. Represents a set of digital elements that are provided according to a specific perspective with respect to the physical world.

Service. The service element is included in the metamodel to represent functionality of any kind that is used by other components in the system. It is an adaptable element.

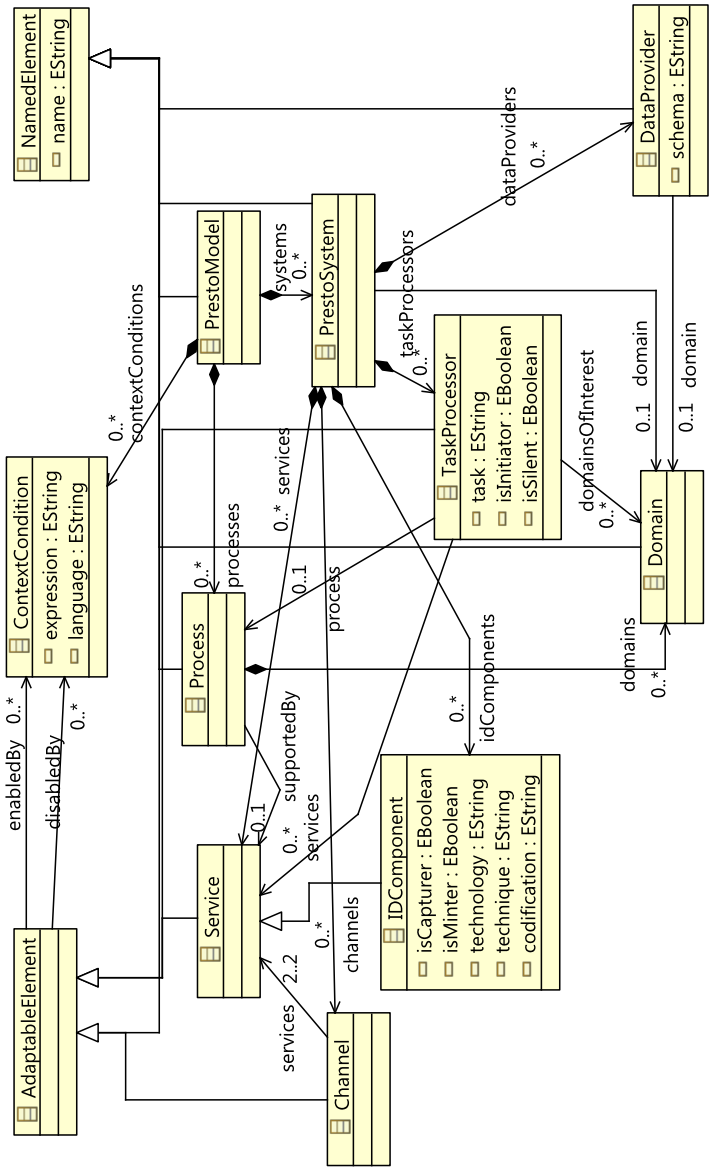


Figure A.2: Presto metamodel

Channel. Represents the possible information exchanges between two Service elements. Services and channels are the basic pieces used to define reconfigurable architectures.

Adaptable Element. This is an abstract metaclass that is used to define the elements that can be enabled and disabled depending on context events. In particular this is the case of *Service*, *Channel*, *Task Processor* and *ID Component* elements.

Context Condition. Defines a specific context condition. The context condition is an expression defined in any query language supported by the underlying platform.

The metaelements defined can be used to describe systems in Presto terms. The following section introduces the constraints defined on these elements to limit their possible combinations.

A.2.1 Constraints

In this section we introduce additional constraints to the metamodel defined. The constraints have been defined by means of the Check language. More detail on the constraints defined is provided below.

