Pau Giner Blasco

Business Process Modeling for the Internet of Things

Master's Thesis

September 14, 2008

UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Supervisor:
Dr. Vicente Pelechano Ferragud

To my grandmother.

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 Domain Specific Language is defined to cope with the particular requirements of the Internet of Things domain. From this specification, following a set of guidelines, 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.

Different case studies have been successfully developed to validate the applicability of the proposal. 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 Específico de Dominio (DSL) adaptado a los requisitos de la "Internet de las Cosas". A partir de esta especificación se puede obtener una solución software siguiendo una serie de directrices.

Como validación de la propuesta, varios casos de estudio han sido desarrollados con éxito. 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 Específic de Domini (DSL) adaptat als requeriments de l'"Internet de les Coses". A partir d'aquesta especificació és possible obtenir una solució de programari seguint una sèrie de directrius.

Com a validació de la proposta, diversos casos d'estudi s'han desenvolupat amb èxit. 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.

Preface

When I started writing *Prosopagnosia*, a humble short story around the ideas of identity and change, I could never imagine that these concepts would become such a central part of my research. From the time perspective, it is not so strange; these universal concepts have occupied the mind of humans for long. From times of the ancient Greeks, identity and change have been a subject of interest for philosophers, as reported by Plutarch in *Lives of the Noble Greeks and Romans*:

> "The ship wherein Theseus and the youth of Athens returned had thirty oars, and was preserved by the Athenians down even to the time of Demetrius Phalereus, for they took away the old planks as they decayed, putting in new and stronger timber in their place, insomuch that this ship became a standing example among the philosophers, for the logical question as to things that grow; one side holding that the ship remained the same, and the other contending that it was not the same."

We live in a changing world. Nowadays, Business Processes in organizations –like the Ship of Theseus– never stop changing. They involve different real-world elements participating in this identification paradox. Technology can alleviate the problem, but only from a conceptual level –where the problem has its roots– it can be addressed. Having the opportunity to face such a transcendent issue with this work is quite challenging.

Valencia, September 2008 *Pau Giner*

Acknowledgements

This work would not have been possible without the support and encouragement of many people.

Foremost, I would like to thank my supervisor, Dr. Vicente Pelechano, who shared with me a lot of his expertise and research insight. His unconditional enthusiasm in my work has been quite motivating. I also like to express my gratitude to Prof. Oscar Pastor for his direction of our research center and for transmitting me his sane capability of self-improvement.

A special gratitude is due to Vicky who first brought me into the world of research. Her fantastic qualities as researcher and person make working with her a pleasure.

I would also like to express my thanks to Joan, Pedro, Manoli and Javi for their invaluable help. I wish to thank Carlos and Fani for being such incredible partners in this journey that research is. Thanks to Paqui and Mario for their trust in me when I directed their work.

I am grateful to Ana for all the time she saved me thanks to her effort in making bureaucracy almost invisible.

Special thanks for Bea, Giovanni, Paco and Nathalie for all the funny moments we have shared, showing that they are as good friends outside the university as good partners inside. I would also like to thank the rest of friends and colleagues from the ProS research center – Ignacio, Luis, Jose Luis, Sergio, Gonzalo, Isabel, Matilde, Juan Carlos, Juan, Jorge and Nelly– for their collaboration.

Finally, 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.

Contents

List of Figures

List of Tables

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. So, there is still a challenge in **automating the linkage between digital and physical worlds**.

Nowadays, Information Systems dealing 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 reducing the gap between physical and virtual worlds (Strassner & Schoch, 2002).

The high heterogeneity in identification technologies and the fast-changing nature of business processes makes hard to develop Information Systems for the Internet of Things in a sound manner. This thesis provides a development method for building this kind of systems from a model driven perspective. The benefits of modeling technologies are quite interesting for the purpose of

this work. Using models, the development process is faced from a higher abstraction level and the construction of the final product can be systematized.

The rest of this chapter is organized as follows: Section 1.1 explains the purpose of this work. Section 1.2 details the problem this thesis resolves. The goals defined for the present work are described in Section 1.3. In Section 1.4, the approach followed in this thesis to fulfill the detected goals is introduced. 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 thesis.

1.1 Motivation

Business processes in organizations usually involve real-world objects. Information Systems need mechanisms to connect those elements at the physical space with the information about them at the digital space. Maintaining the connection between both, real and virtual elements, in sync is thus essential. Baggage loss or medicine counterfeiting illustrate the consequences of breaking this linkage.

In addition, a poor integration of physical elements usually produces process bottlenecks. This lack of efficiency can be perceived at motorway tolls, supermarket or library queues. Given several persons forming a queue, it is easy to find that the queue is originated by a human transferring data to an information system[1].

When physical objects participate actively in business processes, the gap between physical and virtual worlds is reduced. 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.

Integrating real-world objects in business process 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). When real-world

[1] Being probably toilet queues the most common exception.

objects are seamlessly integrated, business process-supporting tools become a true invisible tool in the sense defined by Mark Weiser (Weiser, 1994):

> "A good tool is an invisible tool. By invisible, I mean that the tool does not intrude on your consciousness; you focus on the task, not the tool. Eyeglasses are a good tool – you look at the world, not the eyeglasses."

Facing the development of this kind of tools, however, is not an easy task. Business processes are constantly changing, demanding 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 it 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. The work that has been done in this thesis is an attempt to improve the development of business process-supporting applications in the context of the Internet of Things by considering these problems, which can be stated by the following three research questions:

Research question 1: How should real-world elements be integrated in business processes?

Research question 2: How should business process specifications be defined during the development?

Research question 3: How can business process specifications be systematically mapped to technological solutions that fulfill the requirements of the Internet of Things?

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

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.

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 from a wide perspective. In the present work, the integration of real-world elements in business processes is faced from three different points of view. When studying this integration, it is considered how real-world elements influence (1) the information present at the digital space, (2) the evolution of the business process and (3) the surrounding environment.

Regarding **research question 2**, another goal of this work is to avoid dealing with technological details when capturing business process requirements for the Internet of Things. This work defines a method that captures the requirements of this particular kind of applications by means of modeling techniques. Modeling standards from the business process area such as Business Process Modeling Notation are extended to cope with the integration of real-world elements in business processes.

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.

1.4 The Proposed Solution

Industrial revolution supposed a transition from the handcraft of products to an automated production process. Software industry, conscious of the increase of system complexity, demands systematic development methods that ensure the quality of the final product. In order to improve the way applications for the Internet of Things are developed, this work proposes a development process to systematize the construction of such applications. The presented development method is based on modeling to achieve these goals.

Model Driven Engineering (MDE) (Kent, 2002) 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** prior to its construction.

MDE can help to handle better system complexity. By using models, some of the current problems in software development can be addressed. Schmidt analyzed (Schmidt, 2006) the current problems in software development derived from the growth of platform complexity and the inability of general-purpose languages to mask complexity. The fast evolution of platforms and the appearance of new ones is translated in a great effort to manually build and port applications. When working with large-scale distributed systems, the lack of an integrated view –coupled with the danger of unforeseen side effects– often forces developers to implement suboptimal solutions that unnecessarily duplicate code, violate key architectural principles, and complicate system evolution and quality assurance. To face those problems Schmidt proposes MDE as a solution to alleviate software complexity.

Both factors analyzed by Schmidt –heterogeneity in platforms and the distribution of the system– apply in a great measure when the Internet of Things is involved. It is easy to imagine that those problems would really affect a system handling thousands of physical objects identified by means of different technologies with diverse computing resources involved. Therefore, MDE becomes a good candidate for the development of the Internet of Things.

This work proposes a MDE method for the production of business process-supporting systems in the Internet of Things domain. This have been achieved following these steps:

1. **Architecture definition:** a software architecture is defined to cope with the particular requirements of the Internet of Things. Although it constitutes a particular solution, architectonic concepts have been detected in a technology-independent fashion.
2. **Development automation:** architectural concepts are formalized using modeling techniques and the development for this architecture is automated as much as possible.

3. **Development process definition:** Once defined the primitives that capture the requirements for this kind of systems, a development process is defined.

This constitutes an architectural-centric bottom-up approach. The result is a model-based development process that systematizes the production of business process-supporting systems for the Internet of Things.

1.5 Thesis Context

This Master's Thesis was developed in the context of the research center *Centro de Investigación en Métodos de Producción de Software* of the *Universitat Politècnica de València*. 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.

1.6 Thesis Structure

The approach followed in this work involves raising the abstraction level in development. The work has been structured to reflect this abstraction process. First, Chapter 2 gives an overview of some relevant concepts related to Business Process Management and Automatic Identification technologies in which this work relies. Chapter 3 defines a software architecture that fulfills the requirements for the kind of applications faced in this work. Chapter 4 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. A Domain Specific Language is defined to capture requirements for this kind of applications.

Chapter 5 introduces a development process that covers from the specification of the system to the implementation of the final solution by following a sequence of systematic steps. Chapter 6 details how the proposal has been validated. Finally, Chapter 7 summarizes the contributions and provides some insights about further work.

2

Related Work

The present work is placed in the intersection of two disciplines. On the one hand, the Internet of Things envisions a seamless integration of real and virtual worlds. On the other hand, Business Process Management is promoting an engineering approach for the analysis design and execution of business processes in organizations. The present chapter provides an overview of the state of the art in these areas reviewing trends at both, technological and conceptual levels.

The rest of this chapter is organized as follows: Section 2.1 presents the Internet of Things vision and the existing support in terms of technologies, frameworks and languages. Section 2.2 provides an overview of the Business Process Management initiative with special attention to business process modeling. Finally, Section 2.3 concludes the chapter.

2.1 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 labelled 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. Augmenting with new capabilities the real-world objects that surround us implies a big change. Since the Internet of Things supposes a social revolution, it is not clear that it could be fully defined while being at the middle of the change. Therefore, 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. This is what Schmitt defines as *Ambient Business* (Schmitt et al., 2006). 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. Some prototypes have been developed to experiment its benefits in contexts as grocery retail (Roussos et al., 2002), aircraft maintenance (Lampe et al., 2004) and vineyard control (Brooke & Burrell, 2003). Experimental developments are 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.

2.1.1 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 centimetre 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.

2.1.2 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.

Global Sensor Network. This work (Aberer et al., 2006) comes from the sensor networks area. Global Sensor Network (GSN) is a middleware for connecting different data sources providing zero-programming deployment is defined. The main concept behind the GSN is the *virtual sensor*. A *virtual sensor* is a data stream received either directly by a real sensor or

provided by another *virtual sensor*. *Virtual sensors* can consume several streams and produce only one.

Virtual sensors provide all the necessary information for their deployment and use. This description is defined using XML and includes aspects such as metadata for identification and discovery, data structure of streams, a SQL-based specification of the stream processing and some functional properties –related to persistency, error handling, life-cycle management and physical deployment.

GSN acts as a container for several virtual sensors, and it is connected in a decentralized fashion by following a peer-to-peer architecture.

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 neighbourhood –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.

2.1.3 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 policies. 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 behaviour 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.

Context-Oriented Programming. The goal of the Context-Oriented Programming (COP) (Keays & Rakotonirainy, 2003) language is to support AmI systems focused on alleviating the problems derived from the change of context in applications of this kind –adaptability, portability and complexity. Four programming constructs –goals, contexts, open terms and stubs– are defined in COP. *Goals* describe the purpose of an activity. *Context*, which refers to any information that may have impact in the behaviour of the program, is used by *open terms* and *stubs* to limit or describe their domain of validity.

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.

2.1.4 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 bi-dimensional barcodes result attractive because of the easy way in which they can be produced and processed.

Regarding **frameworks**, the strategy followed in the present work is to combine an event-based approach with a service based one, without requiring computing capabilities for the physical elements involved. The **service oriented smart items infrastructure** considers the integration with business processes –overlooked in the rest of analyzed proposals– but it requires computational capabilities for the physical elements involved. In the present work, the possibility of incorporating common real-world objects in business processes is essential.

The retrieval of data and services from physical elements are addressed by different proposals such as **Global Sensor Network**, **Web presence** and **Open lookup** infrastructure by using reliable technologies –mostly web-based– that are in use today. The current work is not facing this aspect since it is considered well addressed and any of these proposals can be used.

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** or **Context-oriented programming** have no specific constructs for identification, being identifiers considered as sensed information. **Rei** offers a powerful policy description language. In the present work, policies are also based on the idea of describing what entities can/cannot and should/should not do. However, since the present work deals with physical objects that can cross the boundaries of many different organizations, a simplified policy sys-

tem is considered. In this way, since it requires less effort, it can be easily accepted by the involved partners.

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.

2.2 Business Processes Management

A business process is the flow or progression of activities –each of which represents the work of a person or a system– towards some business goal. Business Process Management (BPM) is an initiative to promote the automation of business processes in organizations. BPM is a set of technologies and standards for the design, execution, administration, and monitoring of business processes.

For the present work, it is specially relevant the support for modeling and execution of business process. The following subsections provide an overview of these aspects.

2.2.1 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 activities, 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 behaviour 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 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 2.1 shows the complete set of events defined by the notation.

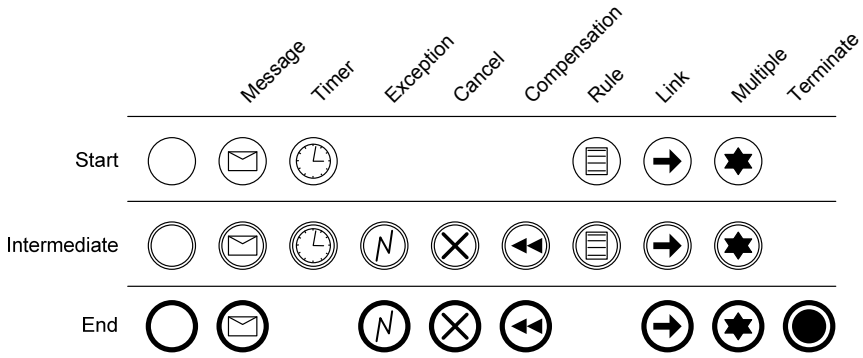

Fig. 2.1. Event types defined by BPMN

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 2.2 depicts the different types of activities and the markers available to specify when the activity can be repeated and how.

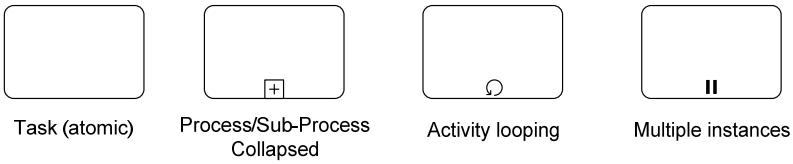

Fig. 2.2. Activity types defined by BPMN

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 behaviour control. Figure 2.3 depicts the different types of gateways provided by the notation.

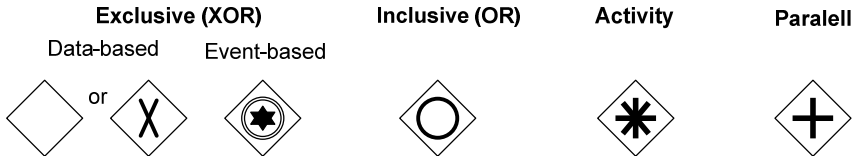

Fig. 2.3. Gateway types defined by BPMN

Connecting Objects

This element allows 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 will represent the two 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 2.4 depicts the different connecting objects defined by the BPMN notation.

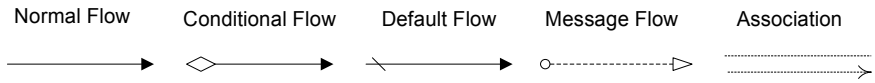

Fig. 2.4. Connecting object types defined by BPMN

Swimlanes

This element allows grouping Flow objects based on a particular criterion. This category includes two types of elements –see Fig 2.5– which are:

Pools: represent a Participant in a Process.

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

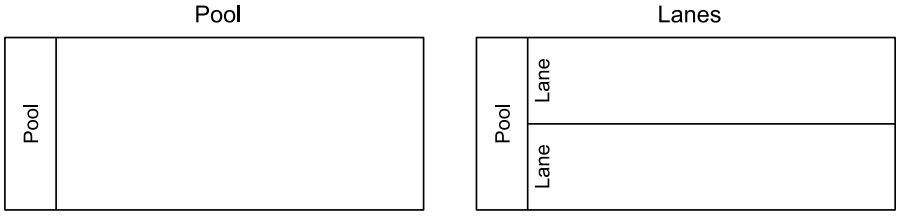

Fig. 2.5. Gateway types defined by BPMN

Artifacts

The elements included within this type are introduced into business process models to improve their understanding –see Fig. 2.6. 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.

Annotation: Text Annotations are the mechanism provided to modellers to introduce additional information for the reader of a BPMN Diagram.

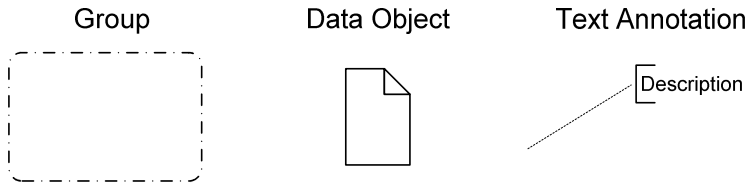

Fig. 2.6. Artifact types defined by BPMN

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

2.2.2 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 –e.g., 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. So business processes modelled with BPMN can be translated automatically to an executable WS-BPEL definition.

2.2.3 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 to implement them. However, business process management is 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, favouring the implantation of the Internet of Things.

2.3 Conclusions

This chapter illustrates the state of the art in the two disciplines this work is related to. Both areas are really active these days with many emerging initiatives. However, there is still a lack of proposals to bridge both areas. On the one hand, the Internet of Things area is still focused on implementation issues, with little attention to modeling aspects. On the other hand, business process modeling initiatives do not provide support for considering the specific requirements of physical elements.

The present work proposes an integral approach facing the construction of the Internet of Things from a business process modelling perspective. In this way, both disciplines can be benefited from each other. The Internet of Things area can improve its development methods by the use of modeling and the Business Process area can design processes that are closer to the real world, where activities take place.

3

An Architecture for the Internet of Things

In the development of business process-supporting systems for the Internet of Things, many technologies of different kinds are involved. This technological heterogeneity is related to both, the Auto-ID aspects and the BPM aspects. In order to avoid ad-hoc developments, this work proposes to raise the abstraction level for the development of such systems. In order to do so, as a first step, the present chapter defines a software architecture to deal with the particular requirements of this kind of applications avoiding technology heterogeneity.

The presented software architecture is defined following an architectural process that decouples architectural concepts from technological solutions and stresses the relevance of automating the development process. This offers a twofold benefit; on the one hand the obtained architecture is less sensible to technological evolution, making it long-lived. On the other hand, architectural concepts are clearly defined, which is basic for automating the development process as it is illustrated in Chapter 4.

The rest of this chapter is organized as follows: Section 3.1 presents the process followed to define the architecture. Section 3.2 defines architectural concepts in a technology-independent fashion. Section 3.3 presents a programming model to develop for the architecture. Section 3.4 defines the mapping between the technology-independent concepts defined in the architecture and a particular technological solution. In Section 3.5, a mock platform to ease the development and facilitate testing is defined. Section 3.6 presents the development of a vertical prototype to validate the fulfillment of the requirements for

the architecture. Section 3.7 presents different case studies developed that had lead to the consolidation of the architecture. Finally, Section 3.8 concludes the chapter.

3.1 The Architectural Process

In this chapter an architecture to integrate Auto-ID mechanisms in business processes is defined. In order to offer support to business processes that involve real-world elements, many technological aspects should be considered related to both, business process management and Auto-ID. Some examples include Business process execution engines –based on different specifications like WS-BPEL or XPDL–, interoperability solutions –such as Web Services or CORBA– to integrate different systems, and middleware to integrate Auto-ID devices –such as RFID antennas, barcode readers and the like. The involved technologies are many in both fields, and new ones are expected to appear.

In order to avoid that the presented architecture could be affected by the high technology diversity in service orchestration and Auto-ID solutions, 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. 3.1– composed of the following steps:

1. **Elaboration phase.** The architecture is defined 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**. In this way, software development for the architecture becomes more effective.

