DISS. ETH NO. 18787

Application-level System and Tool Support for Auto-ID Application Development

A dissertation submitted to the
ETH ZURICH

for the degree of
Doctor of Sciences

presented by
Matthias Lampe
Master of Science Computer Science, Portland State University
born June 10, 1971
citizen of Germany

accepted on the recommendation of
Prof. Dr. Friedemann Mattern, examiner
Prof. Dr. Elgar Fleisch, co-examiner

2010

Abstract

Radio Frequency Identification (RFID) systems have begun to find greater use in the consumer object identification market, in industrial automation, and in mobile asset and supply chain management, in a wide range of industries such as retail, pharmaceutical, and defense. The term automatic identification (Auto-ID) applications is used to generalize RFID applications and includes other Auto-ID technologies such as barcode or Bluetooth. More complex Auto-ID applications also use sensor technologies not only to identify objects but also to track their state (e.g. temperature sensors to ensure quality of perishable goods).

Bridging the gap between the physical world of objects (e.g., products and logistical units) and the digital world of IT systems to improve existing business processes is the main driver of the widespread RFID adoption. However, even if the gap between the physical and the digital world is bridged, there is still a gap between applications or enterprise resource planning (ERP) systems on one side and Auto-ID systems on the other. Application developers are faced with several challenges: Instead of concentrating on the application or business logic they have to deal with Auto-ID specific details such as readers, tags, or specific error sources. They also have to duplicate code when developing a new application for capturing Auto-ID data from readers, for filtering and aggregating Auto-ID data to increase their accuracy, for providing persistent storage and querying capabilities, and for reacting to business events such as the arrival of a shipment.

To facilitate the development of Auto-ID applications, this thesis provides concepts, programming models, building blocks and tools that abstract from Auto-ID specific details and provide the necessary services and the appropriate level of reuse. In particular, the contributions are:

- an Auto-ID Object Model that abstracts from low-level Auto-ID and sensor concepts which provides the means for an application to model and represent its domain as the base for the application logic. The model focuses on the domain of Auto-ID applications, that is,

applications whose application logic is based on an implicit or explicit model of the physical world and is triggered by (near) real time observations of the physical world through Auto-ID readers and sensors.

- a state machine-based programming model that allows defining Auto-ID related micro business processes. Typically Auto-ID applications are not interested in all the dynamic changes of the observed physical objects. The business process definitions allow an Auto-ID application to integrate application logic into an Auto-ID infrastructure in order to only report exceptionally states related to the objects, their properties and relationships.
- a visual tool-based approach to instantiate, configure and manage an Auto-ID infrastructure. Typically an Auto-ID infrastructure consists of many different hardware and software components that have to be customized to fit the need of a certain application domain. Such an instantiation and configuration is a tedious task and requires programming skills in addition to Auto-ID and application domain knowledge. The visual tool-based approach provides a concrete representation of the application domain and supports non-software developers in the different tasks over the lifecycle of an Auto-ID infrastructure.

The evaluation, based on a prototypical implementation of an Auto-ID Infrastructure called the Object Monitoring System (OMS) and on several representative case-studies, shows that the contributions of the thesis provide the right level of abstractions and services to facilitate the development of Auto-ID applications in several different application domains. The Auto-ID Object Model provides for an adequate representation of the application domains. The programming model allows defining a variety of Auto-ID related business processes in the application domains supporting fast processing of Auto-ID data in a business workflow. The visual tool-based approach proves to facilitate the instantiation, configuration and deployment of the Auto-ID infrastructures in the application domains especially for non-software developers.

All in all, this thesis provides sound concepts and its contributions help to facilitate the development of Auto-ID applications, help to decrease development costs and, in general, contribute to the dissemination of Auto-ID technologies in industry and other areas of application.

Kurzfassung

Radio Frequency Identification (RFID) Systeme finden eine immer grössere Verbreitung bei der Konsumgüter-Identifikation, in der industriellen Automatisierung, im Mobile Asset und Supply Chain Management sowie in weiteren Bereichen wie zum Beispiel Einzelhandel, Pharma und Verteidigung. Der Begriff „automatische Identifikationsanwendungen“ (Auto-ID-Anwendungen) verallgemeinert RFID-Anwendungen und beinhaltet andere Auto-ID-Technologien wie Barcode oder Bluetooth. Komplexere Auto-ID-Anwendungen benutzen ausserdem Sensor-Technologien zum Überwachen der Objektzustände (z.B. Temperatursensoren, um die Qualität von verderblichen Gütern sicherzustellen).

Der Haupttreiber bei der Einführung von RFID ist die Überbrückung der Lücke zwischen der physischen Welt der Objekte, d.h. der Produkte und logistischen Einheiten, und der digitalen Welt der Informationssysteme zur Optimierung der vorhandenen Geschäftsprozesse. Auch wenn die Lücke zwischen der physikalischen und der digitalen Welt überbrückt wird, gibt es jedoch weiterhin eine Lücke zwischen den Anwendungen oder Enterprise-Resource-Planning-Systemen (EPR-Systemen) auf der einen Seite und den Auto-ID-Systemen auf der anderen.

Anwendungsentwickler sind mit mehreren Herausforderungen konfrontiert: Anstatt sich rein auf die Anwendung und Geschäftslogik konzentrieren zu können, müssen sie sich mit Auto-ID-spezifischen Details wie Lesegeräten, Transpondern oder besonderen Fehlerquellen beschäftigen. Ausserdem kommt es zur Duplizierung von Code bei der Entwicklung einer neuen Anwendung zur Erfassung von Auto-ID-Daten von Lesegeräten, zum Filtern und Aggregieren von Auto-ID-Daten, um deren Qualität zu verbessern, zur Bereitstellung von persistentem Speicher und Abfragefunktionalität und zur Reaktion auf geschäftsrelevante Ereignisse wie z.B. der Zustellung einer erwarteten Lieferung.

Um die Entwicklung von Auto-ID-Anwendungen zu erleichtern und zu unterstützen, liefert die vorliegende Arbeit Programmiermodelle, Bausteine und Werkzeuge, die von Auto-ID-spezifischen Details abstrahieren,

und die die erforderlichen Dienste und die geeigneten Stufen der Wiederverwendung anbieten. Im Einzelnen sind die Beiträge dieser Arbeit:

- Ein Auto-ID-Objektmodell, das von systemnahen Auto-ID- und Sensor-Konzepten abstrahiert und für eine Anwendung die Möglichkeit bietet, ihre Domäne als Basis für die Anwendungslogik zu modellieren und darzustellen. Der Fokus liegt auf der Klasse der Auto-ID-Anwendungen, d.h. auf Anwendungen, deren Anwendungslogik auf einem impliziten oder expliziten Modell der physischen Welt basiert und von (Beinahe)-Echtzeit-Beobachtungen der physischen Welt durch Auto-ID-Lesegeräte gesteuert wird.
- Ein auf Zustandsautomaten basierendes Programmiermodell, das die Definition von Mikro-Geschäftsprozessen erlaubt. Auto-ID-Anwendungen sind normalerweise nicht an allen dynamischen Änderungen der zu beobachteten physischen Objekte interessiert. Die Definition von Geschäftsprozessen gibt einer Auto-ID-Anwendung die Möglichkeit, Anwendungslogik direkt in eine Auto-ID-Infrastruktur zu integrieren. Dadurch können Berichte an die Anwendung auf besondere Zustände der Objekte, deren Eigenschaften und Beziehungen eingeschränkt werden.
- Eine auf visuellen Werkzeugen basierte Vorgehensweise, um eine Auto-ID-Infrastruktur zu instanziieren, zu konfigurieren und zu verwalten. Eine Auto-ID-Infrastruktur besteht aus vielen verschiedenen Hardware- und Softwarekomponenten, die an die speziellen Erfordernisse einer Anwendungsdomäne angepasst werden müssen. Solch eine eine Instanziierung und Anpassung ist eine schwierige Aufgabe und erfordert neben dem Auto-ID-Wissen und der Kenntniss über die Anwendungsdomäne Softwareentwicklungskenntnisse. Die auf visuellen Werkzeugen basierte Vorgehensweise bietet eine konkrete Repräsentation der Anwendungsdomäne im Sinne der Endanwender-Programmierung an. Sie unterstützt damit Anwender ohne Softwareentwicklungskenntnisse bei den verschiedenen Aufgaben, die im Laufe des Lebenszyklus einer Auto-ID-Infrastruktur anfallen.