Element name unicity

We have enforced the unicity for each kind of elements in the Presto architecture. The degree in which two elements can overlap varies in each case. For some elements it is not allowed to have the same name but others can as long as they meet some constraints (e.g., participate in separate systems). Since these constraints normally require to navigate the model, some extensions have been defined over Presto in order to facilitate this task. In particular, listing A.13 shows the definition of the *parentModel* extension to navigate containment relationships for some of the elements in the Presto metamodel.

Listing A.13: Constraints for validating name unicity.

```
PrestoModel parentModel(PrestoSystem this) :
  eContainer;
```

```
PrestoModel parentModel(ContextCondition this) :
  eContainer;

PrestoModel parentModel(Process this) :
  eContainer;


PrestoSystem parentModel(IDComponent this) :
  eContainer;

PrestoSystem parentModel(TaskProcessor this):
  eContainer;

PrestoSystem parentModel(DataProvider this):
  eContainer;

PrestoSystem parentModel(Channel this):
  eContainer;

PrestoSystem parentModel(Service this):
  eContainer;

Process parentModel(Domain this) :
  eContainer;
```

Based on the previous extension, constraints that validate the correct naming for the different elements are defined. For each element, their siblings are accessed and some properties are verified in order to find any inconsistency. Listing A.14 shows the different constraints defined.

Listing A.14: Constraints for validating name unicity.

```
context NamedElement ERROR "Invalid name for a "+this.
    metaType.name+" element":
  name.length >= 0;

context PrestoSystem ERROR "PrestoSystem '"+this.name+"'
    name must be unique":
  !(this.parentModel().systems.exists(x|x!=this && x.name
      == this.name));
```

```
context DataProvider ERROR "DataProvider '"+this.name+"'
    name must be unique in a domain":
  !(this.parentModel().dataProviders.exists(x|x!=this && x.
      name == this.name && x.domain==this.domain));

context IDComponent ERROR "IDComponent '"+this.name+"' name
     must be unique for an interaction technique and
    technology":
  !(this.parentModel().idComponents.exists(x|x!=this && x.
      name == this.name && x.technique == this.technique &&
       x.technology == this.technology));

context TaskProcessor ERROR "TaskProcessor '"+this.name+"'
    name must be unique in a process for tasks that are not
     context-sensitive":
  !(this.parentModel().taskProcessors.exists(x|x!=this && x
      .task == this.task && (x.disabledBy.isEmpty || x.
      enabledBy.isEmpty)  ));
```

The first constraint forbids the definition of empty names. Since *Presto Systems* are identified y two systems with the same name are not allowed. Conversely, *Data Providers* are allowed to share the name provided that they belong to different domains. *Identification Components* are considered to be duplicated when they have the same name and they use the same technology and identification technique since this is the information that is used for presenting the components to the user. Finally, multiple *Task Processors* can only be defined for the same task and process if it represents variants for the process adaptation. That is, the defined constraints verify that the *Task Processor* variants are associated with a valid *Context Condition*.

Identification functionality

The *Identification Components* defined in Presto architecture represent a communication channel between the physical and the digital world that can be supported in one or both directions. The constraint defined in listing A.15 validates that the Identification Component defined provides at least one identification function.

Listing A.15: Constraint for the Identification Component functionality.

```
context IDComponent ERROR "Identification Component " +
    this.name + " must provide one identification function
    at least":
  isMinter || isCapturer;
```

Generic and specific services

The Presto metamodel defines the Service metaclass to allow the definition of generic functionality that other components could require. in addition, *Identification Components* are also defined as Services for the sake of homogeneity. However, it must be checked that the services list contains only generic services since the generator expects *Identification Components* to be in the service list of each *Presto System*. The listing A.16 shows the extension made to the metamodel to distinguish generic and specific services.

Listing A.16: Constraint for the Identification Component functionality.

```
Boolean plainServices(PrestoSystem this):
this.services.forAll(e|e.isPlainService());

Boolean isPlainService(Service this):
true;

Boolean isPlainService(IDComponent this):
false;
```

The *plainService* extension makes use of polymorphism to differentiate generic services from those that have a specific role in the Presto architecture as it is the case of *Identification Comonents*. Listing A.19 makes use of the previous extension to detect the presence of Identification Components on the service list.