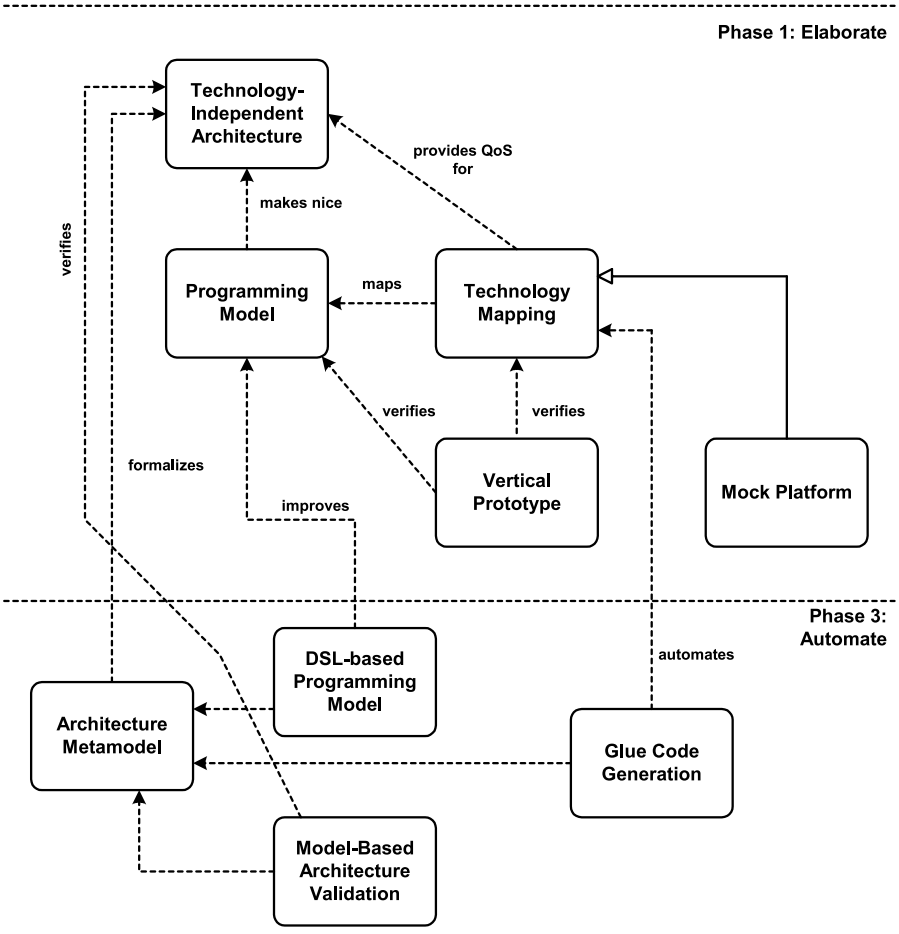

Fig. 3.1. Phases of the architectural process

This chapter is focused on the definition of the architecture. So it comprises the two first steps –elaboration and iteration– of the architectural process. The third step of the process –focused on automating the development for the defined architecture– is faced in Chapter 4.

3.2 Technology-independent Architecture

The present work deals with Information Systems that integrate real-world elements to support business processes in an organization. This definition is quite broad but this work is focused on applications where this linkage is highly exploited and identification –specially Auto-ID– is relevant. These systems contain several elements suitable to be identified by means of potentially different technologies.

An example of such a system could be a library in which clients can borrow books just by picking them up. By providing a digital identity to books and member cards, the system can be aware in every moment who is borrowing a book. If a non-member –or a member with an expired card– takes some book; the member first, and then, the security personnel can be warned. By using Auto-ID technologies, there is no need for librarians to transfer the data about book loans to the Information System. So, the process becomes more efficient and queues can be reduced.

The defined architecture is focused on the role identification plays in business processes. A software architecture supporting a system like the described in the library example should fulfill many requirements derived from the automation of the physical-virtual linkage. In the following subsections these requirements are stated and architectural concepts are proposed to define an architecture at conceptual level that fulfills them.

3.2.1 Requirements for the Architecture

Real-world objects can affect business processes in different ways. Figure 3.2 illustrates an example of how a system can react when a physical object is involved in a business process. In the example, the illustrated task consists in the reception of a package containing some material previously requested. The detection of an element –a package in this case– has a threefold impact on the

system: (1) The system can retrieve some data about the detected package such as the consignee or the requested material. (2) The process can continue immediately with the next task –notify the consignee that his package just arrived– since the package detection event determines the completion of the current task. (3) The different services presented at the system can adapt their behaviour to the requirements of the detected elements –e.g., since the package is fragile, the "caution" light incorporated in the reception table is turned on, to indicate that the package should be handled with care.

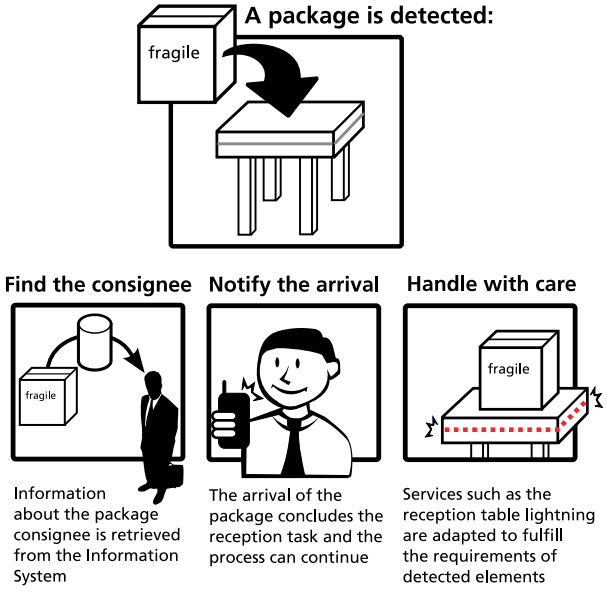

Fig. 3.2. Impact of real-world elements in the system

As illustrated in the figure, in this work the integration of real-world elements in business processes is faced from three different perspectives: (1) the transient of information between physical and virtual worlds, (2) the generation of business level events that guide the process and (3) the requirements for adaptation introduced by real-world elements. A more detailed description of these three perspectives is provided below:

Information flow. When real-world elements participate in business processes, information is constantly moving between physical and digital

spaces. Physical objects are detected and information about them is stored and retrieved from the system. The system should provide support for the identification of real-world elements but also for the materialization of identifiers in the physical world.

Process guidance. A business process consists in a set of activities. Several instances of the process –e.g., different material requests– are executed concurrently and the detection of a physical element can play an important role in the process course. On the one hand, detecting an element can be used to find the associated process instance. On the other hand, detection events can determine the termination of a task. Normally, humans are in charge of selecting the process instance they are working with, and explicitly indicate when they have finished a certain task. Automating task completion and correlation makes the process more fluent.

Adaptation triggering. An environment with lots of services embedded in it, should provide its functionality in the best possible way. Physical elements have their requirements regarding system adaptation. This applies to any identifiable element including objects, people and places. In the previous example a warning light is turned on when a fragile package is detected. In addition, the system can require adaptation to particular needs of people –e.g., visual-impaired users require acoustic feedback– and places –e.g., a meeting room requires a silent behaviour for services.

These three dimensions are considered along this work; from the definition of the software architecture to the definition of the development process.

3.2.2 Conceptual Definition

In this section we present a technology-independent description of the architecture introduced in this work. By using a description based on technology-neutral concepts we obtain **a sustainable software architecture**. This is, an architecture that is not affected by technological hypes and can evolve in time along several technological cycles.

For the definition of the architecture we rely on the *component* concept. Components become the basic software pieces that will conform the system. Component functionality is described by means of *interfaces*. Components are connected by *wires*, and they offer asynchronous communication. We have

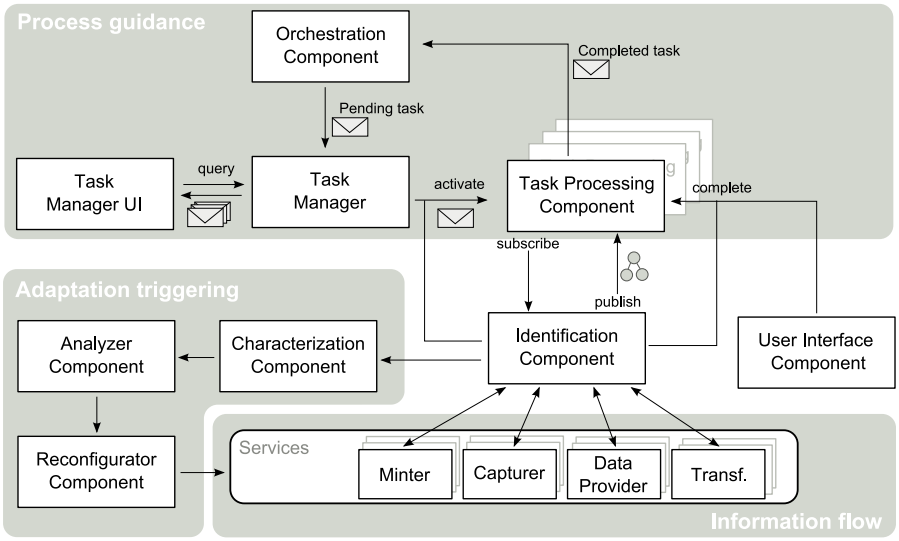

Fig. 3.3. Architecture component overview

opted for asynchronous communication since business processes considered are usually long-running and tend to involve human participation. In addition, a subscription mechanism following the Observer Pattern (Gamma et al., 1995) is used to deal with identification events for Auto-ID related components.

It is worth noting that several instances of a process are usually running at the same time, so correlation mechanisms are needed. For example, in a library, requesting new books and receiving them happens asynchronously. When new books arrive, it is important to determine to which of the many active requests they correspond to.

Since this work is considering the integration of Auto-ID mechanisms in business processes, the presented architecture has to balance the message-based approach usually associated with business processes with the event-based approach of Auto-ID mechanisms. The presented architecture considers events as low-level messages that are processed –e.g., filtered, aggregated, etc.– to generate business-level messages. Fig. 3.3 illustrates the components involved in this process. In this diagram the different components are represented by boxes and arrows are used to indicate the communication among them. Message-based communication is indicated by means of an envelope.

The components defined respond to needs detected for each of the three different aspects identified previously – information flow, process guidance and adaptation triggering. The *Identification Component* is the central element in this architecture and it is involved in the three identification aspects detected, acting as a connection point.

This component provides the mechanisms to bridge physical and digital spaces at different levels. It is in charge of (1) providing the required information to tasks by consuming the Auto-ID services, (2) notifying the activation and completion of tasks, and (3) triggering system adaptation when detected elements have adaptation requirements.

The rest of components that form the architecture are in charge of supporting one of the identification aspects. More detail about the different components involved in each aspect is given below.

Information Flow Components

For the present work it is assumed that the underlying system functionality is organized in *Services*. *Service types* are defined by clear interfaces and it is possible that many services exist of the same kind. In order to offer its functionality, services can make use of *Resources*. For example the lightning service of a particular room is a service of the more generic lightning service type, and different resources –such as light bulbs, neon tubes, gradual lights and the like– can be used by this service.

In the present work four service kinds regarding identification are considered. These are *Capturers*, *Minters*, *Transformations* and *Data Providers*. For each service of these types a component exists in the system in order to provide the corresponding functionality and interact with the adequate resources.

Capturers –such as a barcode reader– can acquire identifiers from the Physical Space to the Digital Space. *Minters* are in charge of generating physical representations of an identifier –such as a printing device that produces a barcode label. *Transformations* define conversion operations between different codifications. *Data Providers* provide the functionality required to obtain the information associated to an identifier, modify this information and create new instances –e.g., obtain the book details from its ISBN number.

Process Guidance Components

In order to support business processes, the architecture should provide support to keep track of the different activities in the process. The components considered in the present architecture for the control of the process flow are detailed below.

Orchestration Component. This component is in charge of managing the state of the long-running business process. It interchanges messages with different systems in order to orchestrate the process. It is in charge of creating new process instances, keep track of the different process instances and perform message correlation –i.e., find the process instance related with the received message.

Task Manager. This component gives support to the asynchronous communication of systems that participate in the business process. It receives messages from the *Orchestration Component* and waits for components to process them. The *Task Manager* can be queried to obtain the pending tasks that conform certain criteria –e.g., targeted to a certain user. Pending tasks can be added and cancelled by the *Orchestration Component* and completed by a *Task Processing Component*.

Task Processing Component. This component receives a message corresponding to a pending task when it is *activated*. Then it retrieves information and composes a response message. When the task is *completed*, the response message is sent to the *Orchestration Component*. To complete the response, the required information can be provided by the user or by some *Identification Component*. The Task Processing Component has to subscribe to the different Identification Components –subscription is active until the task is completed.

These components can have a *User Interface Component* associated to them to permit user participation or just display some helpful information. Although process automation is one of the main goals of this kind of systems, sometimes it is not possible to obtain a complete automation using sensors and user interfaces are required.

Adaptation Triggering Components

When a physical element is identified by a software system it can trigger the adaptation of the system to its needs. The system should react in response, adapting the way in which services are offered. For example, if a quiet location –such as a library– is detected, the system should offer its services in a silent way.

A **policy mechanism** is defined to avoid forcing physical elements and systems to know details about each other, making more flexible their relationship. A policy is defined in the present work as a set of assertions. Each assertion is formed by a pair (P, Q) where P is a property and Q is a qualifier for this property. Properties are general behavioural intents that indicate how system behaviour is expected –e.g., *silent*, *non-distracting*, *efficient* and the like. The qualifier Q indicates the enforcement degree for a property. Four qualifiers are considered: *required*, *preferred*, *discouraged* and *forbidden*.

The qualifiers considered vary in their positiveness and their enforcement degree. When a property is *required*, the system is forced to fulfill this property in its behavior –if this is not possible, it is considered an error. The *preferred* qualifier is not so strict, systems are just recommended to operate in a certain way. Analogously, the *forbidden* qualifier forces the system to avoid a certain property while the *discouraged* qualifier is considered a recommendation for avoiding some property in system behaviour.

An example of the role of policies in system adaptation is illustrated in Fig. 3.4. Physical elements –in this case a physical location such as a library– can be accompanied by a policy. This policy expresses requirements for the system such as requiring a silent behaviour. This requirement is expressed by means of an assertion that states that the *silent* system property is qualified as *required*. If the *silent* property is part of a standard vocabulary, system designers could define how to handle elements that require a silent operation without the need to know details about the specific objects.

In order to support policies at the system side, the different resources of the system should be qualified to indicate which ones provide a better support for a certain property. When there are several possible resources a service can use, the selection will be made considering the fulfillment level each resource provides. The resources that better support the properties demanded by the different policies are chosen. Three fulfillment levels are considered –*complete*,

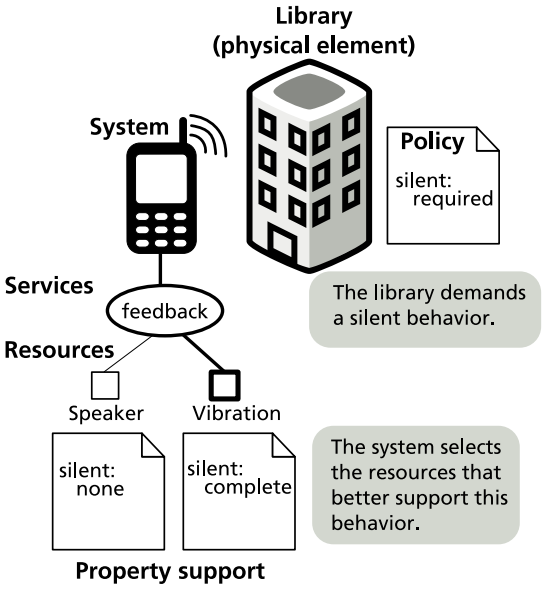

Fig. 3.4. An example of the use of policies for system adaptation

partial or *none*. The *complete* level is for resources that allow for an operation that intrinsically fulfills the property. The *partial* level is for resources that can be accepted as fulfilling a given property. Finally, the *none* level is for resources that do not support a given property.

In the example, a feedback service would be provided using vibration, avoiding the use of speakers since they do not fulfill the *silent* property. In order to support this adaptation process different components are considered in the architecture. These are *Characterization*, *Analyzer* and *Reconfiguration* components.

Characterization component. This component identifies the active policies as a list of properties. This supposes to aggregate the different assertions contained in all the active policies. Assertions of different policies can be contradictory and some trade-off should be considered. These trade-offs consist in prioritizing the most restrictive assertions.

Analyzer component. Considering the active assertions, this component ranks the different resources available in the system. This ranking indicate for a given service which resources fulfill better each assertion.

Reconfigurator component. This component performs the bindings between services and resources. When the system needs to access a resource, this component provides the most adequate resource –considering the ranking provided by the *analyzer component*– to the demanding service.

3.3 Programming Model

Once defined a technology-independent architecture, this section defines how this architecture is used from the developer perspective. Many of the components defined by the architecture are generic. They constitute an infrastructure that could be used for any particular domain. In order to customize a system for a particular application, a business process description should be provided to the *Orchestration Component*. For each task defined in the process, a *Task Processing Component* should be implemented. This component will rely on different *Identification Components* to be aware of the physical world. These components should also be implemented. The defined services required for these Identification components should be developed in case they are not already present in the system. Finally, resources used by these services can be qualified to allow system adaptation. More detail about the definition of these components is given below.

3.3.1 Business Process Definition

The Orchestration Component is in charge of orchestrating the different activities that conform the business process. These activities and their temporal relationship is defined declaratively. Given this definition, the component can invoke external systems, wait for messages or generate new messages for the corresponding components of the architecture.

3.3.2 Task Processing Component

A *Task Processing Component* is defined for each kind of task present at the business process –e.g., the reception of requested material. Given an input message for a pending task, the logic that determines how the resulting response message is constructed should be defined.

For each concrete activity –e.g., the reception of a given book–, there is an input message sent by the *Orchestration Component* [1] to provide some context information –e.g., request number, expected book, etc. By processing this context information, and accessing the adequate Identification Components –e.g., to retrieve the book information.– the task completion message can be produced and returned to the Orchestration Component.

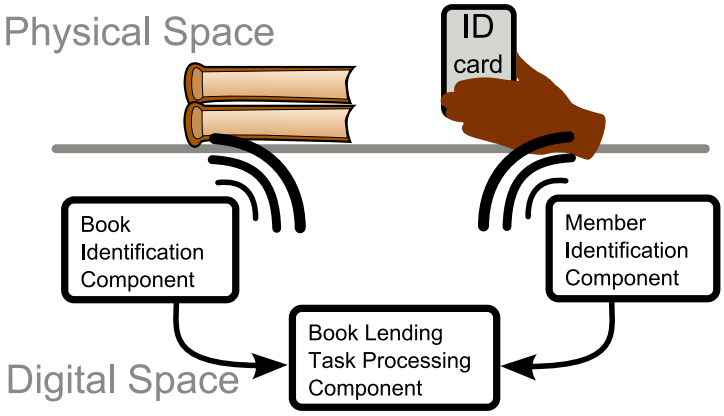

Fig. 3.5. Two Identification Components supporting a task

When defining the Task Processing Component, information dependencies are detected, this determines the required *Identification Components*. For example, in a library, the *lend book* task can require the identification of the client and all the books the client is borrowing. So two Identification Components –implementing the *Identification Component interface*– are required, one for detecting *clients* and another for *books* as illustrated in Fig. 3.5.

3.3.3 Identification Components

The development of Identification components consist in (1) selecting the adequate services required to bridge physical and virtual worlds, (2) controlling task life-cycle and (3) managing error situations.

[1] Messages are obtained from the Task Manager component. However, they are produced by the Orchestration Component, being the Task Manager role a message buffer to store messages until activated.

Selecting Services

The service kinds considered for the present work are *Capturers*, *Minters*, *Data Providers* and *Transformations*. All these elements implement different interfaces, and normally wrap the functionality offered by the specific technological solution.

Once the required elements are defined, they are connected together. In addition, a specific *User Interface Component* offering a customized view of the *Task Manager* for each particular task can be defined to provide specific information required for a certain task in order to improve user experience.

In order to choose the Auto-ID technology used for each component we have to make several considerations. For example, *Capturers* used should be compatible with the *Minters* used in previous tasks –using a barcode reader for some products is useless if these products have not been previously labelled with barcodes. For *Minters* and *Capturers* it is important to know whether they are capable to detect several elements at the same time or not. In addition, some non-functional considerations such as implantation and maintenance costs should be considered for technological election.

Defining Task Life-cycle

When defining *Task Processing Components* the origin of task life-cycle events –initiation and termination of the tasks– should be identified. *Identification Components* can provide this information.

For example, in a library when a package of books arrive –see Fig. 3.6–, the completion of the reception task can be notified explicitly by the user –using a User Interface Component– or implicitly using Auto-ID mechanism– by means of an Identification Component. When Auto-ID events guide task life-cycle, the process becomes more fluent since the system is notified as the task is performed.

Error Management

Since Information Systems have not full control of the physical world, there is a need to define error conditions when the detected elements are not the ones expected. Error management is essential when implementing the access to data providers.

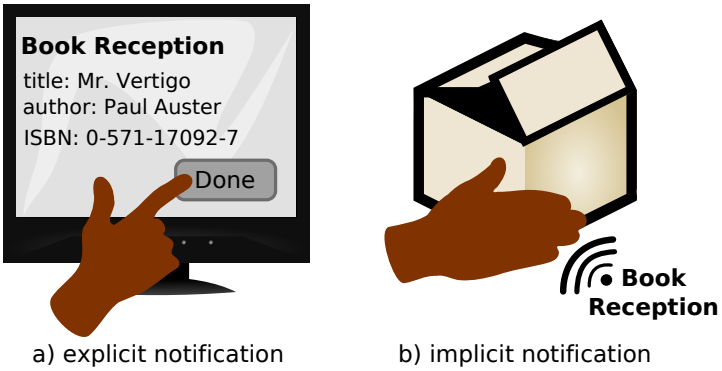

Fig. 3.6. Explicit and implicit task life-cycle

When defining a Identification Component, is important to define how the system is responding in certain situations such as the absence of information associated with an identified element, or the presence of two information pieces associated with the same identifier.