Eine Evaluation, basierend auf einer prototypischen Implementierung einer Auto-ID-Infrastruktur, dem Object Monitoring System (OMS) und mehreren repräsentativen Fallstudien, zeigt, dass die Beiträge dieser Arbeit eine geeignete Stufe der Abstraktion zur Unterstützung und Vereinfachung der Entwicklung von Auto-ID-Anwendungen bieten. Das Auto-ID-

Objektmodell erlaubt die adäquate Repräsentation der verschiedenen Anwendungsdomänen. Das Programmiermodell ermöglicht die Definition einer Vielzahl von Geschäftsprozessen in den verschiedenen Anwendungsdomänen und unterstützt damit die übergreifenden Geschäftsabläufe. Die auf visuellen Werkzeugen basierte Vorgehensweise kann die Instanziierung, Anpassung und Verwaltung der Auto-ID-Infrastrukturen in den Anwendungsdomänen deutlich erleichtern.

Zusammengenommen sollten die im Rahmen der Dissertation erarbeiteten und evaluierten Ansätze helfen, die Entwicklung von Auto-ID-Anwendungen weiter zu vereinfachen, Entwicklungskosten einzusparen und damit einen Beitrag zur Verbreitung von Auto-ID-Technologien in vielfältigen Anwendungsbereichen zu leisten.

Table of Contents

Abstract 3
Kurzfassung 5
Table of Contents 9
Table of Figures 13
Table of Tables 17
Abbreviations 19
1. Introduction 21
1.1. Motivation 21
1.2. Contributions of the Thesis 22
1.2.1. Auto-ID Object Model 22
1.2.2. Visual Tool Approach 23
1.2.3. Methodology 24
1.3. Thesis Outline 24
2. Auto-ID Technologies 27
2.1. Barcode Technology 28
2.2. RFID Technology 30
2.2.1. RFID System Components and Operating Principles 30
2.2.2. RFID Standards 34
2.3. Other Wireless Auto-ID Technologies 36
2.4. Sensor Technology 37
3. Auto-ID System Requirements 39
(R1) Object Representation 41
(R2) Object Persistency 42
(R3) Object Relationships 42
(R4) Location Information 42
(R5) Object Identification 43
(R6) Object Identifier 43
(R7) Object Data Enrichment 43
(R8) Physical World Interaction 44
(R9) Business Context Enrichment 44
(R10) Data Dissemination 45

(R11) Auto-ID Data Aggregation 45
(R12) Auto-ID Data Filtering 46
(R13) Fault and Configuration Management 46
(R14) Tag Identifier Management 46
(R15) Tag User Memory 46
(R16) Sensor Support 47
(R17) Actuator Support 47
(R18) External Reader Triggers 47
(R19) Loose Coupling of Components 47
(R20) Configurability of System 48
(R21) Privacy 48
(R22) Other Application Requirements 48
4. Auto-ID Infrastructure Concepts and Implementation 49
4.1. Filtering and Aggregation of Observations 51
4.2. Overview of Location Models 53
4.2.1. Geometric Location Models 53
4.2.2. Symbolic Location Models 54
4.2.3. Hybrid Location Models 54
4.2.4. Semantic Location Models 54
4.2.5. Summary and Discussion 54
4.3. Auto-ID Object Model 55
4.3.1. Physical Object Representation 57
4.3.2. Properties of Objects 62
4.3.3. Functions 65
4.3.4. Object History and Queries 67
4.3.5. Business Process Support 69
4.4. Auto-ID Infrastructure Implementation 80
4.4.1. Object Monitoring System 82
4.4.2. Alternative Implementation Approaches for the Auto-ID Object Model History 90
4.5. Related Work and Discussion 93
4.5.1. EPC Network 95
4.5.2. Auto-ID Middlewares/Infrastructures 101
4.5.3. Ubiquitous Computing Infrastructures 107
5. Visual and Generative Tool-based Auto-ID System Development Process 111
5.1. Generative and Visual Programming Concepts 113
5.1.1. Generative Programming 113

5.1.2. Visual Programming 116
5.2. Auto-ID System Development Process 120
5.2.1. Visual Instantiation Tool 124
5.2.2. Generation Tool 129
5.2.3. Deployment Definition and Deployment Tool 130
5.3. Related Work 132
6. Case Studies 137
6.1. Smart Medicine Shelf 137
6.2. Tool Management in Aircraft Maintenance 143
6.3. Augmented Knight's Castle 150
6.4. Discussion 158
7. Conclusion 163
7.1. Auto-ID Object Model and Business Process Support 164
7.1.1. Contribution 164
7.1.2. Limitations and Future Work 166
7.2. Auto-ID Application Development Process 168
7.2.1. Contribution 168
7.2.2. Limitations and Future Work 169
8. Appendices 171
8.1. Formal Definitions of the Auto-ID Object Model Set Definition Language in EBNF 171
8.2. Formal Definitions of the Auto-ID Object Model Business Process Condition Definition Language in EBNF 172
8.3. Example Business Process Definition in XML 173
8.4. OMS Database Implementation Details 174
8.5. XML Schemes of the VIT / Generator 174
Bibliography 185

Table of Figures

Figure 2-1 Bridging the media gap between the physical and digital world (based on [62]) 27
Figure 2-2 Barcode types of the EAN.UCC systems 29
Figure 2-3 ISO/IEC standardized 2-dimensionale barcodes 30
Figure 2-4 Operation principles and components of an RFID system 31
Figure 2-5 Different types of RFID transponders and readers 33
Figure 2-6 BTnode (a) and Tmote (b) sensor nodes (from [3] and [15]) . 36
Figure 3-1 Auto-ID usage in a medical supply chain scenario (source: EPCglobal) 39
Figure 3-2 Auto-ID readers in a distribution center feeding captured data to different applications (source: EPCglobal) 41
Figure 3-3 Overview of Auto-ID infrastructure requirements including their dependencies 44
Figure 4-1 Layers of Auto-ID infrastructure 50
Figure 4-2 Example for filtering and aggregation of observations 53
Figure 4-3 Auto-ID Object Model instantiation 56
Figure 4-4 Overview of Auto-ID Object Model 57
Figure 4-5 Auto-ID Object Model entities representing physical objects in the real world 58
Figure 4-6 Example instance of the Auto-ID Object Model (UML object diagram representation) 59
Figure 4-7 Example instance of the Auto-ID Object Model (simplified floor and tree representation) 60
Figure 4-8 Auto-ID Object Model entities representing physical objects in the real world (emphasis on entity Property) 64
Figure 4-9 Example model instance with emphasis on Property and Function 65
Figure 4-10 Auto-ID Object Model entities representing physical objects in the real world (emphasis on entity Function) 66
Figure 4-11 History of located object MineralWater1L.130 67
Figure 4-12 History of the location CooledStorage 68