Listing A.17: Constraint for the Identification Component functionality.

```
context PrestoSystem ERROR "The services feature must not
    include Identification Components":
  plainServices();
```

A.2.2 Tool support

Presto models are not intended for being built from scratch. Normally a Presto model is obtained as a result from a transformation that is applied to a Parkour model. A valid Parkour model will generate a valid Presto model. However, designers can further modify the resulting Presto model and introduce some inconsistent information. The constraints defined in the previous section are useful to avoid these inconsistencies.

In addition to the previous constraints aid was provided for designers in detecting situations that although possible they are not likely to occur very often. In this way, designers can decide whether what they have described is what they intended. In particular we have introduced a constraint for detecting minters that produce identifiers in a manner for which there is not an appropriate capturer defined. The listing A.18 defines the *existsComplementary* extension which is defined over *Identification Components*.

Listing A.18: Extension to find incompatibilities among Identification Components.

```
Boolean existsComplementary(IDComponent this):
if this.isMinter then
this.parentModel().idComponents.exists(e| e.isCapturer ==
    true &&
     e.technology == this.technology &&
     e.codification == this.codification)
else true;
```

The previous extension is applied for minters and looks for some capturer with the same technology and codifications. Note that capturers do not require a minter to be defined since it is common to find objects that are tagged from outside of the scope of the workflow defined. Listing A.19 shows the warning message defined for the minters for which there is no complementary capturer.

Listing A.19: Constraint to find incompatibilities among Identification Components.

```
context IDComponent WARNING "There is no capturer that can
    access " + this.name + " minter":
```

```
    existsComplementary() ;
```

Appendix B

Experimental results

This appendix provides detail on the experimental results that were obtained when applying our approach with end-users. The services defined in the Smart Library case study were evaluated by means of the MoBiS-Q (Vuolle et al., 2008) which is a questionnaire to measure mobile business service experience. Chapter 7 provides more detail regarding the experimental set-up and the case study.

The appendix is structured according to the MoBiS-Q questionnaire sections: *perceived usability of a mobile business service*, *fit for mobile working context*, and *perceived impact on mobile work productivity*. For each of the sections, we present the user answers to each particular topic and the comments they provided[1]. Furthermore, an additional set of questions was included in the questionnaire to evaluate the level of realism achieved during the experiment and to compare with their previous experience with similar services. More detail of each of these topics is provided below.

[1] Users provided feedback in Spanish but it has been translated to English for the publication in this work.

Item	Strongly disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree	Total
Easy to learn	-	2.9% (1)	-	55.9% (19)	41.2% (14)	34
Easy to become skilful	-	-	-	47.1% (16)	52.9% (18)	34
Suitable for work tasks on the move	-	-	2.9% (1)	47.1% (16)	50.0% (17)	34
Quick enough	-	2.9% (1)	8.8% (3)	14.7% (5)	73.5% (25)	34
Functions are necessary	-	-	11.8% (4)	35.3% (12)	52.9% (18)	34
Ease of navigation	-	2.9 (1)	14.7% (5)	29.4% (10)	52.9% (18)	34
Average %	0.0%	1.5%	6.4%	38.2%	53.9%	204

Figure B.1: Results for the "perceived usability of a mobile business service" dimension

B.1 Perceived usability of a mobile business service

This dimension of the questionnaire analyzes the functionality provided to the user. Users were asked to focus on the services provided by the system. Figure B.1 shows the user answers to the different topics defined in this section of the questionnaire.

B.1.1 Feedback from users

For each of the sections of the questionnaire, users were allowed to write some comments. This feedback was useful for redesigning the system in different iterations. The comments provided by users are listed below.

- Guidance to locate similar books is lacking.