3.3.4 Service Definition

Since this work is particularly interested in Auto-ID, the defined service types correspond to basic functionality usually found in this kind of systems (Kindberg et al., 2002). These service include *Capturers*, *Minters*, *Data Providers* and *Transformations*. The present architecture defines an interface for each service type. Since each service implementation conforms to its service type interface, replaceability of services is guaranteed. Although services can be replaced, in the case of identification some semantic mismatch can be produced due to the use of different codifications –this is addressed from a modeling perspective in Chapter 4.

A service can use different resources to provide its functionality. For example, warning the user of a certain event –such as a phone call–, can be done in different ways –e.g., a message in a screen, a beep using the speakers or with vibration– involving different resources. Although the service functionality remains the same, the use of one or another resource has an impact on some non-functional aspects. Some resources can be more power-demanding, more silent or more expensive to use. The policy system used in the architecture allows each service to select the most suited resource.

In order to do so, resources should be qualified, indicating the degree of fulfillment –*complete*, *partial* or *none*– for each of the system properties considered relevant. The labeling of resources follows a similar approach as the followed in folksonomies (Mathes, 2004). Resources are tagged with the properties they support to enable the lookup of resources. Techniques to process tags applied to folksonomies can be also applied to the policy properties –e.g., consider synonym properties if a pair of properties are always present together when resources are qualified.

3.4 Technology Mapping

Once defined the technology-independent concepts that conform the architecture and the programming model that defines how to use this architecture, the technology mapping is elaborated. 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 given below.

3.4.1 Target Technology

Service Component Architecture (SCA) has been chosen as the target technology. SCA is a vendor, technology and language neutral model for implementing Service Oriented Architectures (SOAs). We have adopted this technology for the following reasons:

Component model. The SCA component model fits with the technology-independent architecture concepts defined. SCA components declare explicitly their references in terms of required interfaces and are resolved following the Dependency Injection pattern. Components are injected by a container to solve these dependencies according to a configuration file.

Support for asynchronous communication. Asynchronous communication is supported by SCA by means of the bidirectional interface concept. A bidirectional interface is composed by a pair of interfaces, the provided and the callback interface. A component implementing the bidirectional interface should implement the methods defined in the provided interface, while clients of this component should implement the callback interface.

Data abstraction. For data handling, SCA can use Service Data Objects (SDO). SDO makes it possible to hide the back-end data source, as it offers homogeneous access for XML, relational or file based data sources among others. In addition, SDO permits disconnected data access patterns with an optimistic concurrency control model. These properties are also adequate for the considered application domain. Since data portions from many sources are combined and updated.

Support for distribution. The systems this work is dealing with usually expand across organizations and they involve diverse computing resources. Labels are commonly produced by one system and read by many different systems with the potential need to share information among them. SCA components can be distributed easily in different computing nodes, from different threads in the same machine to different computers. This flexibility is desirable in the current application domain.

Technological integration. SCA allows the use of different technologies for the implementation of components and their communication. This favours the integration of different systems. For the purpose of this work, the possibilities of integrating Business Process Execution engines and Auto-ID middleware solutions is specially interesting.

From the different available implementations of SCA and SDO specifications[2], Apache Tuscany[3] has been chosen since it is an open source solution in a mature state.

3.4.2 Mapping to a Technological Solution

Once SCA has been chosen as the target technology, it is defined how the technology-independent concepts described in Section 3.2 are mapped to implementation assets.

Orchestration Component: Since Business Process Management community is familiar with the WS-BPEL standard, WS-BPEL has been chosen as the language for the *Orchestration Component*. Therefore, a WS-BPEL definition of the business process activities and its interface –by means of a

[2] http://www.osoa.org/display/Main/Implementation+Examples+and+Tools
[3] http://tuscany.apache.org/

WSDL definition– are required. Apache Tuscany supports the implementation of components using WS-BPEL by means of Apache ODE Orchestration Engine. However, since the *Task Manager Component* is exposed as a Web Service, other process execution engines and Web Service-based software –such as Enterprise Service Buses or Messaging Middleware– can be used to drive the business process execution.

Task Manager. This component is implemented in Java. It is a repository for pending tasks. Query capabilities are provided thanks to the XPath support that SDO offers. Pending tasks are formed by a header that includes some metadata –reception date, user group in charge of completing the task, ect.– and the payload – domain-specific data that the *Task Processing Component* will handle. When designing data structures, meta-information included in tasks should be defined. The more meta-data considered, the more fine-grained queries could be. On the other hand, including great amounts of metadata supposes to move complexity to the *Orchestration Component* since this is the component in charge of providing this information.

Task Processing Component. Components of this kind define a set of references according to the information elements they require –or they need to generate– to complete a specific task. SDO API is used to compose the resulting message that is returned to the Orchestration Engine. In the component specification, wires are defined to determine which *Identification Components* are providing the defined references.

Identification Component. This component provides a bidirectional interface to allow subscriptions. Task Processing components that are subscribed to a particular *Identification Component* should implement the corresponding callback interface. Identification Components can be connected to the *Task Manager* to trigger the activation of certain tasks. They can be also connected to Task Processing Components to trigger their completion.

For the definition of *Capturers*, *Minters*, *Data Providers* and *Transformations* an implementation of the corresponding interfaces should be provided. These components can be implemented using several technologies. Fig. 3.7 shows Java interfaces defined for *Capturers*. Capturers should implement the *Capturer* interface. This interface defines a *read* method

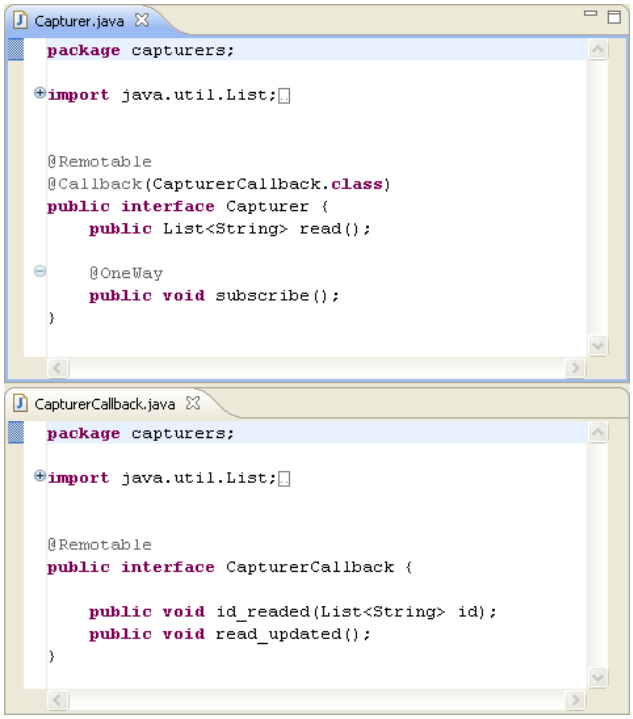

Fig. 3.7. Bi-directional interface for the capturers

for retrieving the current detected identifiers and a *subscribe* method to receive identification notifications. The *subscribe* method is executed asynchronously –see the *OneWay* SCA annotation– so execution can continue without waiting for a response. The *Capturer* interface is associated with a callback interface –see the *Callback* annotation in the Capturer interface– that should be implemented by the classes that make use of Capturers. The callback interface implements two methods that are used in response of the previous ones. The *id_readed* method returns the captured identifiers. The *read_updated* indicates the occurrence of an event in the Capturer.

In addition to Java, for our implementations OSGi is quite interesting to obtain some dynamic capabilities for services.

Reconfiguration components. *Characterization*, *Analyzer* and *Reconfiguration* components are implemented in Java. Policies are defined using

XML, so they can be easily handled thanks to SDO. The XML *namespace* notion is used to avoid name collision when dealing with multiple properties. Service qualification is also defined in an external XML file, so it could be easily updated without affecting the system. When new polices are activated, their constraints are considered and the resource description file is accessed to indicate which resource fulfills better the required properties.

User Interface Components. Apache Tuscany allows the implementation of components using different technologies. For simple user interfaces HTML and Javascript offering a json-rpc[4] based communication, have been used. For more advanced user interfaces, Java libraries for the design of graphical user interfaces and visualization techniques have been also used.

3.5 Mock Platform

Once the technology for the architecture has been decided, it is important to define a mock platform for developers. With a mock platform, developers can run tests locally as early as possible. The choice of SCA and SDO as target technologies offers good properties to be used as a mock platform.

SCA is based on the Dependency Injection Pattern. Components declare their dependencies and the application wiring is defined in an external asset. Thus, components are easily interchangeable as far as the replacing component provides the same interface than the replaced one. This allows an easy definition of mock components for test purposes.

At the data level, SDO makes it possible to hide the back-end data source, as it offers homogeneous access for XML, relational or file based data sources among others. This allows a seamless migration from a mock platform –e.g., based on static XML files– for testing to a production environment –e.g, based on Web Services.

[4] http://json-rpc.org/

3.6 Vertical Prototype

In order to validate the defined architecture we have implemented an application based on the Smart Toolbox (Lampe et al., 2004) case study, a representative application in this domain. This case study consists in monitoring the tools used for aircraft maintenance tasks. During aircraft reparations, the system should prevent tools from being lost and causing potential damage. In order to do so, the content, the location and the carrier of the toolbox is sensed automatically in real-time. This application improves the aircraft Maintenance, Repair, and Overhaul (MRO) process in an unobtrusive manner, so it fits perfectly in the application domain targeted in this work.

In this stage of the architecture development the focus is on testing the non-functional requirements, so only a small subset of the functional requirements were considered for the development. Development was focused on one of the process tasks that consist on the reparation of planes. During reparation, the toolbox content is monitored. When the reparation is completed, the system checks that the toolbox contains all the previously assigned tools –avoiding tools to be accidentally forgotten and causing any damage. Repairing task completion is automatically informed by a location change. When the toolbox detects that the mechanic is leaving a certain location, the task is automatically considered as completed.

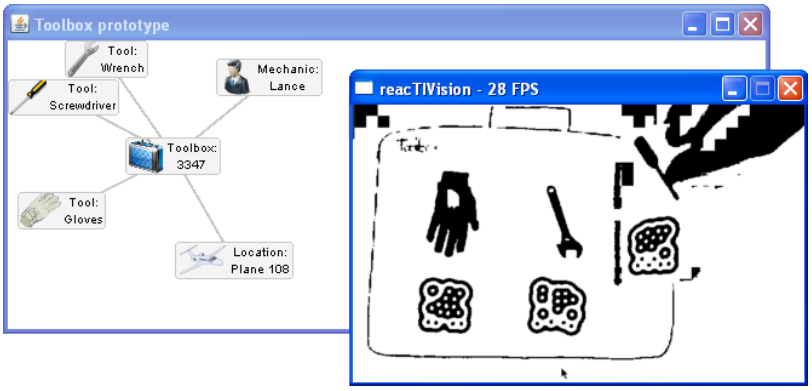

Fig. 3.8. Smart Toolbox prototype

Figure 3.8 shows an screenshot of the developed prototype. This prototype integrates different identification technologies. Mechanics and locations are identified by textual labels while tools contained in the toolbox are identified by *fiducials* (Bencina & Kaltenbrunner, 2005) –two-dimensional barcodes specially designed for real-time video camera recognition.

For the minting of *text labels* and *fiducials*, a common printer was used. For capturing physical elements, different technologies are used. *Textual labels* are transferred to the system using a web interface. For *fiducials*, the *Fiducial Capturer Component* wraps the reacTIVision framework[5]. A wide-angle lens webcam was used as a capturing device. The window at the right hand side of Fig. 3.8 shows the images acquired by the reacTIVision framework. Paper labels with a printed fiducial –accompanied by an illustration of the tool kind– are used to detect the content of each toolbox –a wrench, a screwdriver and gloves for the example. At the left hand side of the figure, there is a representation of the information present at the digital space.

Since the support for WS-BPEL in Tuscany is quite recent, some problems were found with the Orchestration Component, so an external Process Execution Engine –Intalio BPMS[6]– was used instead, using the Web Service binding for the communication with the Task Manager.

This solution increased the reliability of the system but performance was sacrificed by the use of Web Services. Since the architecture is intended to support long-running processes, this performance reduction was not considered meaningful.

3.7 Consolidation of the Architecture

In order to stabilize the architecture concepts, define better the programming model and adjust the technology mapping; we have applied the architecture to two existing case studies. These case studies came from both, Business Process Management area –not dealing with real-world elements– and the AmI area –not considering business process aspects.

Introducing the defined architecture for the development of these case studies offers two complementary visions of the benefits this architecture pro-

[5] http://reactable.iua.upf.edu/?software
[6] http://bpms.intalio.com/

vides. On the one hand, BPMs can be extended to consider real-world elements. On the other hand, AmI systems can become process-aware; adapting to the user specific needs for each task. The comparison with the original systems was useful as feedback to improve the architecture.

The considered case studies are (1) the complete version of the *Smart Toolbox* and (2) the *Smart Library* case study. The Smart Toolbox case study extends the developed prototype to cover the whole MRO process as it was described in the original case study. The Smart Library case study describes a library where members can borrow books in a self-service fashion.

Both case studies have been implemented following the present architecture and have been also used as testbeds for the development method presented in this work. More detail about the development of these case studies –including specification and implementation aspects– is provided in Chapter 6.

3.8 Conclusions

The present chapter introduces an architecture for the support of business processes where real-world elements are involved. The architectural process followed in its design decouples the architectural concepts from the particular technological choices. In this way, the architecture becomes long-lived and can survive the technological evolution of the system.

A programming model is defined for this architecture with support for the testing of the resulting systems. The architecture has been used in several case studies to freeze the architectonic concepts and check that the technological decisions can fulfill the requirements for the applications faced in this work.

Therefore, this architecture responds to the needs derived from the integration of real-world elements in business process. The following chapters build on top of this architecture new concepts and methods to ease the development of this kind of systems.

4

Automating the Development

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. Thus, changes in requirements can be mapped automatically to the particular technology the system relies on, facilitating its evolution.

In order to automate the development process, system descriptions should be machine-processable. Modeling techniques can play a key role in this process. Models are communication artifacts expressed by some kind of language. Modeling languages can take the form of arrows and boxes, text or even speech. But the important thing is to capture the rules that represent this language in a precise way to allow an unambiguous use. In order to determine which constructs can be used to define a model, a metamodel –this is, the model of a modeling language (Favre, 2004)– is used.

Provided that metamodel concepts are defined in a precise way, models can be transformed automatically into new models by means of model transformation techniques. This enables automation in system development since artifacts can be derived in a systematic way. Many technologies and standards give support to this development paradigm. For example, 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.

To allow the description of a system based on the architecture concepts introduced in this work, these concepts must be precisely defined. The present chapter uses modeling techniques to formalize concepts and raise the level of abstraction in the development of the kind of systems this work is dealing with.

The rest of this chapter is organized as follows: Section 4.1 details how the architectural concepts have been formalized. Section 4.2 provides mechanisms to avoid repetitive tasks in the development. Section 4.3 defines modeling primitives to capture the requirements of this kind of applications. Finally, Section 4.4 concludes the chapter.

4.1 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. Using a metamodel, system descriptions become unambiguous at least at syntactic level. This makes the descriptions machine-processable.

The **architecture metamodel** has been defined as the first step towards the automation of the development process. This metamodel captures the concepts defined in Section 3.2 such as the *Identification Component* or the *Task Processing Component*, and the constraints for their composition.

Fig. 4.1 shows a diagram for the architecture metamodel. The *AmiBizSystem* metaclass is the root element that represents the whole system. Elements from the system infrastructure appear as optional metaclasses for which only one instance is allowed in the system. It is the case of the *Orchestration Component*, *Task Manager* or the *Adaptation Component* that includes adaptation related components – such as *Characterization*, *Analyzer* and *Reconfigurator* components.

Domain-dependent elements –such as *Identification* or *Task Processing* components– can appear more than once in a system. So, they have a name –they extend the *Named Element* metaclass– to distinguish each instance. The system functionality is organized in *Services*. The notion of *Service* is

extended in the metamodel to define identification-related components such as the *UI Component* for user interfaces and the *Identification Component*. The *Identification Component* can make use of other services to achieve identification tasks.

It is worth noting that since the Identification Component is also a service, Identification Components can be used by other Identification Components. This is useful for handling containment relationships. For example, in the Smart Toolbox example, a component detects the tools contained in the toolbox and another component makes use of it to provide the detection of the toolbox.

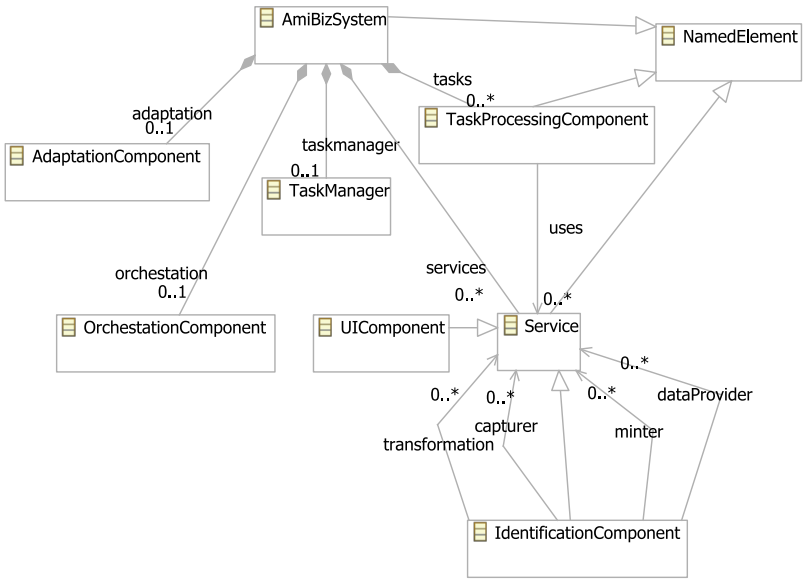

Fig. 4.1. Architecture metamodel

By using the constructs defined in the metamodel, different systems can be modeled. This enables the resulting models to 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[1] (EMF), is a language targeted at the definition of metamodels

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

with precise semantics. EMF provides tool support for the definition of metamodels and the edition of models.

4.2 Glue Code Generation

The technology mapping generally involves several repetitive tasks. For our target technology, the definition of each *Identification Component* involves actions like the definition of a Java class that extends the *IDComponent* interface, the use of SCA annotation to indicate the SCA container that an identification service is provided and the definition of the component in the XML configuration file. 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.

From the description of a system following our defined metamodel, source code can be generated with model-to-text transformations.

Glue code generation have been implemented using Xpand templates from openArchitectureWare[2]. 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 from the model.

The current implementation generates the SCA configuration file for the system, and the Java classes that are required for the implementation of the different components. SCA annotations 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 SCA specific artifacts are generated, the use of SCA is made transparent to the developer, who only has to deal with Java programming.

[2] http://www.eclipse.org/gmt/oaw/

4.3 DSL-based Specification

In order to face the construction of the kind of systems this work is dealing with, a Domain Specific Language (DSL) is defined. In this way, requirements regarding identification in business processes can be defined in a natural way.

DSLs describe concisely problems of a certain domain. Thus, domain knowledge is vital for the definition of a DSL. Prior to the design of the language, the **analysis of the domain** is necessary. In order to do so, the following steps are required (van Deursen et al., 2000; Deursen & Klint, 1997): (1) identify the problem domain of interest, (2) gather all relevant knowledge of this domain, and (3) cluster this knowledge in a handful notions and operations on them.

The present work deals with Information Systems that integrate real-world elements to support business processes in an organization. This **definition of the domain** is quite broad but this work is focused on applications where this linkage is highly exploited and identification –especially Auto-ID– is relevant. These systems contain several elements suitable to be identified by means of potentially different technologies.

Once the domain is identified, **relevant knowledge of this domain** should be gathered in order to characterize it. For the domain of interest in this work, the core concept is the *identification mechanism*. The *identification mechanism* is in charge of preserving the real-virtual linkage.

Systems for the Internet of Things integrate different services –sensors and actuators– seamlessly in the environment. *Identification mechanisms* can be considered as a specific kind of sensor and actuator combination –e.g., a printing service combined with a barcode reading one–. However, some properties specific to identification, not present when considering these sensors and actuators independently, emerge when they are part of an identification system. These properties are described below.

Limited replaceability. The functionality that an identification system provides is well-known (Kindberg, 2002). Functions to create or acquire identifiers are common to any identification technology. However, in contrast to common sensors, the replaceability of services involved in identification is quite limited. Replacing a barcode reader by an RFID reader, despite its common functionality –capture an identifier and transfer it to

the digital word– is not trivial. Some compensatory measures –such as label all products with RFID tags if it was not done preventively– should be taken for this change to work. On the other hand, some services can be re-used with no problem. A printing service can be used to produce both, a barcode or a numeric label.