Figure 4-13 Auto-ID Model with emphasis on Business Process Support 70
Figure 4-14 State machine to monitor the incoming shipment #1020 73
Figure 4-15 Auto-ID Infrastructure deployment for retail store example 80
Figure 4-16 OMS Architecture Overview 81
Figure 4-17 SQL base query for `LocationsOfObject` query 87
Figure 4-18 SQL base query for `LocatedObjects` query 87
Figure 4-19 EPCglobal Architecture Framework compared to OMS 96
Figure 4-20 Comparison of different Auto-ID middleware/ infrastructure approaches 103
Figure 5-1 Tool support for the Auto-ID infrastructure layers 112
Figure 5-2 Generative domain model (from [40]) 114
Figure 5-3 Software development based on Domain Engineering (from [41]) 115
Figure 5-4 Fregean versus analogical representation 117
Figure 5-5 Auto-ID Application Development Process 123
Figure 5-6 GUI of the Visual Instantiation Tool 126
Figure 5-7 Set Management dialog 128
Figure 5-8 Conststructing a set comparison expression 129
Figure 5-9 GUI of the Deployment Definition Tool 131
Figure 5-10 Example deployment of Deployment Tool 132
Figure 5-11 OBS Bean Bilder (from [35]) 134
Figure 5-12 . LEGO Mindstorms Programming Environment 135
Figure 6-1 Setup of the Smart Medicine Shelf Application 138
Figure 6-2 Auto-ID Object Model instance for Smart Medicine Shelf .. 139
Figure 6-3 Instantiating the Smart Medicine Shelf with the VIT 140
Figure 6-4 Business processes of the Smart Medical Shelf 141
Figure 6-5 Smart Medical Shelf deployment 142
Figure 6-6 Setup of the Smart Toolbox prototype (left) and the Smart Tool Inventory applications (right) 144
Figure 6-7 Auto-ID Object Model instance for the tool management in aircraft maintenance application 145
Figure 6-8 Instantiating the tool management in aircraft maintenance application with the VIT 146
Figure 6-9 Toolbox business processes Base Check (left) and Routine Check (right) of the tool management in aircraft maintenance application 148

Figure 6-10 Tool inventory business processes of the tool management in aircraft maintenance application 149
Figure 6-11 tool management in aircraft maintenance application deployment 150
Figure 6-12 Overall setup of the Augmented Knight's Castle 151
Figure 6-13 Auto-ID Object Model instance of the Augmented Knight's Castle 152
Figure 6-14 Instantiating the Augmented Knight's Castle with the VIT 153
Figure 6-15 Business processes of the Augmented Knight's Castle 156
Figure 6-16 Antennas embedded in the playset (left) and RFID transponders to tag toy pieces (right) 157
Figure 6-17 Augmented Knight's Castle deployment 158
Figure 8-1 SQL statements to create the object and property history 174

Table of Tables

Table 4-1 Filter types 51
Table 4-2 Aggregate types 52
Table 4-3 Formal state transition conditions of example process 78
Table 4-4 Overview of definition language for state transition condition 79
Table 4-5 Comparing OODBMS and ORDBMS 90
Table 4-6 Available time-relational DB layers 92
Table 4-7 Layers, data models and data processing of an Auto-ID infrastructure 94
Table 5-1 Configuration and instantiation tasks for the Auto-ID infrastructure software components 121
Table 5-2 Configuration and instantiation tasks supported by VIT 125
Table 5-3 Configuration and instantiation tasks supported by the Deployment Definition Tool 130
Table 6-1 Business processes of the Smart Medical Shelf 140
Table 6-2 Business processes of the tool management in aircraft maintenance application 147
Table 6-3 Business processes of the Augmented Knight's Castle 154
Table 6-4 Auto-ID system requirements implemented by the presented approach 159

Abbreviations

ALE	Application Level Events
Auto-ID	Automatic Identification
EAS	Electronic Article Surveillance
EAN	European Article Association
EBNF	Extended Backus Naur Form
ECA	Event Condition Action
EPC	Electronic Product Code
EPCIS	EPC Information Services
ERP	Enterprise Resource Planning
GUI	Graphical User Interface
HAL	Hardware Abstraction Layer
HTML	Hyper Text Markup Language
HTTP	Hypertext Transfer Protocol
ID	Identification
ISO	International Standardization Organization
JAR	Java Archive
LLRP	Low Level Reader Protocol
LUS	Look-Up Service
MRO	Maintenance, Repair and Overhaul
OMS	Object Monitoring System
OODBMS	Object-Oriented Database Management System
ORDBMS	Object-Relational Database Management System
PML	Product Markup Language
POS	Point-of-sale
RDBMS	Relational Database Management System
RFID	Radio Frequency Identification
SCM	Supply Chain Management
SNMP	Simple Network Management Protocol
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
TCP/IP	Transmission Control Protocol/Internet Protocol

TDB	Temporal Database
Ubicomp	Ubiquitous Computing
UCC	Uniform Code Council
UPC	Uniform Product Code
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
XML	eXtensible Markup Language

1. Introduction

1.1. Motivation

Radio Frequency Identification (RFID) systems have recently begun to find greater use in the consumer object identification market, in industrial automation, and in mobile asset and supply chain management, in a wide range of industries such as retail, pharmaceutical, and defense. The use of RFID systems in these application domains has also been promoted by efforts of the Auto-ID Center to develop low cost RFID tags as an economical replacement of barcodes [133]. Traditionally RFID has been used among others in electronic article surveillance (EAS), livestock tracking, ski ticketing, toll collection, and car immobilizers [154].

Bridging the gap between the physical world of objects (i.e. products and logistical units) and the digital world of IT systems to improve existing business processes [63] is the main driver of the widespread RFID adoption. Automated data gathering through RFID eliminates manual labor and avoids the discontinuity between physical processes and the associated information processing. This results in more accurate and detailed data, higher process efficiencies, increased product quality, and cost savings through faster and better information processing.

Applications using RFID are often labeled automatic identification (Auto-ID) applications. This more general term also includes identification technologies such as traditional barcode and other technologies which can be used to automatically identify objects (e.g. Bluetooth or Wireless LAN). More complex applications also use sensor technologies not only to identify objects but also to track their state (e.g. temperature sensors used to ensure the quality of perishable goods).

However, even if the physical and the digital world get closer together, there is still a gap between the applications or enterprise resource planning (ERP) systems on one side and the RFID systems on the other. Application developers are faced with several challenges: Instead of concentrating on the application or business logic they have to deal with RFID

specific details such as readers, tags, or RFID error sources. They also have to duplicate code when developing a new application for capturing the RFID data from the readers, for filtering and aggregating RFID data to increase their accuracy, for providing persistent storage and querying capabilities, and for reacting to business events such as the arrival of a shipment.

It is therefore important to develop appropriate programming models and building blocks (e.g. middleware or frameworks) that abstract from RFID specific details and provide the necessary services and the appropriate level of reuse to facilitate the development of Auto-ID applications.

1.2. Contributions of the Thesis

In this section, we outline the two main contributions of the thesis to address the needs to support the software building process of Auto-ID applications on different levels: First, an Auto-ID Object Model and the Object Monitoring System (OMS), a component implementing the model, and second, a visual tool-based approach to the instantiation, configuration and management of an Auto-ID infrastructure. The approach that was taken during the course of the thesis is also presented in this section.

1.2.1. Auto-ID Object Model

The widespread adoption of RFID and other Auto-ID technologies requires not only low cost tags and readers, but also software components that, on the one side, manage readers, filter and aggregate captured RFID data, and on the other side, combine and enrich the RFID data with application logic, and generate appropriate business events, by providing a consistent object model to the application.