- The level of detail for pending tasks could be greater (e.g., the date of loan and return, as well as to the possibility of extending the loan period).
- It could be interesting to provide a list of the books that are on a temporal reservation in order to manage them (e.g., cancel reservations).
- Some buttons (specially the sense, ok and back buttons) are too small. Feedback could be provided about the remaining time for the temporal reservation.
- I would prefer bigger buttons for the user interface. It took me some time to feel comfortable with the device, maybe it would not be the case with my own phone.
- I had never used a Smartphone before, but it is simple to do. Some icons could make the available actions more intuitive.
- It is difficult to distinguish between actions and pending tasks at first.
- The temporal reservation feature may not be very useful. A better location mechanism would be required for finding books.
- There were some connectivity problems at the beginning of the experiment.
- I think it is a very useful application and it is easy to use. We can avoid long queues and find books of our interest easily. I found no problems with it.
- I am not used to this kind of devices. Nevertheless, it seems easy to start using them and the services provided do not require much dexterity from the user.
- The navigation was sometimes misleading: I was not sure which book I was working with for moments.
- All I can say about the system is positive. I was impressed, I like it a lot.

Item	Strongly disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree	Total
Ease of use with a device	-	2.9% (1)	11.8% (4)	29.4% (10)	55.9% (19)	34
Screen size is not a limit	-	-	17.6% (6)	35.3% (12)	47.1% (16)	34
Suitable for working on the move	-	-	-	23.5% (8)	76.5% (26)	34
Using a device with one hand	5.9% (2)	8.8% (3)	2.9% (1)	32.4% (11)	50% (17)	34
Ease of use while on the move	2.9% (1)	2.9% (1)	2.9% (1)	23.5% (8)	67.6% (23)	34
Ease of use in a hurry	-	5.9% (2)	26.5% (9)	41.2% (14)	26.5% (9)	34
Average %	1.5%	3.4%	10.3%	30.9%	53.9%	204

Figure B.2: Results for the "fit for mobile working context" dimension

B.2 Fit for mobile working context

This dimension of the questionnaire focuses on the use of the service through the mobile device. Users were asked to focus on the way in which the use of the mobile device affected the services provided by the system. Figure B.2 shows the user answers to the different topics defined in this section of the questionnaire.

B.2.1 Feedback from users

For each of the sections of the questionnaire, users were allowed to write some comments. The comments provided by users are listed below.

- I have used the mobile device with one hand since I am an iPhone

user.

- When the book location is shown, the ok button is hard to see. Maybe the device could be difficult to use for people with fat fingers.

- I thing that the device is required to be too close to the book for detecting it, especially when there were many books close to the one being detected. In my case there were also connectivity problems that produced some delays.

- Once you have used the service several times it becomes is easy and intuitive.

- The ability for working in a rush depends on how familiar you are with the device.

- There are not much affordable phones in the market that are advanced enough to use the services.

- Screen size is not limiting the functionality. Some buttons are small.

- It seems very practical. The screen was big enough to interact with the service without making input mistakes.

- Touch screen was not 100% responsive.

- A better recommending system can be used to offer similar books to the user.

- I had no previous experience with Smartphones and it has been difficult in the beginning.

- It is still a small device. Although it could be perfect for the service, it is impossible to operate fast.

Item	Strongly disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree	Total
Satisfaction with efficiency at work	-	-	5.9% (2)	38.2% (13)	55.9% (19)	34
Use improves fluidity of work	-	-	5.9% (2)	52.9% 18	41.2% 14	34
Able to perform tasks in less time	-	2.9% (1)	2.9% (1)	17.6% (6)	76.5% (26)	34
Able to complete tasks easier	2.9% (1)	-	8.8% (3)	38.2% (13)	50% (17)	34
Less time to go through tasks	-	-	2.9% 1	20.6% 7	76.5% 26	34
Less additional traveling	-	5.9% (2)	2.9% (1)	32.4% (11)	58.8% (20)	34
Better access to information needed			5.9% (2)	47.1% (16)	47.1% (16)	34
Average %	0.4%	1.3%	5.0%	35.3%	58.0%	238

Figure B.3: Results for the "perceived impact on mobile work productivity" dimension

B.3 Perceived impact on mobile work productivity

This dimension of the questionnaire is about improvements in the fluidity of work. Users were asked to focus on the way in which the tasks they normally perform in the library are improved. Figure B.3 shows the user answers to the different topics defined in this section of the questionnaire.