Multiplicity of the real-virtual linkage. The relation between physical elements and their virtual counterpart is not always one to one. Physical elements can share the same identifier –i.e., products that are labeled indicating their product type– or contain multiple identifiers –i.e., a human-readable numeric code accompanying a barcode–. This suggest that identification between physical and digital worlds is not conceived as an identity relation in the mathematical sense, but it is more close to the ideas of the *Counterpart Theory* (Lewis, 1968).

Distributed processes. Another particular aspect about systems of this kind is that they usually expand across organizations and they involve diverse computing resources. Labels are commonly produced by one system and read by many different systems with the potential need to share information among them. To support a whole business process, several systems need to be integrated.

In order to **cluster the knowledge** about the domain providing a good separation of concerns, the modeling primitives that constitute the DSL are structured in six perspectives. These perspectives are related to *data structure*, *business process definition*, *technological aspects*, *services*, *interaction* and *policy properties*.

The following subsections provide more detail about these modeling perspectives defining the concepts by means of a metamodel and illustrating[3] the use of the primitives by means of an example based on the Smart Library case study –for more detail about this case study, see Chapter 6.

4.3.1 Data Perspective

The data perspective represents the available information at the Digital Space. Pieces of information are organized in classes and relationships between these

[3] The concrete syntax used in examples is not normative. Further research is needed to determine the best concrete syntax –either graphical or textual– in terms of intuitiveness and comprehensibility.

classes. The relevant information for all the members of a class is expressed by means of attributes. It can be modeled by means of a class diagram. Instead of using a complex metamodel such as UML2, a metamodel suited to the needs of this work has been defined to capture data structure as it is illustrated in Fig. 4.2.

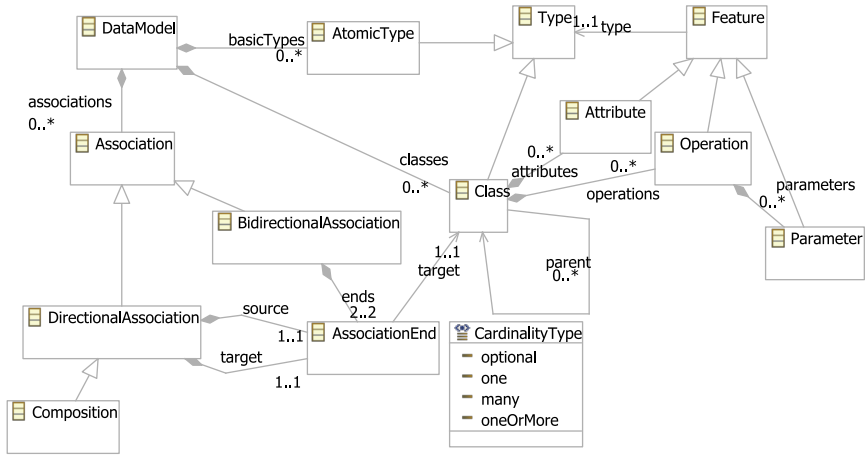

Fig. 4.2. Metamodel for the data perspective

The *DataModel* metaclass is the root element of the data perspective. To define the information handled by the system, it includes *basic types*, *classes* and *associations*. *Basic types* are defined explicitly, to promote extensibility and avoid dependencies with specific platform types, no basic types are predefined.

Classes are formed by *attributes* and *operations* that can contain different *parameters*. Regarding associations, only binary associations are considered. Associations can be *directed* or *bidirectional*, and *composition* is considered a particular case of directed association. The cardinality for association ends considers only the four values defined in the *CardinalityType* enumeration –*optional*, *one*, *many* and *oneOrMore*– not accepting intermediate values.

Example

In the library case study, the data model includes different classes such as Book, Shelf, Member and Loan. Figure 4.3 illustrates by means of a Class Diagram some of the classes defined for the example.

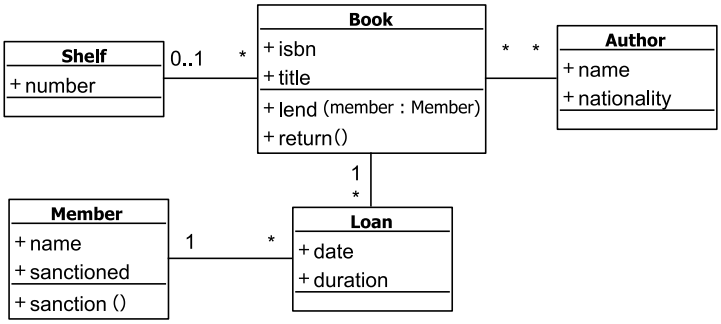

Fig. 4.3. An example of data model

Attributes are used to define the relevant information for these entities such as the book title or the member sanctioned status. The different associations allow to determine which shelf is a book assigned to or which member has borrowed a certain book. Operations to allow the lending and return of books are also defined.

4.3.2 Business Process Perspective

The present work relies in BPMN for business process description. Since BPMN has not support for handling real-world elements explicitly, an extension is proposed to cope with this kind of elements. The BPMN metamodel used is the one defined by the Eclipse SOA Tools Platform (STP) Project[4].

The *Physical Object* primitive is introduced as an extension of the BPMN Data Object to define real-world elements that are involved in a process task. Physical Objects can be required or generated by a task, depending on this, the association between the object and the task is defined in one or another direction. When a task needs the information associated to a *Physical Object*,

[4] http://www.eclipse.org/stp/

the direction of the association is from the object to the task. On the contrary, when the Physical Object is generated by a task, the association is defined in the opposite direction.

Physical Object primitive has the following attributes in addition to the ones such as *name* or *documentation* that are inherited from the Data Object element as indicated by the BPMN specification:

Class. This attribute determines the element from the data model this Physical Object is representing. In this way, the structure of the information associated to a Physical Object is defined.

Mediums. This attribute contains a list of the mediums used to represent the identifier of a given Physical Object. When multiple mediums are present, the identification is replicated using each one.

Multiple. This boolean attribute defines whether many objects of the same kind or just one can be involved in the business process task. Allowing the detection of multiple objects at the same time will have an impact in the way users interact with the system.

Used for correlation. This boolean attribute indicates whether a Physical Object can be used to find the current process instance. Physical Objects that can participate only once in a process instance –such as a book in a loan– are good candidates for correlation.

Example

Members and books are involved in the business process defined for the library loans as shown in Fig. 4.4. The *Physical Objects* represented in the figure are associated with the corresponding classes defined in the data perspective. For the *pick up book* task, members are identified using *paper* and *radio* mediums. Later this could imply to attach and RFID label to the member card and include a written identity number on it.

The use of *Physical Objects* for correlation is also illustrated in the example. Books –identified in a radio medium– are used to determine which is the process instance for the *return book* task. This implies that, for a member, returning a book could be as easy as dropping it in a box.

Objects involved in correlation are represented with a thick border in the figure. However, the design of a concrete syntax for the DSL falls out of the

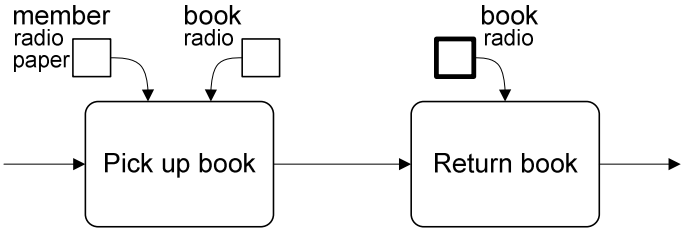

Fig. 4.4. An example of business process model

scope of the present work. So the syntax used for illustrating the DSL by no means is normative.

4.3.3 Technological Perspective

The technological perspective represents the possibilities that exist to connect the data elements with the services used for identification. Several primitives are defined to make this relationship as flexible as possible. The defined primitives are *mediums*, *codifications* and *technologies*. The identity of an element is expressed following a *codification* at the Digital Space. This identity can be represented in the Physical Space in different *mediums* and some *technologies* are able to handle the movement of information between both spaces. Figure 4.5 illustrates the relationship between these concepts by the excerpt of the DSL metamodel that captures the technological perspective.

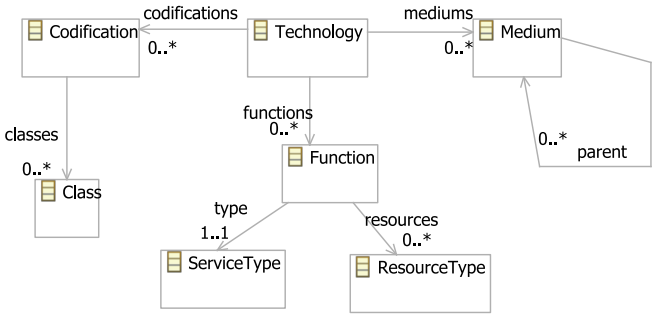

Fig. 4.5. Metamodel for the technological perspective

The identity of objects can be present in the physical world in different forms. *Mediums* are physical supports for identifiers. For instance, a sequence of bars on a paper or a radio wave emitted by an RFID tag.

Mediums can be defined as needed. For each medium it should be indicated the codifications and technologies it supports. *Codifications* represent a family of identifiers. Codifications usually are based on numbering schemata such as Electronic Product Code (EPC) and are used to encode some of the *classes* defined in the data perspective.

Technologies in this context are defined as families of services that can be used for a certain codification in a given medium. A technology can support different service types –e.g., minters and capturers– and to do so, several resource types can be required. The *Function* metaclass defined in the metamodel represents this relationship between *service types*, *technology* and *resource type*.

Example

For the library example, different mediums are defined –see Fig. 4.6. Mediums are related in a hierarchy to specialize them in groups with shared properties. In the example, *numbers on paper* is a kind of *paper* medium. The *paper* medium requires line of sight to allow the extraction of the identifier. Two specializations are defined to distinguish an identifier expressed by means of numbers –readable by humans– from one expressed by means of an image –more likely to be processed by a machine.

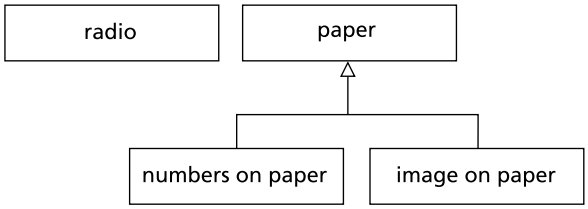

Fig. 4.6. An example of mediums in the technology perspective

Two codifications are defined in the example, *Electronic Product Code* (EPC) and *Consecutive Numbering*. EPC is the codification proposed for RFID while Consecutive Numbering codification consists in the sequence of

natural numbers. Therefore, Consecutive Numbering is much simpler than EPC, but there is not guarantee of global uniqueness of the generated identifiers in different organizations, so it is not well-suited for inter-organizational processes.

Technology	**Function**	**Resource Type**
RFID	capturer, minter	RFID antenna
Fiducials	capturer	video camera, photo camera
	minter	printer
Text label	capturer	keyboard, keypad
	minter	printer, pen

Table 4.1. Technologies and related resources

Technology	**Codification**	**Medium**
RFID	EPC	radio
Fiducials	Consecutive numbering	image on paper
Text label	Consecutive numbering, EPC	text on paper

Table 4.2. Mediums and codifications for the different technologies

The technologies considered for the example are *RFID*, *Fiducials* and a simple *text label*. The relationship among technologies and resource types is captured in Table 4.1. This information indicates which resources are needed to provide a certain service for each technology –the production of Fiducials and Text labels can be done using a printer. Technologies are considered to provide different functions. For each function several resources can be used. These resources are considered alternatives, requiring only one of them to provide the corresponding function –e.g., Fiducials can be captured either using a video or a photo camera.

Table 4.2 captures the relationship among technologies, codifications and mediums for the example. The codifications and mediums supported for each technology can be useful to detect codification incompatibilities –e.g., the *radio* and *image on paper* have no codification in common.

4.3.4 Services Perspective

The service perspective defines two main primitives, *Service Type* and *Service*. A *Service Type* represents a certain functionality and a *Service* represents a particular instantiation of a service type in a given context. For example, considering the *capturer* service type used for identification, several services can be defined such as the *mailbox capturer* –to identify new postal mail– or the *fridge capturer* –to detect when some goods are lacking in the fridge.

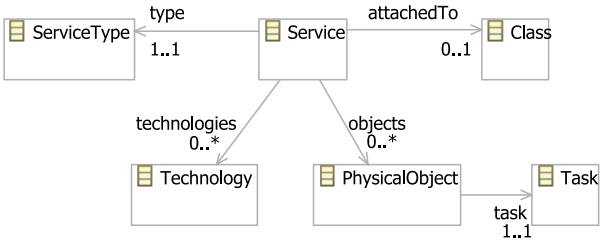

Fig. 4.7. Metamodel for the services perspective

The context of a Service is defined by the business process task this service is used in –indicating the Physical Objects that make use of this service. In addition, services can be attached to elements of the system. In the toolbox example, a tool detection service is included in each toolbox. This has two consequences, (1) each tool detection should include information about the toolbox in which it was detected, and (2) when a new toolbox is created, some resources should be used to incorporate the desired functionality –e.g., installing an RFID antenna is needed each time a new toolbox is created. These relationships are illustrated in Fig. 4.7.

A service that is not attached to any physical element can be also replicated –e.g., forming a network of capturers– but it acts as a whole, not distinguishing the replica in which the detection took place. When this context information is required, services should be attached to the element that provides the required context information.

Example.

In the library example, shelves are equipped with capturing devices in order to detect when a book is taken or returned to the library. In order to represent this, a *Shelf Detector* service is defined –as illustrated in Fig. 4.8. This service is present in every library shelf and it is used for the detection of books and members in the *Pick up book* task, and the detection of books in the *Put on shelf* task. The technology used is *RFID*.

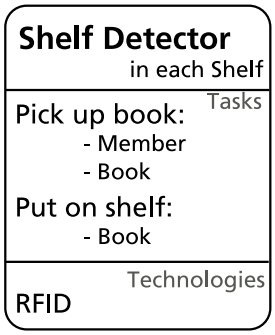

Fig. 4.8. An example of services model

Assigning more technologies implies installing more resources to each library shelf and checking codification compatibility. If fiducials were also used as a technology, on the one hand, a video camera –or a photo camera– should be installed in each shelf. On the other hand, books should use the *Consecutive Numbering* codification in addition to the EPC since Fiducials do not support the EPC codification –as stated in the technological perspective.

4.3.5 Interaction Perspective

Users can handle physical elements in different forms. The interaction perspective defines interaction patterns to permit a natural participation of users in the business process. The defined primitives in this perspective –see Fig. 4.9– extend the *Task* and the *Physical Object* concepts with interaction properties.

The *Ready Signal* metaclass defines the possible ways in which the completion of a task or the detection of an element are triggered. The following patterns are considered:

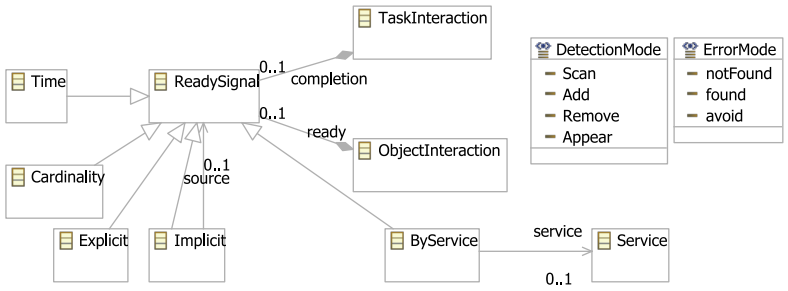

Fig. 4.9. Metamodel for the interaction perspective

Cardinality. Once a certain number of elements are present, the detection is considered complete. The number of elements should be configured. The default cardinality is 1, implying that once any element is detected, the capture is finished.

Time. Since the task is started, there is a limited amount of seconds to complete the task. At the end of this time, the elements detected in this current moment are the ones captured and task is considered automatically finished.

Explicit. Mechanisms are provided to users in order to allow them to indicate when detection is finished and a task is completed. This method provides users with more control of the process, but the process efficiency is penalized.

Implicit. The completion in the detection of a certain object, can trigger the completion of the current task or object detection.

By Service. The execution of a service –not related with identification– can trigger the completion of the current task or object detection.

Physical Objects in addition have a boolean attribute to indicate if the detection requires to provide *feedback* to users. Is worth noting that there could be a mismatch between the multiplicity offered by the identification technology and the one desired –as indicated in the business process model. For example, barcodes –readable only one by one– are commonly used at supermarkets to check the content of a shopping cart –normally containing several products. On the contrary, RFID-enabled passports are read one by one, being RFID technology capable of reading multiple items at a time.

In order to handle this mismatch, the way in which the user interacts with capturers can be also specified indicating one of the following modes:

Scan. The detected elements are those identified in a given instant. Capturing many objects at the same time is only possible if the underlying technology supports it.

Add. Each element identified is added to a selection. When the ready signal is produced, all the detected elements are considered as captured. This is the mechanism traditional supermarkets use to indicate to the system the products of a shopping cart.

Remove. An element is detected when, being detected in a given moment, it is removed and not detected again. This mechanism is useful for technologies that allow the detection of multiple elements, when the interest is in which element becomes absent.

Appear. Elements are detected when they are identified but the reader must be empty before and after the detection. Multiple elements are not allowed, resulting in no detection. This method is useful to offer an explicit identification.

Another important aspect to specify in interaction terms is the error handling mechanisms used. When retrieving the information associated to a detected element, some errors can be produced. The detected error pattens are described below.

Error when not found. In the normal case is considered an error that an element lacks associated information. The physical identifier can be read by the system but the system has no information about this element. This error is probably caused by skipping some process step where information about the element is created.

Error when found. This approach expects that elements are not associated with any information when they are identified in order to create this information. In this case an error is considered when information is found. The detection of an element triggers the creation of a new piece of information attached to this element.

Avoid errors. A combination of both behaviors is also possible. If a piece of information is found for an element, this information is selected. In case it is not found, a new piece of information is created.

Example

An excerpt of the interaction patterns for the library example are illustrated in Fig. 4.10. The *Pick up book* task uses an implicit completion pattern, so it is completed when a book and a member are detected.

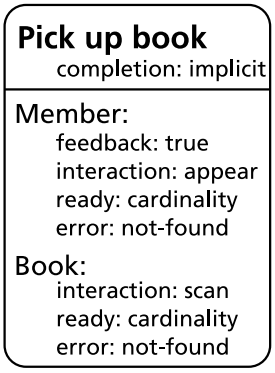

Fig. 4.10. An example of interaction model

The detection of members in this task is based on cardinality –one member is detected– and feedback is provided to the member when this happens to make explicit the interaction. In addition, members should be previously registered in the system, if a non-registered member is detected, an error is produced.

The interaction mechanisms chosen –appear for members and scan for books– allows for the following way of acting. Members can put the book on a special part of the shelf and by approaching their member cart –if no other member card is near the reader–, the task is completed providing feedback to the user.

4.3.6 Policy Properties

In order to capture the different ways in which the system can behave, properties are defined. These properties are used to qualify resource types regarding its fulfillment degree. When some requirements –expressed by means of assertions contained in a policy– are stated, the resources that better fulfill these requirements are chosen.

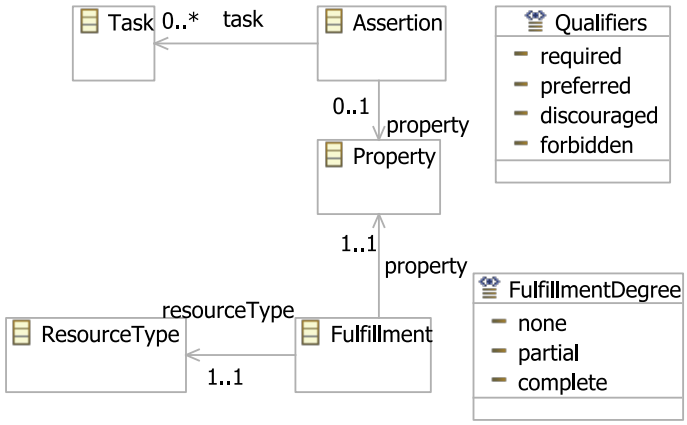

Fig. 4.11. Metamodel for the policy properties perspective

As it is shown in Fig. 4.11, assertions and property fulfillment is expressed in terms of properties. Three degrees are considered for expressing the fulfillment –none, partial and complete. However, four different qualifiers –required, preferred, discouraged and forbidden– can be used for assertions.

The definition of policies enables a dynamic behavior for system services that provides a great flexibility in the evolution of the system. If a new resource type is also supporting a given property, it only has to be qualified in order to take profit of its capabilities.

Example

In the library example, each time a member takes a book, feedback is provided. In this way, the process of borrowing a book is notified to the member. Notification can be performed in different ways such as using a light bulb that flashes or a a speaker that makes a beep. Considering the silent ambient of a library, the use of a light is preferred. However, some visually-impaired members require a feedback mechanism not visually demanding.