In this thesis, we will concentrate on the latter and present the Auto-ID Object Model and the services related to it. The object model is an extended hierarchical, symbolic location model, in which physical objects also define locations. The object hierarchy represents containment relationships between objects. Objects are identified by a unique identification, for example, the electronic product code (EPC) proposed by the Auto-ID Center at MIT, and can have dynamic and static properties (e.g. temperature, size or expiry date). The model also includes object persistence, query capabilities, and business event generation. The latter is a

mechanism based on state machines that provide a declarative programming model to formulate subscriptions for business events.

In this thesis, we also discuss possible approaches in the implementation of the described model to provide a scalable and performing infrastructure. We analyze approaches including temporal databases, tree database extensions and object databases. In addition, we present a prototypical implementation of such an infrastructure, called the Object Monitoring System (OMS).

1.2.2. Visual Tool Approach

A software component or infrastructure that implements the Auto-ID Object Model and its services facilitates the development of Auto-ID applications to a great extend. However, an Auto-ID infrastructure usually consists of many different hardware and software components (e.g. readers, sensors, filtering and aggregation components, distributed OMS components) that have to be customized to fit the need of a certain application or application domain. Such an instantiation and configuration is a tedious task and requires programming skills in addition to RFID and application domain knowledge. During runtime, the management of such an infrastructure poses similar problems. To address these issues, we propose a visual tool-based approach to the instantiation, configuration and management of an Auto-ID infrastructure.

In particular, we show how such a tool allows a non-programmer to visually construct a hierarchical object model using a concrete representation of floor plans. Objects could be, for example, buildings, rooms, shelves, boxes or fork-lifters. Readers and sensors can be placed and configured, and are automatically linked to objects. In addition, the business event subscriptions can be defined.

We also show that the tool facilitates the overall process at different stages in the lifecycle process. During design time, it allows instantiating the Auto-ID application model towards a certain Auto-ID application (e.g. a retail store management application) and facilitates the configuration of all components of an Auto-ID infrastructure. For the deployment phase, it offers a different view to visually setup and to connect the different hardware and software components. During runtime it can act as a “management cockpit” to dynamically visualize the state of the infrastructure and allow changes to the setup and configuration. In addition, it can also be a testing tool to simulate the movement of objects to test the business logic.

1.2.3. Methodology

In order to derive a system family model for Auto-ID applications, we analyze applications of different domains such as consumer object identification market, industrial automation, mobile asset and supply chain management, traditional applications such as livestock tracking, ski ticketing, or toll collection, and novel and envisioned applications such as smart spaces enabled by Auto-ID technologies.

Coming from the application analysis we derive a list of Auto-ID application requirements that a system should fulfill to facilitate the development of Auto-ID applications. Based on these requirements, the Auto-ID Object Model and the OMS were designed and implemented.

To evaluate the proposed Auto-ID model and visual tool-based approach, we applied the Auto-ID application development process to different applications and evaluated it:

1. The Smart Medicine Shelf is an automated shelf in hospitals that keeps track of different kinds of medications that require different conditions (e.g. certain vaccines have to be cooled) and require different access rights (e.g. a nurse is not allowed to access certain drugs).
2. The retail store supply chain application provides support for the logistical management of a retail store (e.g. keep track of incoming and sold goods, automatic replenishment of products in the sales area, control of temperature in freezers, etc.).
3. The tool management in aircraft maintenance application keeps track of tools and parts in an aircraft maintenance environment and includes the Smart Toolbox and Automated Tool Inventory applications.
4. The Augmented Knight's Castle application is a pervasive computing playset which enriches the child's pretend play by using background music, sound effects, and verbal commentary of toys that react to the child's play.

1.3. Thesis Outline

In the *second chapter*, we provide an overview of several Auto-ID and sensor technologies (including barcode, RFID and Bluetooth) with an emphasis on RFID. We briefly discuss the technological principles of operations insofar as they are important to the contribution of the thesis.

The application scenario of a pharmaceutical supply chain is described in the *third chapter*. Based on this scenario we derive system requirements for the building blocks to support Auto-ID application development.

The *fourth chapter* presents the Auto-ID Object Model and the services related to it. We begin by introducing the object model and argue that it is a sufficient model to abstract from RFID specific details. We continue by describing the details of the model and the services that a component implementing the model has to provide. In addition, the Auto-ID based micro business processes that the model provides are presented. We finally discuss possible approaches in the implementation of the described model to provide a scalable and performing infrastructure and present an overall design of such an infrastructure called the Object Monitoring System (OMS). The related work section of this chapter describes related approaches including the work of the Auto-ID Center at MIT (i.e., the EPC Network), Auto-ID Infrastructures, database-centric approaches and ubiquitous computing infrastructures.

In the *fifth chapter* we give an overview of visual and generative programming concepts and propose a visual tool approach to the instantiation, configuration and management of an Auto-ID infrastructure. We present the visual "all-in-one" tool, describe its concrete representation of the application, its roles in the instantiation, configuration and management process, and argue for its usage instead of 'manually' configuring and managing an infrastructure. The design of the visual tool is discussed taking state-of-the art end-user programming concepts into account. In addition, the tool includes application generation and deployment concepts that are also discussed in detail.

The *sixth chapter* presents several case studies from different application domains as a proof-of-concept of the presented work. For each case study, the Auto-ID application development process is sketched and compared to a traditional approach to argue for the effectiveness of the presented model, tool and process. The analyzed Auto-ID applications are: The Smart Medicine Shelf, the tool management in aircraft maintenance application including the Smart Toolbox and Automated Tool Inventory and the Augmented Knight's Castle.

In the *seventh chapter* we draw the conclusion and provide an outlook to further work.

2. Auto-ID Technologies

Several technologies exist that allow identifying physical objects, animals and human beings automatically. Commonly used contactless Auto-ID technologies are barcode and RFID, however, other wireless technologies that provide a unique id have been used such as infrared communication (IrDa), Bluetooth, ZigBee or wireless LAN (WLAN).

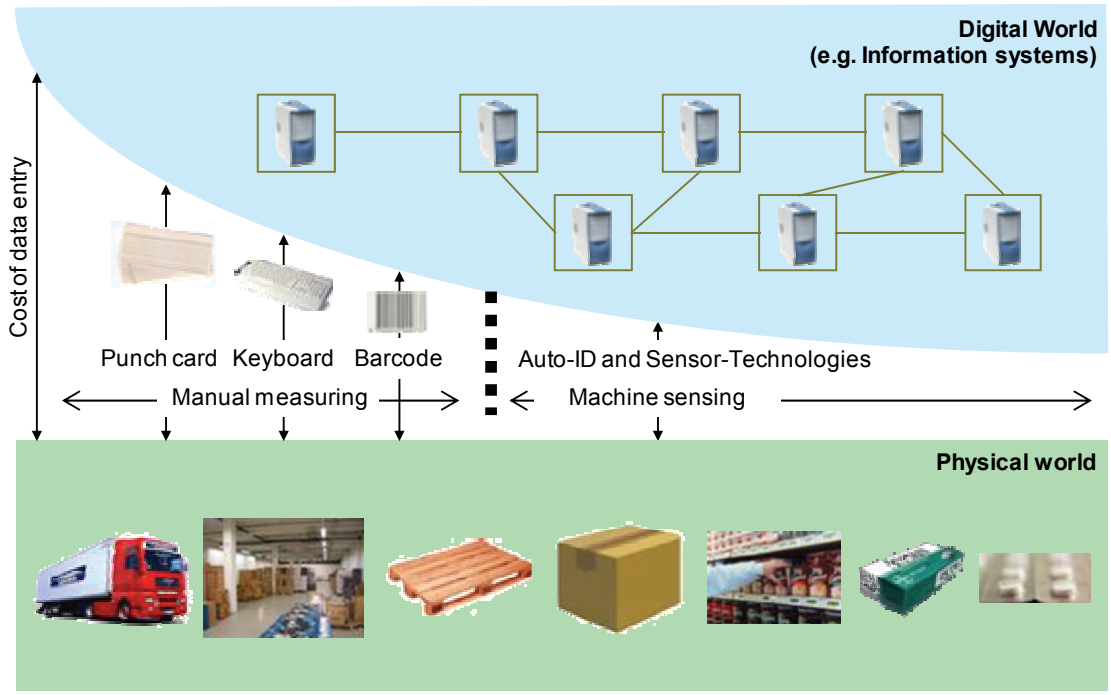


Figure 2-1 Bridging the media gap between the physical and digital world (based on [62])