B.3.1 Feedback from users

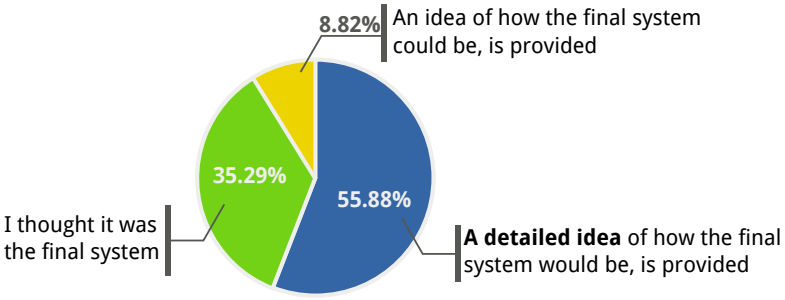
For each of the sections of the questionnaire, users were alowed to write some comments. The comments provided by users are listed below.

- The system seems a universal remote control providing the information needed at each moment.
- Temporal reservation does not avoid walking since users must go to the shelf to take the book.
- More simultaneous tasks would be required to really perceive the productivity increase.
- It would be useful to also integrate pure digital services such as looking books by author or name.
- I found the possibility of finding similar books very useful. The only problem I found is that this is only offered for books you can access from the shelf.
- I found no problems. It is intuitive and fast to operate.
- I have compared the productivity with respect to the usal libraries.

B.4 Additional questions

We included in the evaluation some questions to determine the relevance of the feedback obtained. One of the questions was related to how much the prototype provided a detailed idea of the final system. In this way we could determine how close it was the user experience evaluated from th eone that could be obtained by using a final system. The results obtained are depicted in Fig. B.4. These results show our prototyping technique was capablo of reproducing a level of user experience that is very close to what users were expecting from a final system.

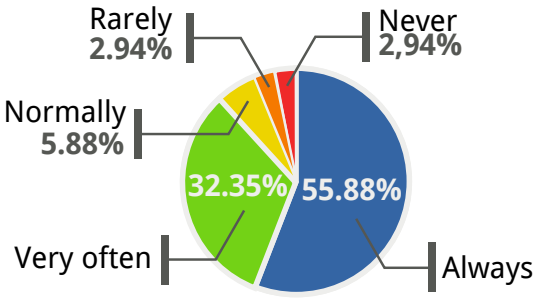
The other question was intended to estimate the acceptance for services such as the one provided. We asked how often they would use the service if it were available in their library. The results obtained are



Item	Count	Percent
It seems not to be useful for the development of the final system	0	0%
An idea of how the final system could be, is provided	3	8.82%
A detailed idea of how the final system would be, is provided	19	55.88%
I thought it was the final system	12	35.29%

Figure B.4: Evaluating the realism of the system

depicted in Fig. B.4. The results show that the service was perceived as very useful and people showed interest in using it.



Item	Count	Percent
Never	1	2.94%
Maybe	1	2.94%
Normally	2	5.88%
Often	11	32.35%
Always	19	55.88%

Figure B.5: Evaluating the acceptance for the services

www.pros.upv.es

Centro de Investigación en Métodos
de Producción de Software
Universidad Politécnica de Valencia
Camino de Vera s/n
Building 1F
46007 Valencia
Spain
Tel: (+34) 963 877 007 (Ext. 83530)
Fax: (+34) 963 877 359

UNIVERSIDAD
POLITECNICA
DE VALENCIA