In order to capture this, in the example –see Fig. 4.12– a service is defined to perform the feedback and it is associated to a *Beepers* technology. This technology represents all the devices that can be used to provide a warning to the user by means of any sense –e.g., visually, acoustically, vibration, etc.

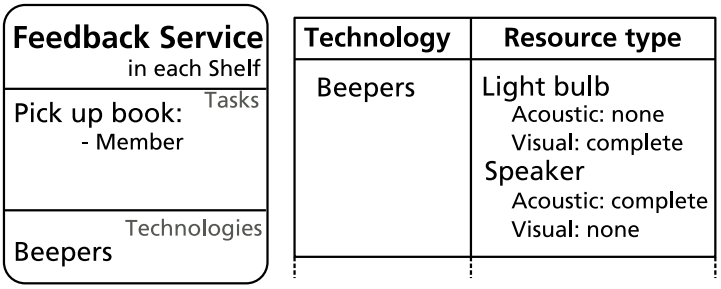

Fig. 4.12. An example of resource qualification to support policies

In the example the considered properties are the acoustic and visual behavior offered by the considered resource types –a light bulb and a speaker.

Given this resource description, a general policy could be defined where an *acoustic* behavior is *discouraged*. With these policy feedback would be offered using light as default. A policy for visually-impaired users can determine that the *visual* behavior should be *forbidden* for them. In this way, the system can adapt to the particular needs of each user, using the speaker only when it is really needed.

4.4 Conclusions

In order to apply MDE principles, this chapter provides concept formalization and abstraction raise for the development of the Internet of Things. The **architectural concepts are formalized** by means of metamodeling. In this way, the architecture can be used at modeling level. This architectural definitions can be transformed automatically in a specific technological solution by means of the **glue code generation** that automates the technological mapping. So, developers have only to deal with specific business logic not worrying about architectural constraints.

A DSL is defined in order to face the specification of requirements about the physical-virtual connection by means of concepts that can be easily understood by the different stakeholders. These concepts are also formalized using metamodels. This constitutes the foundation for the development method introduced in next chapter.

5

A Model-based Development Process

The modeling primitives defined in Chapter 4 are useful for capturing the requirements regarding business processes where real-world elements are involved. The present chapter introduces a development process that determines how to define these models and from them obtain a technological solution based on the architecture defined in Chapter 3 in a systematic way.

The presented development process has been designed to allow a clear **separation of concerns** and **minimize the impact of changes** in requirements. On the one hand, the elaboration of the different models can be achieved by different development groups with different skills. For example, the definition of the business process activities does not require any knowledge about Auto-ID technologies. On the other hand, the process follows an iterative and incremental approach for development, so the complexity of requirement changes can be handled effectively.

Is worth noting that the presented method is only focused on identification aspects. In order to derive a complete software solution, different aspects should be also considered depending on the domain needs. Techniques to model these aspects can be selected from the literature (Krogstie et al., 2007) depending on the desired expressivity.

The remainder of this chapter is structured as follows. Section 5.1 provides an overview of the development process. Section 5.2 defines the steps that should be followed in order to capture system requirements by means of modeling primitives. Section 5.3 provides the guidelines for verifying the obtained specifications. Section 5.4 defines the steps for translating the system

specifications to the final software solution. Finally, Section 5.5 concludes the chapter.

5.1 Method Overview

The present section provides an overview of the model-based development method introduced in this chapter. The development method is formed by a set of steps carried out by different actors. The roles involved in the development and the different coordinated activities that constitute the development process are detailed below.

5.1.1 Roles in the Development

Different roles are involved in the development process in order to provide flexibility in the organization according to the project size. Development roles can all be played by the same person if the project is small, or they can be shared among different groups of developers for large projects. The method considers the following five roles: Business Analyst, System Analyst, System Architect, System Developer and Driver Developer.

Business Analyst specifies the business process that the system is intended to support. *System Analyst* defines how system functionality is organized, how the users are going to interact with the system and the policy definitions. *System Architect* defines which resources are used to offer the functionality needed by the system. *System Developer* translates the knowledge captured in the defined models and expresses it in architecture concepts implementing the remaining functionality. Finally, there is a need for a *Driver Developer* whose mission is to develop the software adapters to enable the different devices to be used in the system. For instance, if the system is implemented using Web Services, a Web Service wrapper should be created for the devices that are not based on this technology.

5.1.2 Development Phases

Models are the main assets in the process. System requirements regarding identification in the business process are captured by models. The method

proposes the use of familiar notations for the domain experts using Domain Specific Languages (DSLs) (van Deursen et al., 2000), the validation of the system prior to its construction and the systematization in the development of the final code for specific platforms. The method proposes three stages:

DSL-based specification of the system. Using the modeling primitives defined in Chapter 4, the requirements for the system are captured. This phase requires several iterations to ensure that the captured requirements fulfill the real needs. Different roles are involved in this stage including the *Business Analyst*, *System Analyst* and *System Architect*.

Model-based verification. System specification should be analyzed to check some constraints imposed by technological limitations. This is done automatically thanks to model validation tools.

Translation to architecture concepts. Once the requirements stated by models have been validated, a model of the system is defined at architecture level. Guidelines are provided to systematize the development of the architecture model. Once the architecture model is defined, glue code is generated automatically and the remaining functionality is implemented. *System Developer* and *Driver Developer* roles are involved in this stage.

5.2 DSL-based Specification

The explicit definition of the business process is central to the system specification. Business process tasks become the basic unit of work during development. Once a set of tasks is detected as part of the business process, the development process can be applied to this set of tasks. Therefore, tasks are used to modularize the system functionality and organize the development. Following the development process iteratively, a prototype of the system– including the considered tasks– is obtained. Then, feedback from users drives changes in requirements and the detection of new tasks of the process. In this way the initial prototype can be evolved to the final solution.

Figure 5.1 gives an overview of the development activities considered in the method using BPMN notation. Activities and the roles in charge of them are illustrated in the figure.

1. The first step in the system specification is the **definition of the business process** by the *Business Analyst*. This definition is not intended to

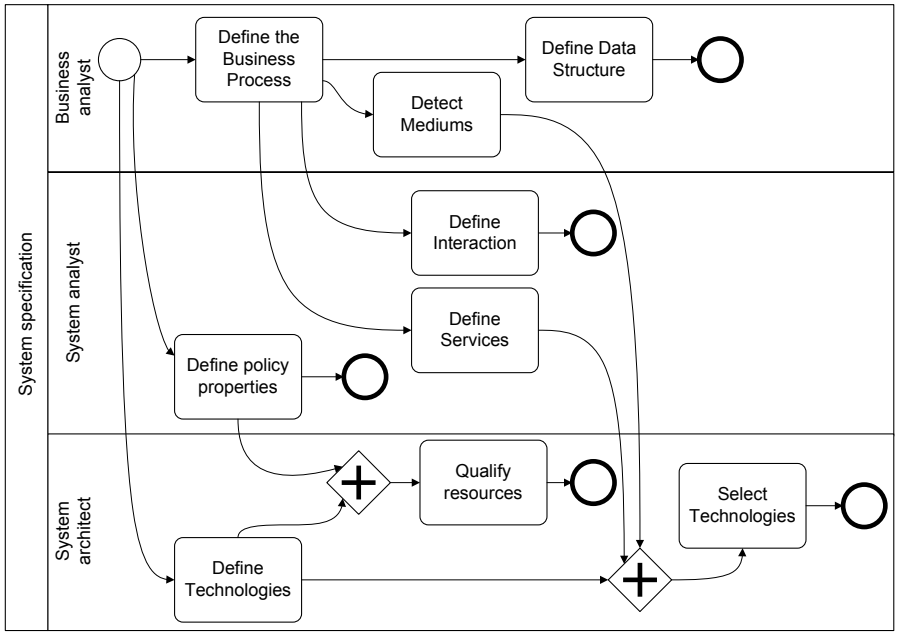

Fig. 5.1. Activities for the system specification

be complete, it can include from one task to the whole process. For the tasks involving real-world elements, the mediums used should be identified. In this way, the Business Analyst states the requirements for identification.

2. The *System Analyst* defines the services present in the system, the interaction mechanisms offered by the system, and the relevant policy properties and assertions. This supposes organizing system functionality, defining how users interact with identification services and defining high level properties that are considered relevant for the system.
3. The *System Architect* selects the most appropriate technologies for each identification service. In order to do this, a technological catalogue is elaborated. Then, the *System Architect* assigns the most adequate technology for each service. In addition, policy property fulfillment is defined for each resource type.

The following sections provide more detail of the different steps of the development process.

5.2.1 Business Process Specification

The specification of the business process activities is useful to detect the tasks where real-world elements are involved. Physical elements can be involved in the process because of their detection or their creation. In any case, regarding the **information flow**, it should be defined (1) the structure of the associated information and (2) the requirements related to their identification.

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. This also depends on the identification granularity level used in the system. 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.

Requirements for identification. Requirements for identification are defined by indicating the mediums used for each Physical Object. The medium concept is still technological-independent but the definition of mediums is useful for guiding the later technological decisions.

For the Business Analyst the *image on paper* medium is related to an identification mechanism that requires direct line-of-sight and it is easy to process by machines. But 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 phone.

In addition, the need for alternative identification –e.g. enclose the machine processable identifier with a more human-usable one– is expressed by indicating multiple mediums. By combining different mediums their weaknesses can be palliated.

Regarding **process guidance**, it is important to determine whether *Physical Objects* are used for correlation or not. When used for correlation, the process can become more fluent specially if the detection of a certain object can determine unambiguously the process instance. In the library example, when returning a book it is clear to which process instance it belongs since one book can be involved in only one loan at a time.

5.2.2 Service Definition

In order to support the different business process tasks, services are defined. Services of very different kinds exist, but this work is specially focused on identification-related service types.

Services can be shared and replicated. On the one hand, **identification services can be shared** among different tasks and/or *Physical Objects*. For example, due to economical restrictions a company can use a single printing service for the minting of all the identification labels for any task of their business processes.

On the other hand, services are not restricted to be offered in a single point in space and time. A service defined once in the specification can be accessible from different contexts. However, if the context from which it is accessed is relevant, it should be indicated in the specification. In order to do so, services can be attached to a data element –e.g., a reading service in each office of a department or each shelf of the library. This information is interesting at runtime –the system should know in which context the detection is produced– but also at deploy-time for guiding the system installation.

Identification services –*Capturers*, *Minters*, *Data Providers* and *Transformations*– provide a well-known functionality. However, for the rest of the services, some methodological approach such as PervML (Muñoz & Pelechano, 2005) could be followed for their specification. How to integrate the service definition within the presented development process falls out of the scope of the present work.

5.2.3 Technology Description

The technological description captures the requirements for identification at technological level. The existing technological solutions, the mediums they support and the resources they need constitute the **technological catalogue**. This is a reusable asset that can be useful for projects of very different domains. Each time a new technology is available, it should be included in the catalogue.

The *System Architect* given the mediums detected by the *Business Analyst* can choose the best technology that fulfills the requirements stated for each medium. This include the compatibility of supported codifications but also other requirements stated in natural language –e.g., not requiring line-of-sight for identification, to be human-processable, etc.

The *System Architect* uses the technology catalogue to assign the resources each service demands. For example, if the catalogue states that a RFID-based capturer requires an RFID antenna, and the system has a capturer service for each room of a house, then an RFID antenna should be placed in each house room for this service. The catalogue restricts the possible resources that can be assigned to a service to allow only the technologically compatible.

Many types of resource can be assigned to a service. If these services are considered alternatives, policy property fulfillment can be indicated to differentiate them. In this way, the system can choose, between the different alternatives, the one that better fulfils the policy requirements in a given moment.

5.2.4 Interaction Specification

One of the most important properties for the systems this work is dealing with is the unobtrusiveness. The presence of an Information System should not force people to act artificially. When defining the interaction mechanisms for each *Physical Object* in a business process, the deviation from the natural way of performing an activity should be kept to a minimum, offering the most natural way of interaction to users.

The *System Analyst* should determine for each task if the completion of the task can be determined automatically, or it requires explicit confirmation by the user of the system. For each Physical Object, interaction patterns

should be selected according to the natural affordances of objects. For example, identity cards are usually shown and hidden in a one-by-one basis; on the contrary, many products are normally present in a shopping cart at the same time –so detecting all of them at the same simultaneously is preferred.

The error conditions are another aspect that has an impact in the process fluency. *System Analyst* should consider what happens when information is lacking for some element, and choose the most adequate error pattern.

5.2.5 Policy Definition

During the development process, the *System Analyst* is in charge of detecting some properties that determine the ways –e.g., silent, resource-saving, etc.– in which the system can operate. *System Architects* define how the different resources fulfill those properties. If a resource is qualified as having a *complete* support for *silent* behavior, this would be preferred when a *silent* behavior is required.

The standardization of system properties would promote the interoperability among systems. Thanks to standard properties, *Physical Objects* can include policies that would be understood by any system the object enters. So this guarantees that the system behavior would respect the object needs when it crosses the boundaries of a system. If a global standardization process is not achievable, the definition of a common vocabulary among business partners should be a must.

The definition of policies can be also used to detect flaws in the system design. The inability of the system to offer a certain behavior indicates the need for new resources that are capable of offering it. The need for new resources can be detected if there is no resource that fulfills some of the defined policy properties. If the system supports the *silent* behavior and the only resource available for a given service does not fulfill this property, it could be desirable to add a new one that offers a *silent* behavior.

5.3 Model-based Verification

One of the most important use of models is to **reason about the system** they describe. Model-based verification can ensure that the system is valid

prior to its construction. This section introduces some validations in order to ensure that some technological decisions made during system specification are consistent.

The presented properties to check in this section are applicable to any system fitting the architecture defined in this work. However, there is also a need for the definition of domain-specific validations for particular needs. For example, a company can define its own validations to ensure the implantation cost for the system is not excessive. During development of a new system, the interesting properties to check should be defined and expressed in a query language to allow the automatic validation of models.

5.3.1 Technology Compatibility

In a business process in the context of the Internet of Things, objects are constantly moving between digital and physical worlds. The used technologies should guarantee that once an identity is transferred to the physical world it could be retrieved again.

In order to do so, for each entry or exit point a given Physical Object moves across the physical-visual linkage, there should be at least a codification in common. To verify this property Object Constraint Language (OCL) (OMG, 2005) can be used. The expression that verifies this property is the following:

```
let objects : Sequence(PhysicalObject) =
PhysicalObject.allInstances() in
objects->forAll(p|objects->select(x|x<>p and p.class = x.class)->
    forAll(q |p.service.codifications -> intersection(
    q.service.codification ) -> notEmpty() )
)
```

5.3.2 Objects Not Minted

A Physical Object, in order to be identified, requires to have an identifier attached to it. In the common case objects are minted prior to their identifiers are captured. So it could be useful to detect Physical Objects modeled in a business process where their identifier is not generated. A query to detect this circumstance is defined below using OCL:

```
let objects : Sequence(PhysicalObject) =
PhysicalObject.allInstances() in
objects->select(x|x.direction= Dir:in and
   not object->exists(y|y.direction = Dir:out
       and x.class = y.class))
```

Is worth noting that this circumstance is not always an error. Objects can be minted in a different process or by an external system. However, the previous query is also useful to detect them. In this way it is easy to obtain a list of *Physical Objects* whose identification depends on external partners.

5.3.3 Unsupported Properties

In order to allow system adaptation, several policy properties are defined for the system during development. If a property is demanded for a given Physical Object, a service cannot use a resource that does not support the property. Therefore is interesting to know if there is any service that becomes unusable when a certain property is required.

The OCL query that checks this condition for all the declared properties and the specified resources is detailed below:

```
Services.allInstances().type.function->forAll(f|
  Properties.allInstances()->forAll(p|
    f.resources.properties->select(x|
    x.fulfillment = Fulfillment:None and x=p)->size()
    < f.resources.size()
  )
)
```

5.4 Translation into Architecture Concepts

In the present section we illustrate how the different concepts defined in the DSL can be mapped to the target architecture. In this way, the production of the final system is systematized. The last stage of the development process is composed by three steps: (1) define the architecture model based on the DSL specification, (2) generate the glue code and (3) complete the generated code by the remaining functionality.

The glue code generation was described in Chapter 4. In this section, it is described the mapping between DSL concepts and the architecture, defining for each element which architectonic components should be defined and how to complete their implementation.

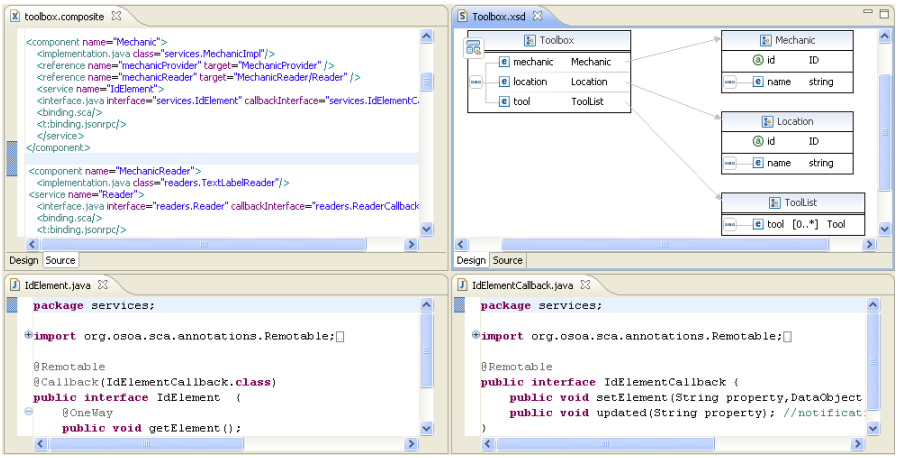

Fig. 5.2. Development of the Smart Toolbox prototype

5.4.1 Business Process Specification

From the business process specification, **an executable definition of the process** should be derived. WS-BPEL can be used for this purpose. This work is particularly interested in how to translate the primitives introduced as BPMN extensions to cope with physical elements into WS-BPEL constructs. For the rest of standard BPMN constructs some of the existing mappings between BPMN and WS-BPEL can be used (OMG, 2006; Ouyang et al., 2006; Recker & Mendling, 2006; Giner et al., 2007a).

In order to support the presence of *Physical Objects* in WS-BPEL, two aspects should be implemented. On the one hand, the message interchange between the Information system and the Business Process Execution system should be defined using WS-BPEL. In this way, the process notifies the system that it is waiting for the detection or minting of an object in the context of a

certain task. On the other hand, correlation sets should be defined according to the correlation properties of each *Physical Object*.

BPMN tasks that involve *Physical Objects* result in a sequence of a pair of WS-BPEL operations. First, an *invoke* operation is defined to indicate the identification system which elements are expected for a given task. This operation sends a message to the *Task Manager* component with information related to the task context. Next operation is a receive operation to wait for the identification system response.

For each detected task in the business process specification, a *Task Processing Component* is required in the architecture and each *Physical Object* participating in the task requires an *Identification Component* in the architecture. Figure 5.2 shows at its bottom the definition of *IdElement* interface –and its corresponding callback interface– that is implemented by *Identification Components*.

The *Task Processing Component* is wired with the Identification components –this is done by the Glue Code Generation– and information retrieved from them should be processed to generate the return message that the process is waiting for. The construction of the message is performed using the SDO API, and the *System Developer* is in charge of implementing it. In addition, Identification Components require to choose their interaction and error management patterns. The corresponding SCA components can be configured using a property to indicate this. However, if some custom behavior is desired –e.g., to obtain personalized error messages– this would require also manual development.

The use of WS-BPEL requires offering system functionality as Web Services. This is not a major problem since SCA defines a Web Service binding that permits to expose the system functionality using this technology.

5.4.2 Data Structure Specification

Information structure captured in this dimension can be easily expressed as an XML Schema –Fig. 5.2 shows at the top-right side the Eclipse XSD editor used. The defined XML Schema is used by the business process definition and the identification infrastructure. On the one hand, the message structure is based on this schema when a message is defined for the process. On the

other hand, the XML Schema is imported by the SDO framework to define the system data types to be used in the Information System.

The different data elements require a *Data Provider* service to retrieve the information from different data sources. The System Developer should implement the accessing mechanism for each information repository.

Since SDO is independent of the back-end data repository, the same data definition allows the most natural way of working in each field. XML and related languages such as XPath can be used in the process definition while at the system data is accessed as Java objects.

5.4.3 Service Specification

A service at specification level corresponds to many services at the architecture level. For each Physical Object involved in a specification Service, a *Service Component* is defined in the architecture. However, physical objects are not mapped to services in a one-by-one basis. Several Physical Objects included in the same specification service can be mapped to the same Service Component if all of them share the same resources to support the used technology –e.g: a barcode and a fiducial-based object will be minted by the same printer, provided both are specified in the same service in the system specification.