A major focus of using Auto-ID technologies is the avoidance of media gaps since they bridge the gap between the real world of physical objects on the one hand and the digital world in the form of information systems (e.g., ERP or warehouse management system) on the other (see ure 2-1) [62]. The consequences include lower error rates, higher process efficiency, enhanced product quality and cost savings thanks to faster and

better information processing. Moreover, Auto-ID technologies provide the basis for many other applications which go beyond simple identification such as continuous cold chain monitoring using sensor technology or the real-time localization of objects in production or logistics processes.

Historically, barcodes have been in use in the US since 1974 and in Europe since 1977 and were the first commercial Auto-ID technology that has been used in retail and logistics to identify products. Generally, barcodes are used to only identify product classes; however, standardized barcode extensions also allow identifying product instances. RFID technology has been used since 1966 only for electronic article surveillance (EAS) and since 1979 to identify animals. Further RFID applications in the 1980ies and 1990ies were car toll systems, car immobilizers, access control (including ski lifts) and first payment applications. The last couple of years, the field of RFID application is now increasingly expanding. The increasing usage of RFID over barcode is also due to the many benefits of RFID compared to barcode technology. The public discussions of RFID transponder technology promoted by the Auto-ID Center and the planned use of RFID in the supply chains of retail companies (e.g., Wal-Mart and Metro) or in the logistics processes of the US Department of Defense have led to a growing emphasis on the potentials of this technology for improving business processes.

2.1. Barcode Technology

The barcode technology was commercially introduced through the Uniform Product Code (UPC) Council, which later became the Uniform Code Council (UCC). The European Article Association, today known as EAN International, was founded with the objective to develop a barcode system compatible to the system by UCC that should be used outside North America. Most application today using barcode are based on the EAN.UCC barcode system.

Two types of barcodes exist: (i) Linear barcodes (i.e. 1-dimensional barcodes) that are mainly used to store article numbers and are printed on product packages and (ii) 2-dimensional barcodes that can store more information and used, for example, in document management and ticketing applications.

Both 1-dimensional and 2-dimensional barcodes are used world-wide to identify products, packages and transportation units along supply chains.

EAN-13 is the barcode symbology that is most widely used in Europe, for instance, to scan products at the point-of-sale (POS). From January 1st 2005 on, all American and Canadian companies that are members of UCC have to be able to also read EAN-13 barcodes. 2-dimensional barcodes are mainly used in applications that require coding more data than simple identification numbers, for example, a shipping address.

The EAN.UCC system is managed by the standardization organization GS1 [6] and provides world-wide unique numbers for the identification of the following types of objects along the supply chain:

- Goods and services (Global Trade Item Number)
- Shipping containers (Serial Shipping Container Code)
- Reusable containers, packages and pallets (Global Returnable Asset Identifier)
- Objects (Global Individual Asset Identifier)
- Service relationships (Global Service Relation Number)
- Locations (Global Location Number)

Figure 2-2 Barcode types of the EAN.UCC systems

These identification numbers are encoded in barcodes commonly using the following five symbologies: EAN/UPC, ITF-14, UCC/EAN-128, Reduced Space and EAN.UCC Composite and Data Matrix [78]. A symbology describes the rules that define the coding of a number using the graphical symbols of barcodes (i.e. the lines and gaps). The symbologies

for 1-dimensional barcodes of the EAN.UCC system that differ in the number of ciphers and the coding schema are shown in Figure 2-2.

The differences of the existing 2-dimensional barcodes are

- the area of application for which they have been developed,
- the number of bits that can be coded,
- the robustness towards errors, and
- the quality of the hardware that is needed to read them.

Some of the 2-dimensional barcodes have been standardized by ISO/IEC such as Data Matrix, MaxiCode, QR Code, und PDF417 (see Figure 2-3). All these codes have a variety of options and features, for example, several variants for error correction.

Figure 2-3 ISO/IEC standardized 2-dimensionale barcodes

2.2. RFID Technology

The following sections give a technical overview of RFID technologies. A more detailed technical presentation of RFID technology is given in [61, 157, 98].

2.2.1. RFID System Components and Operating Principles

A typical RFID system comprises three components: the RFID reader with the coupling unit (i.e. coil or antenna), the RFID transponder(s) and the host with the application using the RFID data (see Figure 2-4).

The reader, which is connected to the host via a serial or network connection, serves depending on the specific RFID system solely as a reader or as a read/write unit if the transponders contain user memory. The application on the host sends commands and data to the reader which after executing the commands sends the return data back to the application. Examples for commands are “read all identification numbers of all transponder that are in the read range” or “write the following data in the user memory of the transponder with the following ID”.

The reader encodes and modulates the commands on an alternating magnetic respectively electro-magnetic field. For passive and semi-passive transponders the field is not only used to transmit data but also the energy to power the microchip of the transponder. Active transponders have their own batteries as energy source.

All transponders that are within the field and read range of the reader receive the transmitted commands and data and then demodulate and decode them. After executing the command they transmit the return data back to the reader using the field.

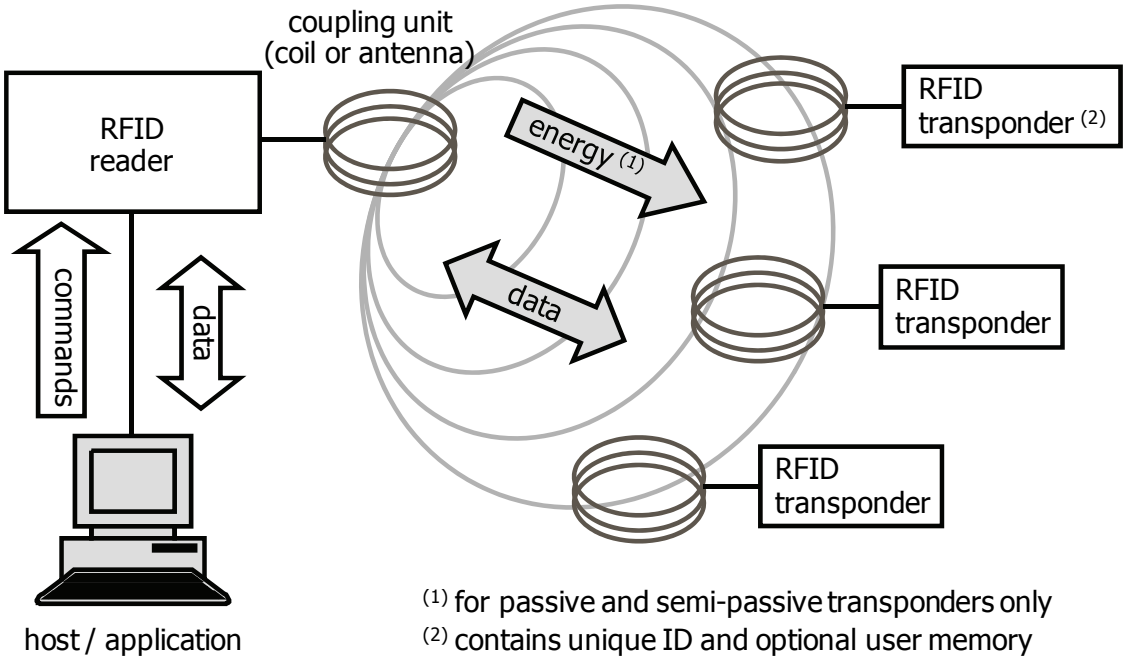


Figure 2-4 Operation principles and components of an RFID system

The common *operating principle* for RFID technology is to use an alternating magnetic respectively electro-magnetic field to exchange information between the reader and the transponders. There are two different underlying technologies used: conductive coupling and backscatter coupling. Conductive coupling is based on the principle of induction, that means, the reader creates with its coil an alternating magnetic field that induces a current in the coils of the transponders. The transponders use the same field to transmit information back to the reader. Backscatter coupling uses an electro-magnetic field that is created using the antenna of the reader. Such an electro-magnetic wave propagates in the space and

induces a current in the antennas of the transponders. The transponders use the reflected wave to transmit information back to the reader.

The *operating frequencies* of most RFID systems lie within the license-free ISM bands (Industrial-Scientific-Medical) which are made available worldwide for industrial, scientific and medical applications. In addition, there is the frequency range below 135 kHz and around 900 MHz. This means that the typical operating frequencies of an RFID system fall within the following four ranges:

- 100–135 kHz (low frequency, LF, inductive coupling)
- 13.56 MHz (high frequency, HF, inductive coupling)
- 868 MHz (Europe) / 915 MHz (USA) / 950–956 MHz (Japan, planned) (ultra-high frequency, UHF, backscatter coupling, first systems using inductive coupling)
- 2.45 GHz and 5.8 GHz (microwave, MW, backscatter coupling)

Regulations impose further restrictions on the operation of RFID systems within the authorized frequency bands. The regulations state the maximum permitted transmission powers or field strengths, permitted sidebands and standardized measurement methods. The frequencies in the range around 135 kHz and 13.56 MHz are available for RFID systems worldwide. This is not the case with frequencies in the UHF range where efforts are being made to harmonize regulations in order to permit worldwide operations of UHF RFID systems.

A *transponder* typically consists of a microchip and a coupling unit. Depending on the different technology the coupling unit is either a coil (for conductive coupling) or an antenna (for backscatter coupling). The unique identification number of the transponder is stored in a memory block of the microchip. The identification number can be written during manufacture in the factory or later prior to initial use. In addition to transponders that only contain an identification number, most transponders in use also contain additional user memory which can be written and read. The range for write access is usually shorter than for read access as the former consumes more energy. For applications that require storing more complex data and keeping it secure, transponders with more complex memory structure and security features exist. The user memory of such RFID transponders is usually split into sections for which access can be regulated by means of codes or challenge response methods.

Many different forms and materials are used for transponders depending on the specific RFID technology and application domain. Examples are so

called “smart labels” which are placed on an adhesive foil to easily attach them to products and packages, or transponders that are packaged in special plastic to be resistant against acid. Figure 2-5 shows a selection of different types of transponders (a. LF transponders in glass capsules for animal tracking, b. standard HF smart label, c. UHF transponder with dipole antenna, and d. UHF transponder with two folded dipole antenna).

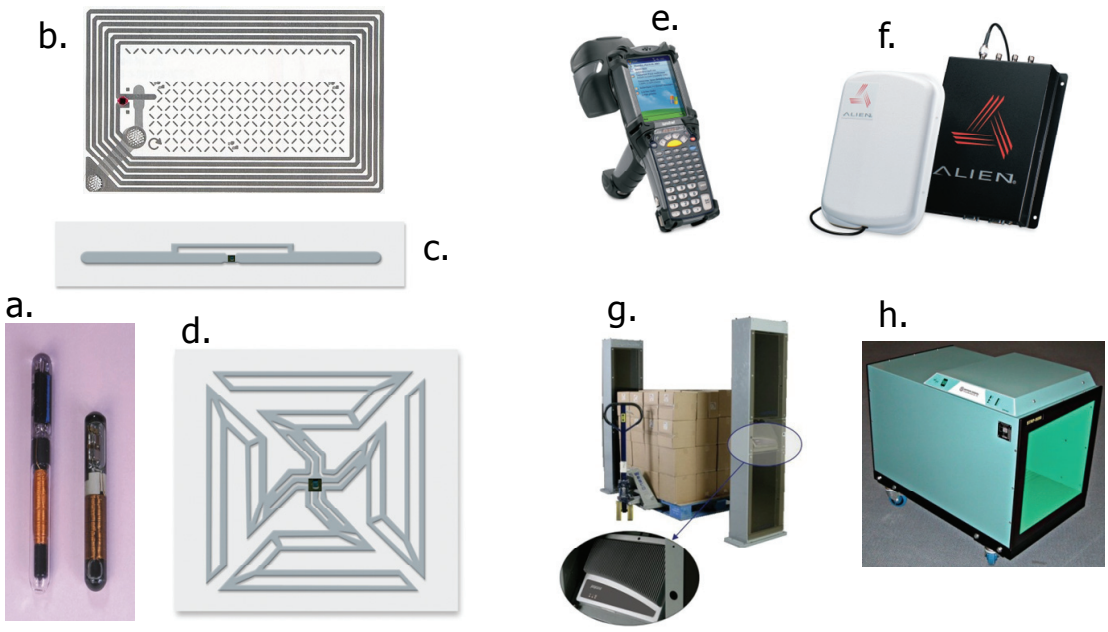


Figure 2-5 Different types of RFID transponders and readers

The form and size of the *reader* depends mainly on their coupling unit, that means, the technology used. Mobile readers are components where the reader, coupling unit and host with application are integrated and packaged together to be handled by a person for mobile readings of transponders typically with lower read ranges (see Figure 2-5, e.). Standard readers are typically packaged in an industry conform casing and an external coupling unit (see Figure 2-5, f.). To be able to read a large number of transponders, for example packages or products on a pallet at incoming goods, the coupling units of a reader can be arranged to a “gate” which also increases the read range. In a similar way the coupling units of a reader (especially for HF) can be arranged to form a “tunnel” where the transponders move through. The magnetic field strength inside the tunnel can be much higher that commonly used since the tunnel shields the field which leads to higher detection rates.

RFID systems can be divided into three categories, based on the typical read range: systems which operate up to a range of 1 cm are referred to as *close coupling* systems. They work with inductive coupling and are mainly used in security-related applications such as access control systems or payment systems. *Remote coupling* systems also work with inductive coupling but in a distance range of up to one meter. Depending on application, the operating frequency is typically 135 kHz or 13.56 MHz. Systems with a range of more than one meter are called *long-range* systems. They typically operate with operating frequencies of 868/915 MHz or 2.5 GHz. However, various manufacturers also refer to RFID systems with a range of up to a meter as long-range systems. Generally speaking, the achievable read range depends on a large number of factors (among other operating frequency, size, shape and quality of the RFID transponder's antenna and ambient conditions). It is therefore difficult to compare the read ranges of different RFID systems. Under ideal conditions, an RFID system operating in the UHF band can achieve a read range of 5 to 7 m. Under real-life conditions this range is only seldom achieved. For semi-active RFID systems the read range is up to 15 m, for active systems up to 100 m. LF and HF systems have a typical read range of 1-1.5 m. In comparison with UHF systems, however, they are less susceptible to interference through ambient conditions.