Depending on the resources used by a service, the implementation of the Service Component would be different. The implementation class –indicated in the SCA configuration file for each component– used for the component should provide access to the resource functionality. To facilitate the development, Java Interfaces are defined for the different service types. The *Driver Developer* should implement the adapters for each particular resource following this interfaces.

The connection of components is defined in the SCA configuration file. This file uses XML to configure the different components involved in the system –at the upper-left side of Fig. 5.2 is an example of configuration for the Smart Toolbox prototype.

5.4.4 Interaction Specification

For each of the *Identification Components* defined, interaction patterns should be implemented to provide the corresponding behavior. Capturer services no-

tify all the detection events to Identification Components, however to implement the different interaction patterns these events should be processed in a different way.

For implementing the different interaction patterns normally two list of elements are handled, corresponding to *detected* and the *selected* element. The interaction patterns considered in the DSL are implemented in the following way:

Scan. For the *scan* pattern both lists contain the same elements, each element added or removed to the detected elements is respectively added or removed from the selected elements list.

Add. For the *add* pattern, each time an element is detected, it is included in the selected list.

Remove. The *remove* pattern implementation is opposed to the *add* one, each time an element is removed form the detected list it is added to the selected list. The difference is that, if an element is detected again, then it is removed from the selection.

Appear. The *appear* pattern, is a combination of the *add* and *remove* patterns. An element is added when it is removed but the detected element list should be empty and only one object is allowed at a time in it.

In addition, the **completion of tasks** can be either triggered automatically or by users. In case it is triggered automatically –by cardinality, time, the detection of an object or the execution of a service–, it should be considered in the implementation of the *Identification Component*. In case task completion is informed by users –implicit completion– a User Interface component should be defined to allow the user to indicate when a task is complete. Many technological options exist for this and the decision would depend on the particular requirements of each application.

Error management patterns should be also implemented when defining the retrieval of information for *Identification Components* depending on the presence of information and the defined pattern in the specification for each detected element. Different means exist to notify errors, such as using the user interface, the execution of a service or a log.

5.4.5 Policy Definition

Architecture services that allow the access to resources should be labeled for their fulfillment of the different policy properties. Since the resource used cannot be decided at design time, the services with several possible resources are wired at runtime – the SCA attribute *wiredByImpl* should be set to false. In addition, this components should implement an interface that allows for the query of their property fulfillment capabilities.

The specification of policies and resource fulfillment is expressed declaratively by means of XML files. These files can be modified during the operation of the system.

5.5 Conclusions

This chapter connects the architecture and the DSL defined in previous chapters. A model-based development method is defined where a **system specification using the DSL** defines the system requirements, **model-based verification** is applied to check consistency for the specification, and guidelines are provided to translate the specification into the architecture previously defined.

In this way, a systematic path between specification and implementation is defined in the present chapter. Business process can be decomposed and their requirements specified from a high level of abstraction, to iteratively obtain the specific software solution.

Is worth noting that although the present development method can be used for the development of systems from scratch, it is also applicable to already existing systems. Since the business process task is used as development units, the process can be applied only to the new tasks or the modified ones. Next chapter provides some results of the application of the method in order to validate the proposal.

6

Validation of the Proposal

The development method presented in this work has been put into practice in order to validate the proposal. The objectives of the validation are (1) checking that the defined architecture could be used to implement the kind of applications targeted in this work, and (2) verify that the modeling primitives are expressive enough to capture the required knowledge about the system to drive its development.

In order to do so, two representative case studies have been selected for the application of the proposal. Case studies were not defined from scratch but based on the literature to ensure their representativeness. These case studies came from both, the Internet of Things area and the Business Process Management area.

The original case studies do not consider either the business process modeling or the integration of real-world elements, but for the purpose of the present work these case studies have been extended to incorporate the lacking aspects. On the one hand, from the Internet of Things field, the Smart Toolbox case study provides support to aircraft maintenance operations. The original case study does not consider an explicit definition of business process so it has been incorporated to better fit the application domain. On the other hand, the smart library is based on the classical library scenario where books are borrowed by library members but introducing Auto-ID technologies to fit in the Internet of Things domain.

Provided that the developed case studies are representative enough for the applications domain, the development of such case studies becomes a good

test for the present development method. Therefore, by successfully developing the case studies, the architecture components and the modeling primitives are proven capable of representing the system at specification and implementation levels. In this way, the applicability of the method is guaranteed.

The rest of this chapter is organized as follows: Section 6.1 presents the Smart Toolbox, a case study from the Internet of Things area where an explicit business process definition has been introduced. Section 6.2 presents the Smart Library, a case study for the improvement of library loan process where physical elements are seamlessly integrated. Finally, Section 6.3 concludes the chapter.

6.1 Smart Toolbox

The Smart Toolbox case study was originally introduced by Lampe (Lampe et al., 2004) as an example of application of Auto-ID technologies to improve industrial processes. The proposed scenario is based on the maintenance, repair, and overhaul (MRO) process of aircrafts. MRO activities are strictly regulated and their cost represents 12% of the total operating costs of an aircraft. Thus, companies are interested in making the execution of MRO more efficient.

Tool management is improved for the MRO process in this case study. The original MRO process includes several manual tasks related to the identification of tools. Mechanics have to check whether all tools are in their box and no tools have been exchanged with colleagues. If a tool is missing, the aircraft is checked until the tool is found, resulting in a time-consuming activity.

The proposed solution is to monitor the tools used for aircraft maintenance tasks by means of Auto-ID technologies. During aircraft reparations, tools are monitored preventing them to be lost, and therefore, causing potential damage or introducing delays in the process. In order to do so, tools are labeled and each toolbox is equipped with an identification system. The original case study is based on RFID. However, it has been extended to introduce different Auto-ID technologies for the identification of tools, mechanics and locations.

The toolbox is the main element in the present case study. Different information is sensed for each toolbox –see Fig. 6.1–, being its content –the tools it contains– the essential information for detecting missing tools. However, some

additional information can complement the content information in order to define better the context of the toolbox –e.g., who carries it and where it is. This contextual information is useful to determine which activity is being carried out –which one from all the planned reparations is the toolbox involved in.

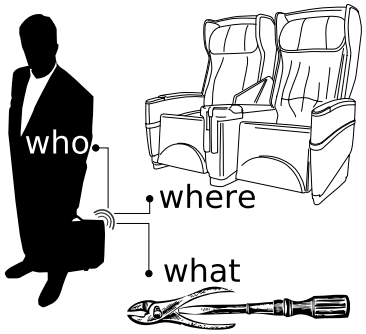

Fig. 6.1. Illustration of the Smart Toolbox scenario

The Smart Toolbox is part of a more generic set of applications, corresponding to the Smart Box concept (Floerkemeier et al., 2003). Smart Box applications are based on Auto-ID, but not all of them are intended to support a business processes. In this case, the original Smart Toolbox case study does not consider the modeling of the business process or an explicit representation for its execution. Introducing a process definition provides two benefits respect to the original case study. On the one hand, a business process model is useful to provide an **intuitive description of the MRO process** improving its comprehensibility compared to reading manuals and legislations. On the other hand, using an executable definition, there is a guarantee that **the process is carried out as described** avoiding mismatches between process specification and the Information System implementation.

The specification of the system and some details about its implementation are provided below.

6.1.1 System Specification

The first step for the development of the case study is the specification of the system. For capturing the identification requirements of the Smart Toolbox system, the DSL defined in Chapter 4 is used.

The definition of the different aspects considered in the DSL such as business process, the technological aspects, the defined services, interaction and policy properties for the present case study are detailed below.

Business Process

The first step for the specification of the system is to make an explicit definition of the business process. The business process specification represents the MRO activities. According to the original case study, the activities comprised in the process are the following:

Customer order and planning. A MRO event is initiated with the customer order. At an agreed date the plane is brought to the hangar and the relevant documents –logbooks– are handed over to the service center. The logbooks contain information about flying hours, operating hours, starts and landings, condition of the plane and its parts as well as complaints. Based on this information the MRO tasks are planned. Some of these data were recorded automatically –e.g., flying hours–, others were recorded by the pilots or owners of the plane –e.g., starts and landings, complaints– or during the last MRO. This holds the potential of inaccuracy –e.g. valuations– or human errors. The planning is recorded in the Maintenance Planning Document (MPD) that describes the MRO tasks and for each task the necessary activities.

Procurement of parts and tools. Based on the MPD, the necessary parts and tools are determined. Missing parts can be ordered from the procurement center by using the Illustrated Parts Catalogue (IPC). Special tools can be checked out from a central tool inventory. Missing spare parts, long delivery times for parts, or misplaced tools can delay the MRO process.

Carry out MRO actions. The MRO activities are carried out according to the MPD. Some failures are only discovered in this step, which makes it necessary to complete the MPD and procure additional spare parts or tools. The mechanics use a PC to record all activities they carry out in

the discrepancies report (ROD). For each activity the mechanics identify all parts that are subject to inspection, replacement, or repair by its serial number and describe the status of the part. The duration of the task is documented as well. The correct inspection procedures for the parts need to be looked up in the MRBs that are only available in printed form. This task is very time consuming and it is assumed that sometimes the mechanics forget it.

Control and delivery. Once the MRO is completed, an inspector checks the result. The inspector fills up an "Aircraft Certificate of Release to Service and Maintenance Statement" that describes all checks that were carried out, the repairs that were done, and the parts that were replaced. Finally, the plane is delivered to the customer.

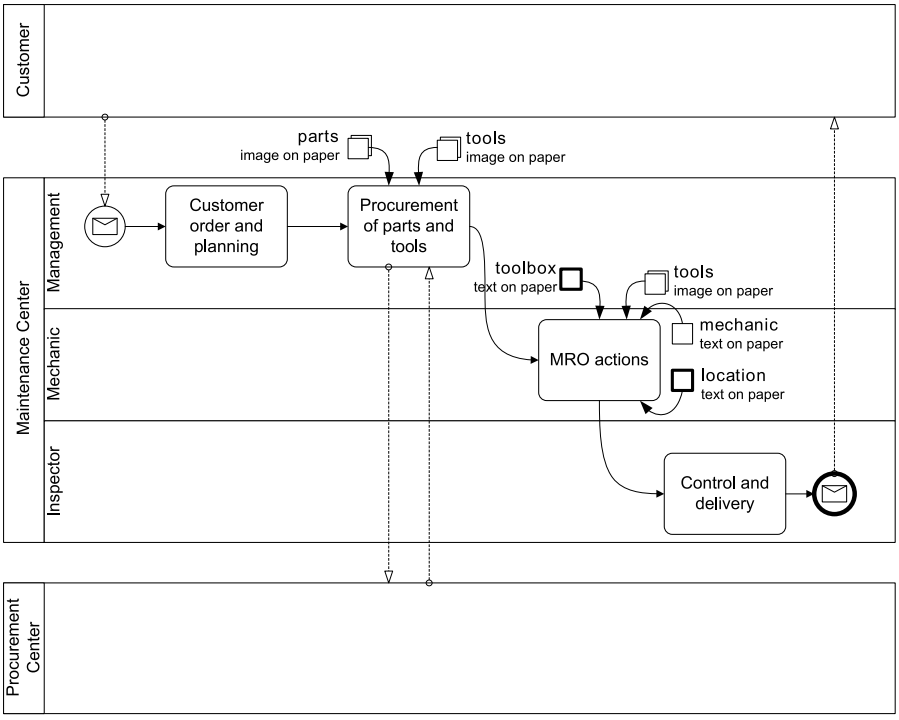

Fig. 6.2. Business process diagram for the Smart Toolbox scenario

Figure 6.2 illustrates by means of a BPMN diagram the MRO business process activities. The most interesting activities for the purpose of this work are the ones where physical elements are involved –*Procurement of parts and tools* and *Carry out MRO actions*. The remaining of the system specification is only concerned with these two tasks since identification is not relevant for the rest of the process.

The definition of mediums for the identification is also captured in Fig. 6.2. Two mediums –namely *image on paper* and *text on paper*– are defined, both based on paper tags. This is because, one of the goals for this development was the use of real technologies in the implementation, avoiding simulators. So, only cost-effective technologies are considered. Since the detection of parts and tools is the most critical one, they are identified using the *image on paper* medium. This medium although requires direct line-of-sight, provides a greater automation level compared with the *text on paper* one –used for the rest of physical elements of the case study.

Data Structure

In order to support the *Procurement of parts and tools* and *MRO actions* some information is required at the digital world. Its structure is captured in Fig. 6.3 by means of a Class Diagram. Each *Reparation* should be performed in a *Location* –such as the cabin or the left wing– of a *Plane*. Each reparation requires a certain parts and tools. *Parts* and *tools* required for a reparation are indicated by its types –e.g., a hammer or a wrench in case of tools, but not a specific hammer from all the available ones.

In the example, a *Toolbox* can be in a *Location* and it can be carried by a *Mechanic*. It can contain several *Tools* which usually are a subset of the tools that were assigned to this toolbox. There are different types of tools. A *ToolKind* represents the tools of the same type and it provides information regarding tool lifetime. When a tool is used during a longer period, it should be replaced.

Technologies

The mediums considered for the identification of the different elements are paper based –*image on paper* and *numbers on paper*. These mediums introduce the following requirements for identification:

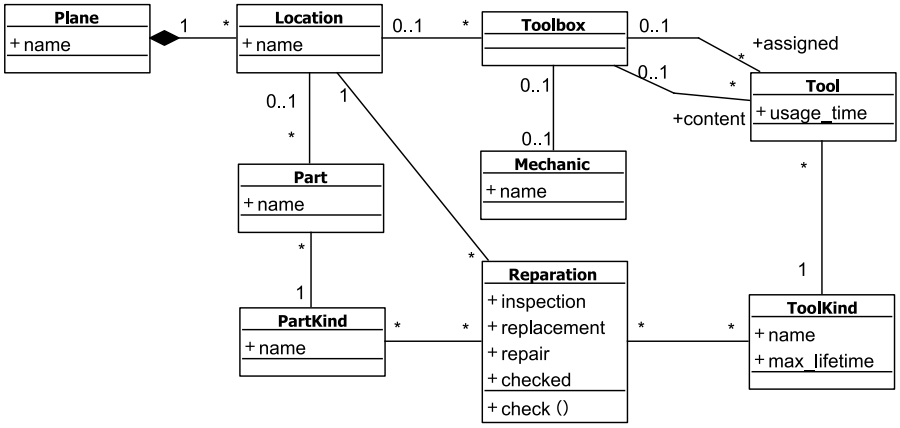

Fig. 6.3. Data model for the Smart Toolbox scenario

Cost-effective. They are cheap to produce and require simple devices for their capture. The cost depends on the quality of the used material –e.g., water-resistant plastic and inks– and the concrete technology used.

Line-of-sight required. Since identifiers are encoded graphically on a paper surface, they should be recognized optically. So it is required a direct line of sight between the reader and the label. This introduces some limitations such as requiring the absence of obstacles between the capturer and the label, the need for good lightning conditions and some limits in the distance and view field width at which a label can be identified.

Although both of the considered mediums are paper based, there are some differences between both. The *image on paper* medium offers a greater degree of automation –allowing the detection of multiple identifiers at the same time. On the one hand, *image on paper* medium is easily machine-processable but humans are not capable of processing it. On the other hand, *numbers on paper* medium is easy to process by humans but its capture is difficult to be automated.

The technologies considered for supporting the requirements introduced by the different mediums are *Text label* and *Fiducials*. These technologies demand different resources to perform identification tasks –for the *MRO actions* task, only capturers are needed. For capturing *Fiducials*, a video camera is required. *Text label* requires a typical input device such as a keyboard or a keypad with

the numeric keys. For the use of *Text label* technology, humans are responsible of transferring identifiers to the system by means of an input device. However, with *Fiducials* users do not have the need for decoding the identifier, and just placing the object in front of the reader is enough.

Regarding codifications, *Consecutive Numbering* is used for all the physical elements involved in the case study. This is a simple codification but it is suited for the current case study since it can be used in both of the considered mediums. Either numbers or images can be used to represent this codification in different technologies.

Services

Identification is present in different situations for the current case study. Fig. 6.4 provides an informal diagram of the different physical elements involved in the case study and their identification needs. The toolbox is the central element in this case study. For each toolbox it is necessary to sense its context in each moment. This includes the mechanic that is carrying the toolbox, the location at which the toolbox is and the tools that it contains. Tools and parts are also identified in their arrival to the Warehouse. Services are defined to cover these needs.

Services are specified to identify where physical elements can cross the boundary of the system. Services define where and how identification functionality is offered. Fig. 6.5 illustrate the different services for the Smart Toolbox case study. The identification services defined to support the *MRO Actions* task are *Toolbox Context*, *Toolbox Content* and *Toolbox Detector*. The *Toolbox Context* service is present in each toolbox, and it is used to detect the Mechanic and the Location using the *Text label* Technology. In order to support this technology, a keyboard can be incorporated to the toolbox. The Mechanics use this keyboard to introduce their identifiers and the identifier of the location in which they are.

The *Toolbox Content* service is also required for each toolbox. It is used to monitor the content of the toolbox. The tools contained in the toolbox are detected using *Fiducials*. This implies to install a video camera in the toolbox since an image capturing resource is required for the capture of *Fiducials* –as it is stated in the technological model of the specification. In addition to their context, toolbox should be detected too. The *Toolbox Detector* service is in

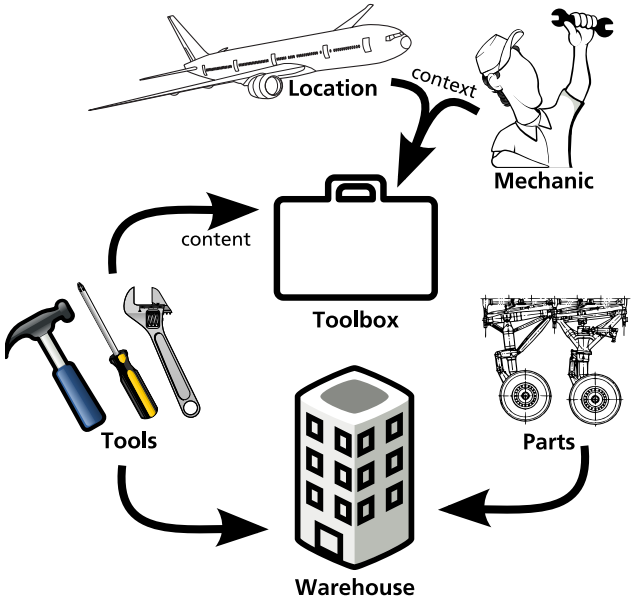

Fig. 6.4. Identification needs for the Smart Toolbox scenario

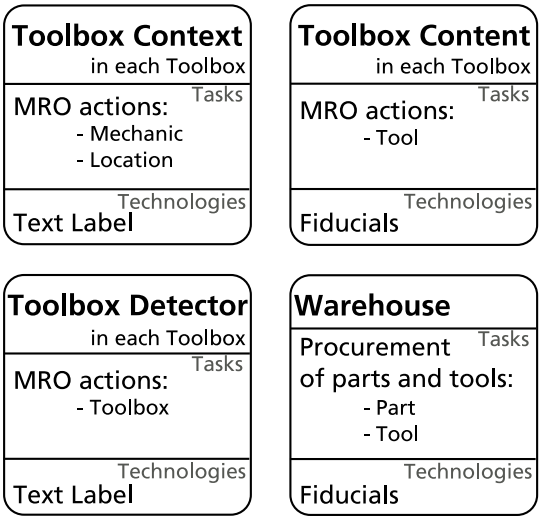

Fig. 6.5. Services model for the Smart Toolbox scenario

charge of this. By attaching this service to each toolbox, the toolbox could be identified at deployment time not requiring user intervention at run-time.

Regarding the *Procurement of parts and tools* task of the MRO process, a service is defined to detect the reception of tools and parts. The *Warehouse* service detects the available parts and tools as they arrive. Fiducials are used for the detection.

Interaction

Interaction aspects are defined for the *Physical Objects* present in the different tasks of the business process. The interaction patterns used for the tasks where physical elements are involved are represented in Fig. 6.6.

As stated in the business process description for the *Procurement of parts and tools* task, several tools and parts can be involved in the process at the same time. The big size of plane parts and the possible number of tools makes difficult to place all the received elements in the reader at the same time. In this case, the *add* interaction pattern is chosen. The received elements can be placed near the reader in small groups. All the groups are considered as if they were detected at the same time when the reading process is completed. The completion of the reading process is indicated *explicitly* by the user. However, task completion is indicated implicitly when the reading for parts and tools is complete.

In this case the error pattern used is *error when found*. Since the received material is new and should be identifiable in an individual basis, no other element can share its identifier, so there should be no information about these elements in the system.