The strengths of RFID, particularly in comparison with the barcode, are to be found in the fully automatic, simultaneous detection of several RFID transponders, with no line of sight required between reader and RFID transponder. This means that RFID transponders can be embedded in objects without them being visible from the outside, thus enabling their use in extreme conditions such as dirt or heat. A higher read range is also possible than is the case with barcode scanners; furthermore, information on an RFID transponder with memory can be changed while in use, which is not possible with a barcode.

2.2.2. RFID Standards

The various standards for RFID can be classified in the following three categories: Air-interface standards, tag data standards and reader-host interface standards.

Air-interface standards specify the interface between RFID transponders and readers and are intended to ensure that the RFID transponders and readers of different manufacturers can communicate with one another. For

this purpose, the standards define not only the physical layer with carrier frequency, encoding, timing, modulation technique and data transmission rates but also the multiple-access method and the command set. The appropriate standardization efforts have largely been undertaken by the Joint Technical Committee 1 (JTC1) of the International Standards Organization (ISO) and the International Electrotechnical Commission (IEC) as well as more recently by EPCglobal Inc., the successor organization of the Auto-ID Center.

In the HF frequency band, the standards are ISO 15693, ISO 14443 and ISO 18000 Part 3. In addition, EPCglobal is developing a standard for HF. In the UHF range there are the following two standards: ISO 18000 Part 6 and EPCglobal UHF Class 1 Generation 2 [57]. Alongside the standardization efforts in the HF and UHF ranges, there are also standards for the LF and MW frequency ranges under ISO 18000. Here, the LF section largely corresponds to the earlier standard ISO 11785 and its further development ISO 14233.

As far as *tag data standards* for RFID transponders are concerned, a basic distinction can be drawn between two different approaches. In all parts of ISO 18000 the individual RFID transponder is designated by a unique identification number which is already written onto the RFID transponder's microchip during the manufacturing process. Information on the product designated by the RFID transponder can be stored in the RFID transponder's memory by the user, while the size of the memory is variable and only the maximum value is specified (e.g. 8 kB for ISO 18000 Part 6 Mode A). The RFID transponders which meet one of the specifications of the Auto-ID Center or its successor organization EPCglobal merely contain a unique identification number, the "electronic product code" (EPC), but no additional memory. Information on the product to which the RFID transponder is fixed, such as e.g. manufacturer's code, product type and serial number, is encoded in the EPC itself [50].

Reader-host standards relate to the communication between reader and IT infrastructure. Within the framework of EPCglobal, the Reader Protocol [56], Reader Management [55] and Low Level Reader Protocol (LLRP) [53] standardize in different ways the access to readers for applications. The LLRP, which allows more control over the underlying reader hardware and air-interface, is currently the preferred protocol by EPCglobal and reader hardware manufacturer and will succeed the Reader Protocol.

2.3. Other Wireless Auto-ID Technologies

In addition to the most prominent Auto-ID technologies, other wireless communication technologies that provide a unique ID can be used for the purpose of identification. Potential technologies widely available are infrared communication (IrDa) [7], Bluetooth [2], ZigBee [19] or wireless LAN (WLAN) [17].

A "tag" using such a technology could simply be the communication unit itself built into another device (e.g. a cell phone or a notepad) or a special "active tag" hardware that includes the communication unit. Existing sensor nodes or special modules such as the BTnode [27] (using Bluetooth), the Tmote [15] (using ZigBee) or the Epsilon Wi-Fi module [5] (using WLan) can easily be utilized as "active tags" (see Figure 2-6 and also Section 2.4). Using RFID terminology, a reader for such tags is a base station using the same communication technology and special software that manages the discovery of the tags.

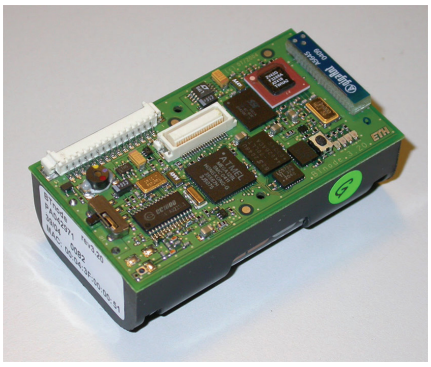
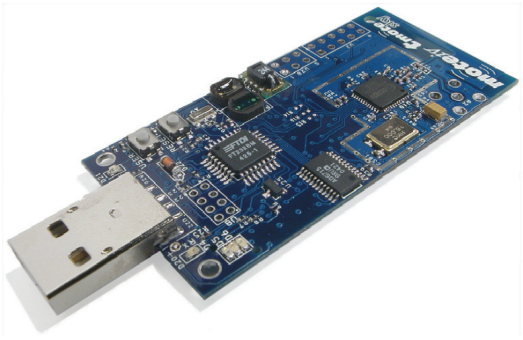
(a)  (b) 

Figure 2-6 BTnode (a) and Tmote (b) sensor nodes (from [3] and [15])

Since the different technologies have their advantages and drawbacks, applications have to select a technology depending on their specific requirements. Bluetooth and WLAN (i.e., communication standards based on IEEE 802.11) have the disadvantage of high energy consumption since the communication unit has to be active for a reader to discover these units. However, Bluetooth provides sleep modes to reduce energy consumption and has been successfully used as communication technology for sensor nodes [91]. Another drawback of Bluetooth is the discovery protocol which can take up to several seconds to discover and connect to a tag. Moreover, the reader can keep connections to only 7 units at a time and connection switching has to be performed for more units.

ZigBee (resp. the underlying communication protocol IEEE 802.15.4) has the advantages of very low energy consumption and a simple and fast discovery protocol. In addition, a ZigBee unit can easily connect to 65535 other units. The drawback of ZigBee is the low data transfer rate if large amounts of data have to be transmitted.

Infrared as a communication technology has the drawbacks of necessary line-of-sight, low read range and narrow read angle, and susceptibility towards artificial light. Infrared is therefore limited for applications with few tags in close proximity to a reader.

2.4. Sensor Technology

Sensor technology in addition to or in combination with Auto-ID technology provides a means to automatically obtain environmental information about the world of physical objects to be stored in the virtual world of information systems. Many applications such as supply chain management benefit from environmental information that is connected to locations, goods or transportation units. For example, knowing that the temperature of a chemical for IC production left a well-defined temperature range during transportation, or knowing that a container was dropped or opened during transportation can save production and logistics costs. For simple environmental information such as temperature, sensors bridge the media gap leading to more accurate and detailed data in the information systems. Moreover, sensors allow measuring environmental parameters that could otherwise not be acquired.

Results coming from engineering fields such as micro system technology and nanotechnology lead to new advances in short range sensor technology. The size of sensors is further reduced and the environmental parameters that sensors can measure lead from typical physical values such as temperature, light or acceleration to sensors that are able to detect gases and liquids. Moreover, new generations of sensors are able to report their measurements wirelessly using energy from an electro-magnetic field in their environment [112]. Sensors can also be incorporated directly into RFID tags, for example a humidity sensor build into an EPCglobal Gen2 transponder [161].