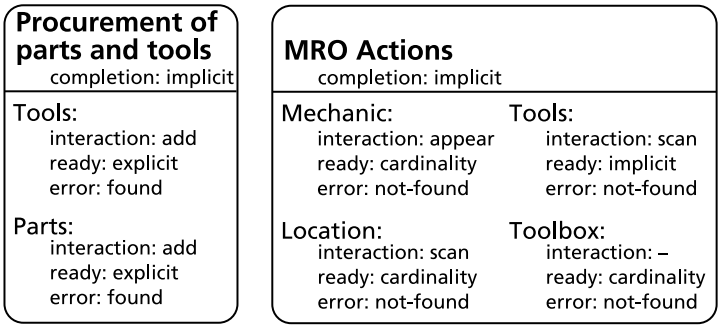

Fig. 6.6. Interaction model for the Smart Toolbox scenario

Regarding the *MRO Actions* task, interaction is defined for the different elements involved in this task. For *mechanics* and *locations* the interaction pattern selected is the *appear* pattern. The reading is considered complete by *cardinality* –detection consists in the first element that is read for each kind. The different tools are detected using the *scan* pattern. The toolbox has no identification pattern assigned since its identification is predefined –each toolbox has computing capabilities that inform of a preset identifier.

The *MRO Actions* task is considered completed when the location is changed. So completion is defined as *implicit*. When the task is completed, the toolbox content is checked to ensure tools are not missing. Information about all the detected elements should be present in the system. So *error when not found* is the error pattern defined for all the detections of this task.

Policies

During the MRO process some of the plane parts require special care due to its fragility or dangerous nature. The illumination system of the warehouse can be used for indicating the need for special care for the received parts in an unobtrusive manner. In order to do so, light of different colors are used. A white light would be used as default and a red one would indicate the materials require a special care. In order to support this behavior three steps are required. First, the *handle with care* property should be defined as a relevant property. Then, the lightning service should be specified. Finally, the resources involved in the *lightning service* should be qualified to indicate their support for the property.

The lightning service is supported by the lightning technology. Two resource types –*white light* and *red light*– are assigned to this technology to provide lightning capabilities. The *red light* resource type is qualified as providing a *complete* support for the *handle with care* property. Once resources are qualified, policies can be defined and applied. In the default policy an assertion indicating that the *handle with care* property is *discouraged* would ensure that red lightning is only used when it is really needed. In this way, *white light* is used by default and *red light* is only used when an object requires the *handle with care* property in its policy.

6.1.2 System Development

Once the system is specified, the system can be developed following the guidelines describes in Chapter 5. Apache Tuscany implementations of SCA and SDO are used for the implementation of the different components of the system and a business process execution engine is used to keep track of the process state. More detail about the different elements developed for this case study is provided below.

Business Process

The first step is to obtain an executable definition of the business process. Intalio BPMS is used as the process engine, using WS-BPEL as the process description language. The partners involved in the process are the *Customer* that wants a plane to be repaired, the *Procurement Center* that provides the tools and parts for the reparation and the *Maintenance Center* that provides support to the MRO process. For these partners a WS-BPEL *PartnerLink* element is defined. In addition, the *Identification System* is also considered in the process.

Since the tasks involved in the MRO process are performed sequentially, the WS-BPEL operations that constitute the process description are executed in a sequence. This sequence of operations is the following:

1. The process starts with a *receive* operation for the *Customer* order that triggers the initiation of the MRO process. The received message includes the relevant information about the plane –flying hours, operating hours, starts and landings, condition of the plane and its parts as well as complaints.
2. In response to the customer order, a *reply* operation returns a message to the customer indicating the day and hangar for the current reparation request. The *Maintenance Center* is in charge of assigning mechanics and tools for the reparation.
3. The *Procurement Center* is *invoked* to request the needed tools and parts for the reparation. The request message includes the number and type of the required tools and parts.
4. A *receive* operation is executed in order to make the process instance wait for the notification of tool and parts arrival. The message received

includes the information of each particular tool –including its identifier. This information is sent electronically by the *Procurement Center*, but the physical elements can suffer a delay.

5. An *invoke* operation is executed to notify the *Identification System* that several parts and tools are expected to arrive for the current reparation request.
6. A *receive* operation is executed to make the process wait for the detection of the physical parts and tools. Once received the material, it is compared with the expected ones, raising an error in case it does not match.
7. An *invoke* operation is used to notify the Identification System that the different elements involved in the *MRO actions* task –toolbox, mechanic, location and tools– are expected to be identified.
8. A *receive* operation is executed to make the process wait for the detection of the elements involved in the *MRO actions*. The content of the tools present in the toolbox are compared to the ones assigned to detect missing tools.
9. An *invoke* operation is executed to notify that the reparation task has concluded and the plane is ready for inspection.
10. A *receive* operation is executed to make the process wait for the results of the checks performed by the inspector. The received message contains the information defined in the "Aircraft Certificate of Release to Service and Maintenance Statement".
11. An *invoke* operation is executed to notify the Customer that the MRO process has concluded and the certificate is ready.

System Components

At the Identification System side, to support the implementation of the system several components are required. A *Task Processing Component* is defined for each task present in the process. Simple graphical user interface –defined as *User Interface Components*– are developed for tasks that do not integrate physical elements. These interfaces collect the required data for the corresponding *Task Processing Component* that composes the message that is returned to the *Orchestration Component* in order to allow the process to continue. These interfaces are implemented using a web form.

For the tasks with intervention of real-world elements, Task Processing Components require to define an *Identification Component* for each *Physical Object* involved in the corresponding task. Fig. 6.7 illustrates the identification-related components and how are they wired together. Is worth noting that the Toolbox Identification Component and all the dependent components should be replicated for each toolbox present in the system. This implies to deploy the corresponding components into the computing capabilities the toolbox provides and installing the physical resources they require.

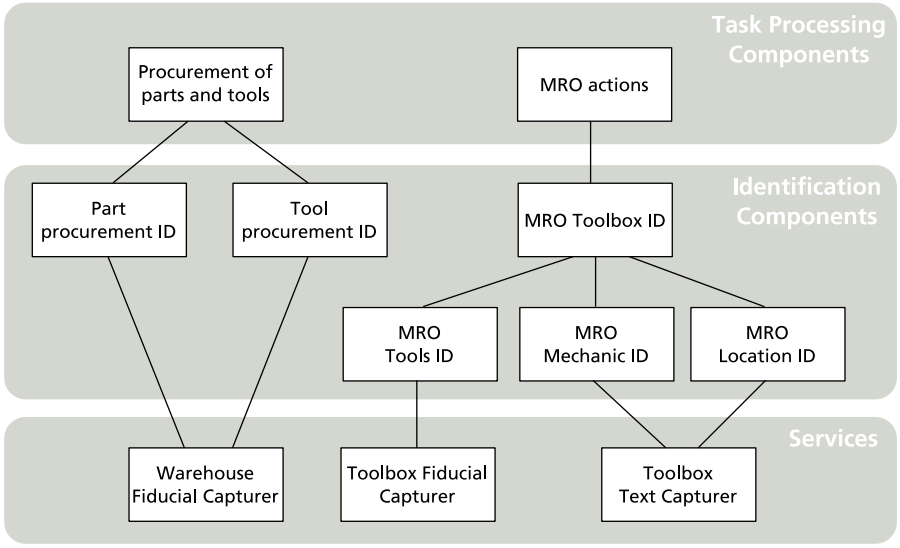

Fig. 6.7. Architectonic components for the Smart Toolbox scenario

The services defined in the specification determine how the different components are wired together. Some *Identification Components* share the underlying *Capturer Service*. Capturers are shared when they are part of the same service of the system specification. For example, since a unique service is defined for capturing mechanics and locations –see Fig. 6.5–, only a *Text Label Capturer* is defined for both –this implies using the same keyboard for identifying both kinds of elements.

The technology chosen for each service in the specification determines the implementation of each Capturing Service. The Text Capturer is implemented

as a web interface where an input box allows the introduction of text by any conventional input device.

Fiducial Capturer is implemented as a wrapper around the reacTIVision framework. This is a computer vision framework for the fast and robust tracking of fiducial markers attached onto physical objects, as well as for multi-touch finger tracking. It was mainly designed as a toolkit for the rapid development of table-based tangible user interfaces and multi-touch interactive surfaces.

Fiducials can be tracked by means of a conventional video camera. The reacTIVision framework sends OpenSound Control (OSC) messages via a UDP network socket to any connected client application. It implements the TUIO protocol, which was specially designed for transmitting the state of tangible objects and multi-touch events on a table surface. To support the different components that require fiducial-based identification for the Smart Toolbox case study, different UDP ports are used.

Data Structure

Data providers are also required for accessing the information related to the different physical elements involved in the process. A data provider is implemented for each element of the data model. SDO is used for accessing and manipulating information.

Information is structured following the data structure as it is specified in the system specification –see Fig. 6.3. Information structure is defined by means of an XML Schema and information regarding physical elements is stored in XML files. However, the use of SDO allows for different data backends.

Interaction

The developed system should support the interaction mechanisms defined in the specification –see Fig. 6.6–. Each *Identification Component* receives different detection notifications from the different Capturers. Depending on the interaction pattern defined in the system specification, these notifications are handled in a specific way. The same is true for error conditions.

User interface components are defined using the Widget implementation supported by Apache Tuscany as a means of wiring SCA components in Web 2.0 style applications. The only exception is an interface that represents the context of each toolbox –contained tools, carrying mechanic and the current location.

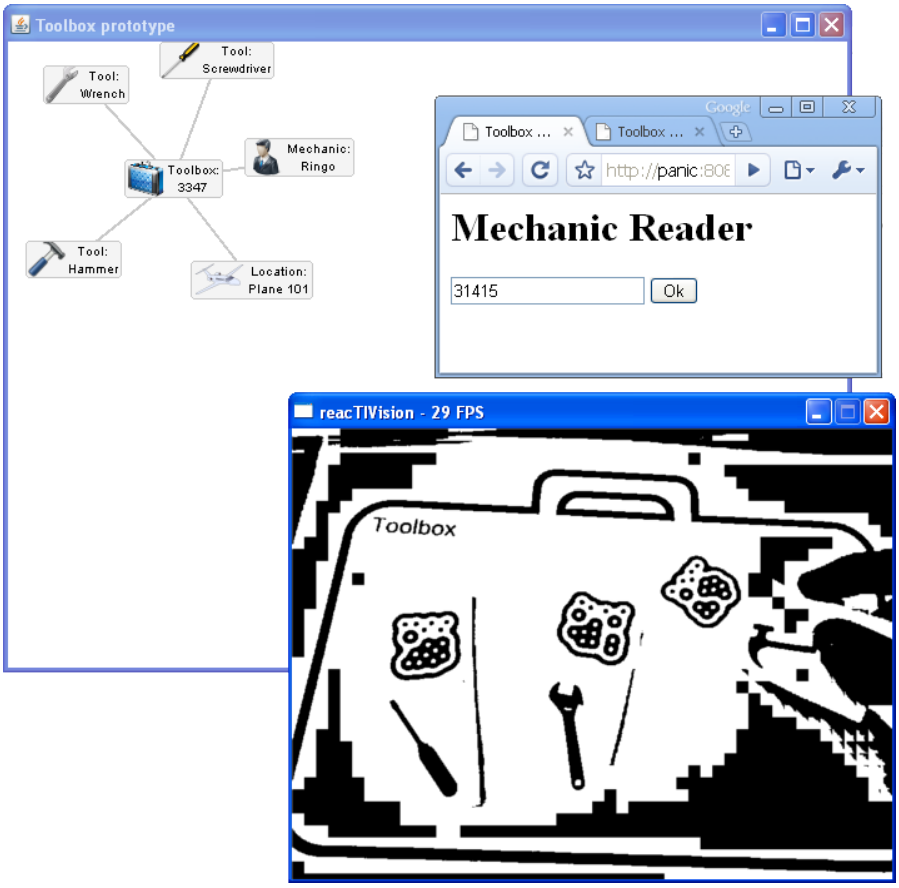

Fig. 6.8. Different interfaces for the Smart Toolbox prototype

In order to provide an intuitive representation of the toolbox context, prefuse[1], a data visualization library is used. Visualization techniques are

[1] http://prefuse.org/

quite useful for representing large amounts of data in an intuitive way. As it is illustrated in top-left side of Fig. 6.8, information is represented in a graph. Nodes contain an image that defines its type –toolbox, mechanic, location or the different *tool kinds*– and a text label with their name. The different arcs determine the associations between elements. In this way, the content of the different toolboxes can be monitored easily.

6.2 Smart Library

Prior to the arrival of Internet, libraries were the main information repositories. The need for cataloging and structuring all this information has been faced for long, becoming a classic example of Information System. However, although the central interest of libraries is in physical elements such as books, not much effort has been made to automate the linkage between physical and digital spaces in their Information Systems –being the use of barcodes the most advanced Auto-ID technology in use. With the emergent Internet of Things, a better integration between physical and digital elements is possible in order to improve the different activities that take place in the library.

One of the main activities of libraries is to lend books to their clients and control that books are returned on time. The present case study –named the Smart Library– is based on the classical library scenario but allowing the library members behave more autonomously. Each library shelf is provided of Auto-ID capabilities. Members can just take the required books and books are borrowed for the system. For returning books, members only have to place them in the returning box. Eventualities such as delays in returning books or books that are taken by a sanctioned members –or non-members– are also considered. More detail about the complete specification of the system and its implementation is provided below.

6.2.1 System Specification

The specification of the system becomes the first step in the development of the Smart Library case study. For capturing the identification requirements of the system, the DSL defined in the present work is used. The definition of business process, the technological aspects, the defined services, interaction and policy properties is detailed below.

Business Process

The business process that is followed in the Smart Library is slightly different from the traditional lending process. The system is designed to be as unobtrusive as possible for users. Figure 6.9 shows the BPMN diagram that represents the loan book process.

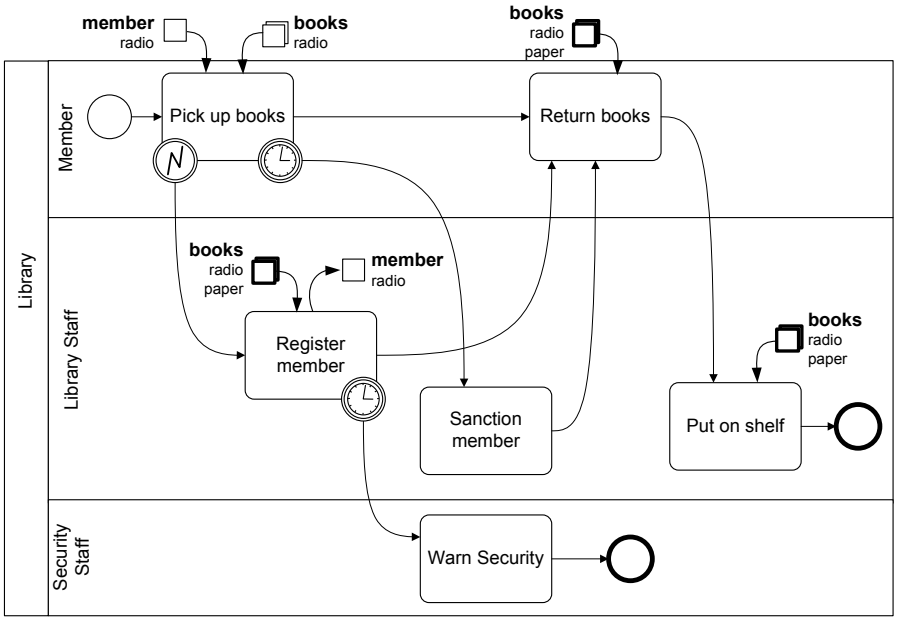

Fig. 6.9. Business process model for the Smart Library scenario

In a common scenario, a member approaches the shelf and takes the books she wants. When a book is taken, feedback is provided to the user to indicate the completion of the lending process. Later, when the member is finished with her books, she –or anybody else in her behalf– only has to place them in one of the return boxes –notice that books are used for correlation as it is illustrated using a thick border in the diagram. If the member is delayed too much in returning some books, a sanction is applied. Sanctioned members are not allowed to take books, so if a sanctioned member is detected picking books, security personnel are warned about the situation.

Library users that are not members are allowed to take books provided they register as members after taking them. These users should find a librarian in order to register as members in a short period of time. If this period becomes too long, security personnel are warned about the situation. Library personnel is equipped with a mobile device in order to make their work more flexible, allowing them to perform their activities anywhere in the library. Librarians are in charge of registering new members and placing the returned books in its place. The use of a mobile terminal allows to access information closer to the place the activity takes place.

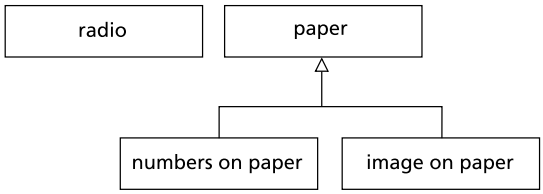

Fig. 6.10. Mediums defined for the Smart Library scenario

Physical Objects are identified in the present case study by means of different mediums. The defined mediums –see Fig. 6.10– are *radio* and *paper*. The *paper* medium is specialized in two mediums depending on the use of images or numbers to express the identifiers –see inheritance relationship in Fig. 6.10.

The *radio* medium offers wireless identification at different distances not requiring direct line of sight between labels and capturers. Since a great degree of automation is desired, the *radio* medium is considered the main medium for identification in this case study. However, the identification by means of *paper* is also considered as a backup for books. Since books are paper-based, the use of this medium is quite natural.

Data Structure

The information handled at the digital space in the Smart Library case study includes the definition of *Books*, *Members* and *Loans*. In addition to these main classes, information about books such as their *Author* or the *Shelf* assigned to them is considered. Loans are created as books are borrowed by a member.

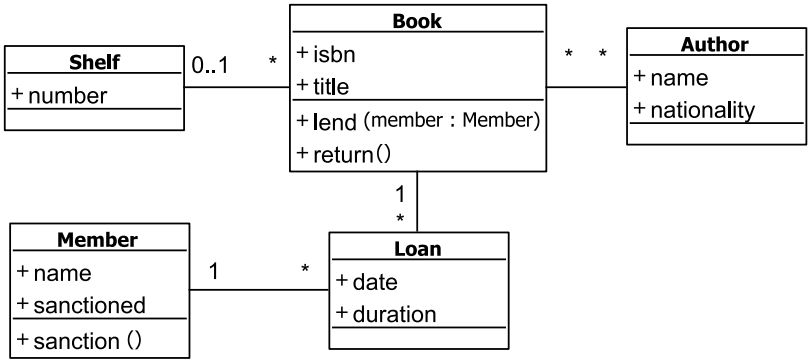

Fig. 6.11. Data model for the Smart Library scenario

The attributes considered for the different elements have been kept to a minimum since the purpose of this work was on identification aspects. In a real scenario more information about the books –and possibly other kinds of media– would be considered.

Technologies

Several identification technologies have been selected for this case study. These are *RFID*, *Fiducials* and *Text Label*. *RFID* provides identification capabilities supporting the *radio* medium. *Fiducials* and *Text label* provide identification capabilities by means of the *paper* medium. *Fiducials* technology uses images while the *Text label* uses numbers to represent identifiers. The considered technologies support different codifications. EPC is supported by *RFID* and *Text label*, while *Consecutive numbering* is supported by *Fiducials* and *Text label*.

In addition to the identification technologies, the *Beepers* technology is included to represent the different devices that can be used to provide feedback to the user. Devices of this kind can provide feedback by means of any of the different human senses.

Table 6.1 defines the possible resources that can be used for each technology in the different functions the technology offers. *RFID* technology requires an antenna for both minting and capturing identifiers. Image capturing devices, such as a video or photo camera, are required for *Fiducials*. While input devices are required for *Text label* capturers –a keyboard or a numeric keypad

Technology	Function	Resource Type
RFID	capturer, minter	RFID antenna
Fiducials	capturer	video camera, photo camera
	minter	printer
Text label	capturer	keyboard, keypad
	minter	printer, pen
Beepers	feedback	light bulb, speaker

Table 6.1. Technologies and resources for the Smart Library scenario

are considered for the example. Printing resources are required for the minting of paper-based technologies. For providing feedback, beepers would use either light or sound depending on the chosen resource.

Services

Different services are defined to cope with the requirements of the case study. Each library shelf is provided a *Shelf Detector* service. This service is in charge of detecting the movement of books in the shelf. It also detects the members that take books to register the corresponding loan. *RFID* is used as the identification technology to offer a good process fluency. A feedback mechanism is also incorporated in each shelf –by means of the Feedback service– to inform of member identification the need for identifying members.

A *Return Box* service is defined to allow an easy return of books. Different return boxes can be present in the library making no difference in which one members place the books in –this service does not provide contextual information since is is not attached to any element. The *Mobile Librarian* service is used for each librarian to offer assistance to members for their registration and the return of books. In addition to *RFID*, *Text label* technology is used by this service, so it is useful when there is some problem with the RFID infrastructure –e.g., a damaged tag.

Interaction

The interaction mechanisms considered for each task of the process in terms of identification are intended to offer a natural interaction to users. For the *pick up books* task, books are identified as they are *removed* from the shelf. The

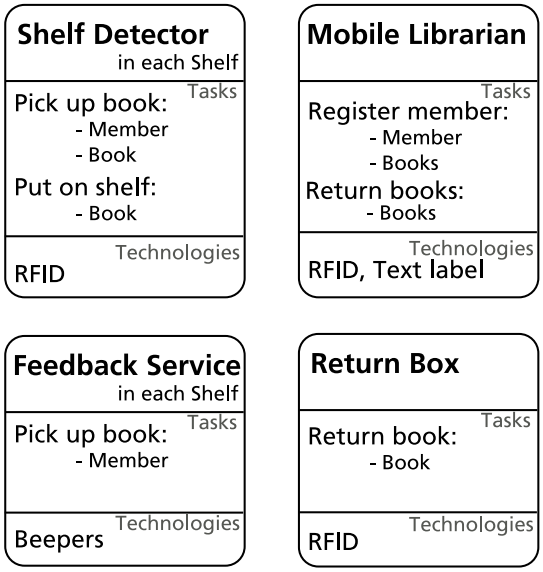

Fig. 6.12. Services model for the Smart Library scenario

total of books involved in this task is determined *implicitly* by the detection of a *Member*. This is, the books removed previously to the identification of a *Member* are the ones she is picking up. *Members* are identified following the *appear* pattern with a cardinality of one for the detection –i.e., the detection takes place when the member card is detected and later is not. This means that the member is detected when, after approaching the shelf and haven taken some books, she leaves.

The *register member* task takes place when a user that is not registered in the system has taken some books. These books are detected by parts – one by one or several at the same time–, so the *add* pattern is chosen as the interaction pattern. The completion in the detection of the taken books is determined explicitly by the librarian. In the case of *return books* and *put on shelf* tasks, detection also follows the *add* pattern, but the completion in the detection is determined by cardinality, so a one-by-one approach is followed.

For all the tasks involving *Physical Objects*, the error pattern chosen is the *not found*. So information about each identified element should be present in the system.

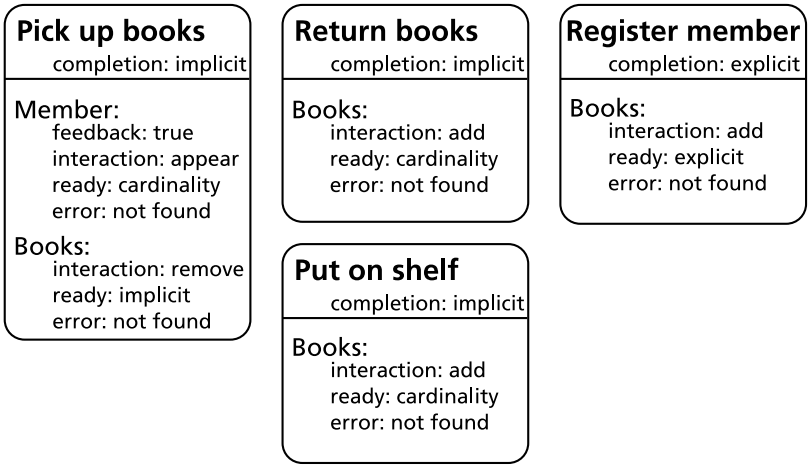

Fig. 6.13. Interaction model for the Smart Library case study

Policies

Although non-obtrusiveness is the main goal of this case study, making the user aware of certain system behavior is essential to avoid uncertainty. When a loan is performed, the user should be notified. In order to do so, feedback is provided in the detection of a member during the loan process.

As it is stated in Table 6.1, the *Beepers* technology –used for offering feedback– can use two different resource types, a light bulb that flashes or a a speaker that makes a beep. Considering the silent ambient of a library, the use of a light is preferred. However, some visually-impaired members require a feedback mechanism not visually demanding.

The first step to offer the system adaptation to this requirement is to define the properties of interest for the system. In this case the properties are *visual* and *acoustic* behavior. Then, resource types are qualified to indicate that the light bulb supports the *visual* behavior while the speaker supports the *acoustic* one.

Given this resource description, a general policy could be defined where an *acoustic* behavior is *discouraged*. With this policy, light is used as the default feedback. A policy for visually-impaired users can determine that the *visual* behavior should be *forbidden* for them. In this way, the system can adapt

to the particular needs of each user, using the speaker only when it is really needed.

6.2.2 System Development

A prototype of the system previously described has been developed. For the development of the Smart Library prototype, Apache Tuscany is used as the implementation of the SCA and SDO standards. The business process for the library loans is supported by Intalio BPMS, a business process execution engine. To support RFID identification the Accada middleware is used. The use of Accada allows for the development of RFID based solutions that follow EPCglobal standards. In this way, RFID devices can be simulated when the system is prototyped.

More details about the development of the Smart Library prototype is provided below.

Business Process

An executable definition of the business process is obtained for the loan process of the Smart Library. The current business process is an intra-organizational process since not external partners are involved. So the WS-BPEL definition only includes the *Library* and the *Task Manager* as partner links. The *Task Manager* is in charge of the activities that require interaction with the physical world –either because they require user participation or because physical elements are involved– introducing the need for asynchronous communication. The *Library* partner link represents the library Information System that coordinates the process.

The process flow for this process is linear for the common case. However, exception handling is required in some of the tasks. For example, the *pick up book* task considers three different exceptional situations. These exceptions occur when (1) a member is not detected, (2) a sanctioned member is detected and (3) a book is not returned in the expected time. To express this in the WS-BPEL definition, the *pick* activity is used. This activity waits for the occurrence of exactly one event from a set of events, then executes the activity associated with that event. So three events are defined, the two former ones are message based –using the *onMessage* event type to detect sanctioned or

non registered members– while the later is timer based –using the *onTimer* event type.

For the rest of tasks the defined WS-BPEL follows the invoke-receive schema described in the guidelines of Chapter 5 –already applied for the Smart Toolbox case study.

Book identifiers are used for the process correlation, since a particular book can only be involved in just one loan process instance at a time. In this way, given a book, the loan process in which it is involved is clearly defined. To offer more flexibility in the process, each book initiates an individual instance of the process. So, users are not forced to return all their books at the same time.

System Components

For each of the tasks requiring identification of physical elements, a *Task Processing Component* is defined. These components make use of different *Identification Components* depending on the *Physical Objects* they access.

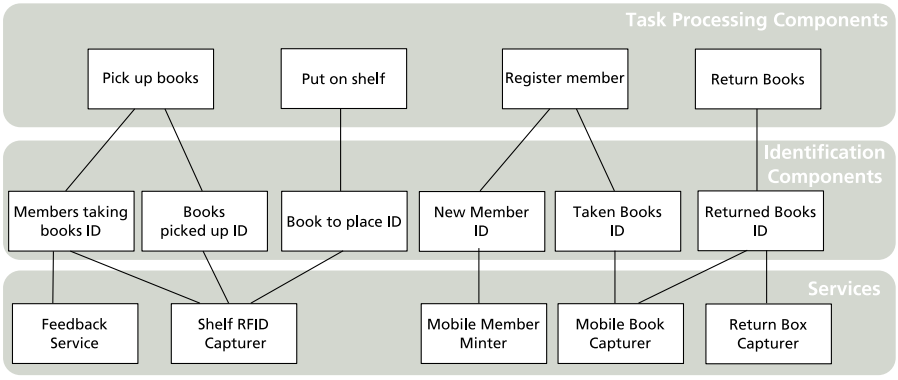

Fig. 6.14. Architectonic components for the Smart Library scenario

Some *Capturers* are shared among different Identification Components. This is possible when a service is defined in the specification including several tasks with some physical elements that can be identified with the same technology. For example, the *pick up books* and *put on shelf* tasks are supported by the *Shelf Detector Service* from the specification –see Fig. 6.12. So the

identification of members and books for these tasks is done using the same capturer –since RFID is used in all the cases.

RFID Capturers implementation is a wrapper around the Accada API that implements the EPCglobal Reader Protocol. Identifiers are transmitted using HTTP and structured by means of XML. The XML element *tagID* contains the identifier while the context of identification is expressed by different elements. The *tagEvent* indicates information such as the event type –e.g., observed or lost– or the time in which the event was produced. The *sourceInfo* indicates which antenna performed the detection, which is useful to distinguish between the detections associated to the different library shelves in the case study.

Data Structure

The structure of the data handled by the system was defined in a XML schema. Each of the classes present in the data model –see Fig. 6.11– is defined as a *Type* in the schema –defining its attributes and relationships. For each physical element involved in the business process an element is defined in the schema. In addition, for each physical element, a *Data Provider* component is created. The implementation of Data Providers relies in SDO to access a XML file where all the information about the library is stored.

Interaction

To interact with the system, the RFID simulator provided by Accada was used. With the simulator different antennas and tags can be defined, allowing for the triggering of events in an easy way. A screenshot of the simulator for the present case study is depicted in Fig. 6.15. The detection events produced using the simulator are managed by the Identification System in order to provide the adequate interaction according to the patterns defined in the system specification.

For the implementation of graphical interfaces for the different tasks that require user intervention, Eclipse Forms[2] were used. Eclipse Forms are a Java solution based on the Standard Widget Toolkit[3] (SWT) that offers facilities

[2] http://www.eclipse.org/articles/Article-Forms/article.html
[3] http://www.eclipse.org/swt/

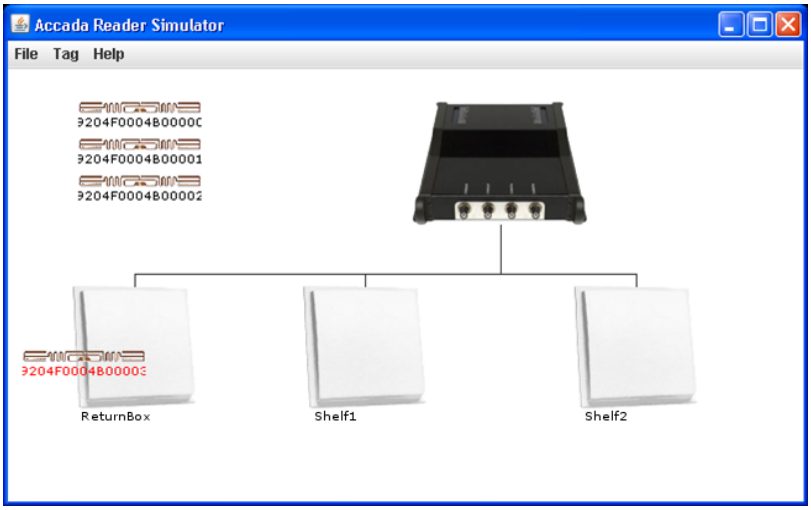

Fig. 6.15. The RFID simulator used for the Smart Library case study

for creating web-like forms in the Java platform. The use of Eclipse Forms results in responsible and clean user interfaces. Its message handling mechanisms resulted quite adequate to notify the user when the specified error conditions occur.

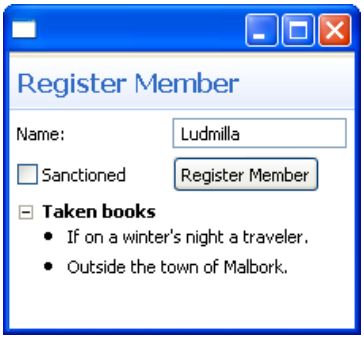

Fig. 6.16. Interface defined for the *register member* task

Figure 6.16 shows the interface defined for the Register member task. In the upper part the data required for completing the task –name and sanctioned status of the new member– is collected. Bellow a list of the books

taken by this person is shown. Since task completion is explicit for this task, the interface includes a button to trigger its completion.

6.3 Conclusions

The application of the development method in the presented case studies has provided valuable feedback at different levels. After this process the expressivity for capturing identification requirements provided by the DSL and the infrastructure support from the defined architecture are considered adequate for the application domain.

The resulting applications from these case studies are interesting by themselves. Although some aspects have been simplified for the development of prototypes, the technologies in use are production ready. In addition, the design effort to improve both case studies has lead to better enhancing the user experience and the business process fluency.

Considering that the case studies were originated in the two different areas that this work comprises, a notion of how applications of each area can be extended to cover the remaining aspects is also provided.

7

Conclusions

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 7.1 presents the main contributions to both the Business Process Management and the Internet of Things communities. Section 7.2 provides an overview of the publications that have emerged from this work. Finally, Section 7.3 outline the ongoing and future work that can extend this research line.

7.1 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. So, 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 the Internet of Things. The method comprises from specification to the final implementation. To promote separation of concerns, different development roles are defined for the method.

7.2 Publications

The present work connects two different research areas such as Business Process Management and the Internet of Things from a modeling perspective. A total of **14 research publications** of different nature have been elaborated in the context of this work. Table 7.1 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, **two senior theses** were co-directed in the context of this work to explore some concepts and related technologies. A more detailed description of the different contributions in these works is exposed below. Publications are organized according to their relation with the Business Process Management area or the Internet of Things area.

7.2.1 Business Process Related Publications

The first phase of the work consisted on an exploration of Business Process modeling area and model transformation techniques. The interest was to

Publication	Author Position	Type	Published at
(Giner & Pelechano, 2008)	1st	International Conference	AmI 2008
(Giner et al., 2008a)	1st	International Conference	ICEIS 2008
(Giner et al., 2008b)	1st	International Conference	UCAmI 2008
(Giner et al., 2008c)	1st	Workshop	DSDM 2008
(Giner et al., 2007b)	1st	Conference	JISBD 2007
(Giner & Torres, 2007)	1st	International Workshop	IDEAS 2007
(Giner et al., 2007c)	1st	Demo Session	JISBD 2007
(Giner et al., 2007a)	1st	International Workshop	MDWE 2007
(Cetina et al., 2008)	2nd	International Workshop	Models@Run.time 2008
(Torres et al., 2007a)	2nd	International Workshop	CAiSE Forum 2007
(Torres et al., 2007b)	2nd	Workshop	PNIS 2007
(Torres et al., 2007d)	3rd	International Journal	IEEE Latin America Transactions
(Torres et al., 2007c)	3rd	International Journal	ERCIM News
(Torres et al., 2006)	3rd	Conference	JISBD 2006

Table 7.1. Summary of Publications

explore the expressivity of BPMN and the use of model transformation techniques to derivate SOA-based applications. These concepts were applied in a Web Engineering method developed by Victoria Torres constituting a common point of research that resulted in four publications (Torres et al., 2007d,b,c, 2006). Essential 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.

A more direct contribution to the present work is the definition of model transformations that bridge the gap between BPMN and WS-BPEL. In the present work, this is used to automate part of the development process. The result of this is the publication of a paper with the definition of this model transformation (Giner et al., 2007a) and a demo session showing the defined tool support (Giner et al., 2007c).

A Senior Thesis in this area titled “Generación automática de aplicaciones web de soporte a procesos de negocio” (II-B-DSIC-379/06) was co-directed.

In this work, some of the architectonic concepts required for the support of business processes were explored.

7.2.2 Internet of Things Related Publications

Once confidence in business process modeling notations and execution support was obtained, the challenge was the application of these concepts to reach the physical world. This was faced considering (1) the architecture, (2) system specification and (3) development method for the kind of applications this work is dealing with.

The architecture presented in the present work was published in the European Conference on Ambient Intelligence (AmI'08) (Giner & Pelechano, 2008). That work included the description of the architectural process followed, the definition of the architecture at conceptual level and its mapping to a SCA-based solution. The reconfiguration aspects of the architecture and the policy mechanism incorporated to the architecture has been published in the 3rd Symposium of Ubiquitous Computing and Ambient Intelligence (UCAmI'08) (Giner et al., 2008b), and the strategy for adaptation has been published in the Models@run.time Workshop (Cetina et al., 2008) from the Models 2008 conference.

Regarding system specification, a conceptual framework was defined to detect the different concepts involved in the identification process (Giner et al., 2008a). The description of some interaction aspects (Torres et al., 2007a) and the different primitives that conform the DSL (Giner et al., 2008c) presented were also of interest for the modeling community.

Finally, the study of the requirements for a model-based development process (Giner & Torres, 2007) and the definition of a method for integrating Ubicomp services with business processes (Giner et al., 2007b) has lead to the development process presented in this work.

In addition, a Senior Thesis titled "Diseño e implementación de una arquitectura para orquestar servicios en sistemas pervasivos" (II-B-DSIC-33/07) was co-directed. In this work, some of the architectonic concepts required for the integration of AmI systems and business processes were explored.

7.3 Future Work

The use of modeling techniques to formalize concepts allows for the automation of software development. In the present work, only the generation of glue code is obtained automatically. The next big step is **automating the development process**. In order to do so, the DSL primitives that capture the requirements for the system should be proven expressive enough and model transformations should be defined to formalize the mapping between the different primitives.

In order to evaluate the expressivity of the defined DSL, **the development of more case studies** in production environments is required. The feedback of development teams and final users would be really valuable. This feedback could also help to find a concrete syntax for the primitives defined in the language.

The **development of tools** to support the process and the **integration with existing modeling solutions** in the area such as PervML would provide a great value to the work. The integration with PervML would allow the definition of some aspects –such as service behavior– that are not related with identification but are needed for the construction of a complete system.

References

Aarts, E., Harwig, R., & Schuurmans, M. (2002). Ambient intelligence. *The invisible future: the seamless integration of technology into everyday life*, (pp. 235–250).

Aberer, K., Hauswirth, M., & Salehi, A. (2006). Middleware support for the "internet of things". In *5th GI/ITG KuVS Fachgespräch*. Stuttgart, Germany.

Alves, A., Arkin, A., Askary, S., Barreto, C., Bloch, B., Curbera, F., Ford, M., Goland, Y., Guzar, A., Kartha, N., Liu, C. K., Khalaf, R., Knig, 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.

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). Integrat-

ing 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.

Brooke, T., & Burrell, J. (2003). From ethnography to design in a vineyard. In *DUX '03: Proceedings of the 2003 conference on Designing for user experiences*, (pp. 1–4). New York, NY, USA: ACM.

Cetina, C., Giner, P., Fons, J., & Pelechano, V. (2008). A model-driven approach for developing self-adaptive pervasive systems. Models@run.time Workshop. Springer LNCS.

Deursen, A., & Klint, P. (1997). Little languages: little maintenance? Tech. rep., Amsterdam, The Netherlands.

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.

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 meta-pyramids: Languages vs. metamodels - episode ii: Story of thotus the baboon. In J. Bzivin, & R. Heckel (Eds.) *Language Engineering for Model-Driven Software Development*, vol. 04101 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum fr 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., Lampe, M., & Schoch, T. (2003). The smart box concept for ubiquitous computing environments.

Floerkemeier, C., Roduner, C., & Lampe, M. (2007). Rfid application development with the accada middleware platform. *IEEE*

Systems Journal, Special Issue on RFID Technology.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns*. Addison-Wesley Professional.

Gershenfeld, N., Krikorian, R., & Cohen, D. (2004). The internet of things. *Scientific American*, *291*(4), 46–51.

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. 3rd Symposium of Ubiquitous Computing and Ambient Intelligence (UCAmI'08). Springer Advances in Soft Computing.

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 *European conference on Ambient Intelligence (AmI 08)*. 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 Ingeniera de Requisitos y Ambientes 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).

Greenfield, A. (2006). *Everyware: The Dawning Age of Ubiquitous Computing*. Berkeley, CA: New Riders Publishing.

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 applications: 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).

Jamali, B., Sharp, E., Thorne, A., & Cole, P. (2007). Technology selection for identification applications. Tech. rep., Auto-ID Labs.

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.

Keays, R., & Rakotonirainy, A. (2003). Context-oriented programming. In *MobiDe '03: Proceedings of the 3rd ACM international workshop on Data engineering for wireless and mobile access*, (pp. 9–16). New York, NY, USA: ACM.

Kent, S. (2002). Model driven engineering. In *IFM '02: Proceedings of the Third International Conference on Integrated Formal Methods*, (pp. 286–298). London, UK: Springer-Verlag.

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.

Krogstie, J., Opdahl, A. L., & Brinkkemper, S. (2007). *Conceptual Modelling in Information Systems Engineering*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.

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.

Lewis, D. (1968). Counterpart theory and quantified modal logic. *Journal of Philosophy*, *65*, 113–26.

Mathes, A. (2004). Folksonomies - cooperative classification and communication through shared metadata.

Mayer, R. J., Painter, M. K., & DeWitte, P. (1992). Idef family of methods for concurrent engineering and business re-engineering 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).

Muñoz, J., & Pelechano, V. (2005). Building a software factory for pervasive systems development. In *CAiSE*, (pp. 342–356).

OMG (2005). OCL 2.0 Specification. ptc/2005-06-06.

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).

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.*.

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.

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.

Schmitt, C., Fischbach, K., & Schoder, D. (2006). Towards ambient business: Value-added services through an open object information infrastructure. In *Proceedings of the CollECTeR Europe*, (pp. 141 – 148).

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 ubiquitous computing on business processes. In F. Mattern, & M. Naghshineh (Eds.) *Short Paper Proc. International Conference on Pervasive Computing*, (pp. 62–74). Pervasive2002.

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., 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.

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.

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., 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.

Weiser, M. (1994). The world is not a desktop. *interactions*, *1*(1), 7–8.