The three fields that contribute to the overall functionality of a sensor are: Sensor structure, manufacturing technology, and signal processing algorithms [89]. The advances in sensor technology can therefore be attri-

buted to the technical progress in these three fields. As Kanoun et. al. states, “in the last years, a significant upturn is observed in these fields involving a great potential for completely novel approaches of sensors and sensor systems” [89].

According to Singh [140], the following development trends contribute among others to the advances in the field of sensor technology:

- *Microelectromechanical systems (MEMS)* technology to develop new forms of sensing.
- Developments in the electronics area lead to *miniaturization and ruggedization* of sensors increasing the potential application areas.
- *Incorporation of intelligence* into sensors (e.g., microprocessor/ microcontroller- or chips-based) enables preprocessing and smarter sensing such as real-time processing already on the sensor platform.
- *Networking* of sensors enables collaborative and smarter sensing.
- *Standard evolution* such as the IEEE 1451 sensor standard family [108, 110] enables plug and play of sensor platforms and leads to a greater adoption of sensors.

Networking of sensors emerged to an own research field of sensor networks. A sensor network consists of sensor nodes that form a virtual wireless network to collaboratively measure different environmental parameters and perform processing of sensor data. Sensor nodes are devices that combine sensing, computing and wireless communication capabilities. Typically, they consist of an embedded microcontroller platform, a radio transceiver with antenna, different types of sensors and a battery (see Figure 2-6 for two typical sensor node hardware). The topology of the network and other properties depend highly on the application the network is used for [59, 84]. The main application areas for sensor networks are: Species and environmental monitoring, agriculture and farm-animal monitoring, intelligent environments, production, logistics and asset tracking, facility management, military, and disaster relief [90, 128].

3. Auto-ID System Requirements

The following paragraphs outline a supply chain scenario – a pharmaceutical supply chain with a pharmaceutical manufacturer, a distribution center and a drug store (see Figure 3-1) – that represents common applications of Auto-ID technology. Emphasis is given on the distribution center (see Figure 3-2). We use this scenario and the use cases to derive and illustrate the system requirements that an Auto-ID infrastructure should address.

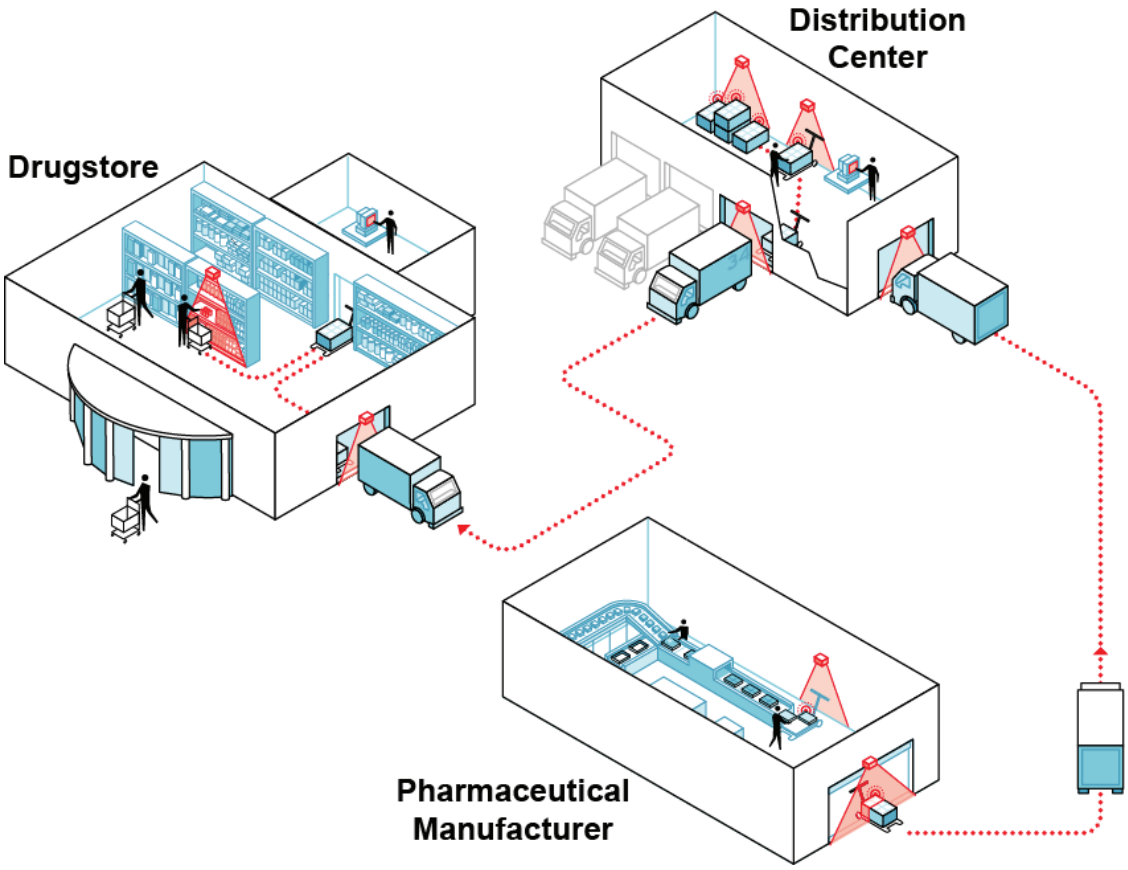


Figure 3-1 Auto-ID usage in a medical supply chain scenario (source: EPCglobal)

Pharmaceutical goods are produced and shipped to the distribution center. When leaving the manufacturer, the goods are identified by readers at the dock doors for outgoing shipments, checked that the shipment is complete and correct and an advance shipping notice (ASN) is generated for the distributor.

The goods arrive at the distribution center and are identified by the readers at the dock door for incoming goods. In addition to the IDs, state about the goods such as temperature during transport is received from sensors on the cases. The captured information is processed and the shipment is checked for completion and correctness using the information of the ASN. Aggregated information is then transmitted to enterprise resource systems. The goods are placed in the warehouse, where readers regularly scan the inventory and monitor the state of the goods. At regular intervals the inventory counts of the corresponding product categories are updated in the legacy warehouse management (WM) systems. If the state of certain goods is invalid, warehouse staff is notified to check and correct the conditions.

Goods for drugs and retail stores are picked from the warehouse and packed at the corresponding pick and pack station. A reader monitors the tagged items currently packed so that a local application can support staff with a near real-time comparison of items actually packed and the items on the pack list.

Before the shipments are loaded into the trucks at the loading dock, they pass a reader that scans the tag on the pallet and passes this information to the supply chain management system, which sends an ASN to the recipient of the shipment. On a nightly basis, all tag IDs of the items packed and shipped are transmitted to the healthcare authorities to comply with pedigree legislation.

To maintain an adequate service level, the readers report exceptions to a remote system monitor. Auto-ID system integrators can inspect a configuration of a reader and reconfigure reader devices remotely.

Based on an analysis of different Auto-ID applications including the above and the study of other work on Auto-ID/RFID middleware [28, 38, 66, 129, 131], we identified the following high-level system requirements an Auto-ID infrastructure should meet.

The system requirements are related to and dependent on each other (see Figure 3-3) and can be roughly categorized into the following two

groups: System requirements dealing with “low-level” Auto-ID issues, that means, issues that are concerned with Auto-ID reader and data management (Requirements (R11) to (R18)) and system requirements dealing with “high-level” Auto-ID issue, that means, issues that are dealing with Auto-ID data organization and enrichment, which is closer to the application domain (Requirements (R1) to (R10), (R19) and (R20)).

The system requirements are further refined and specified in the Auto-ID infrastructure concepts and the Auto-ID Object Model which are presented in section 4.

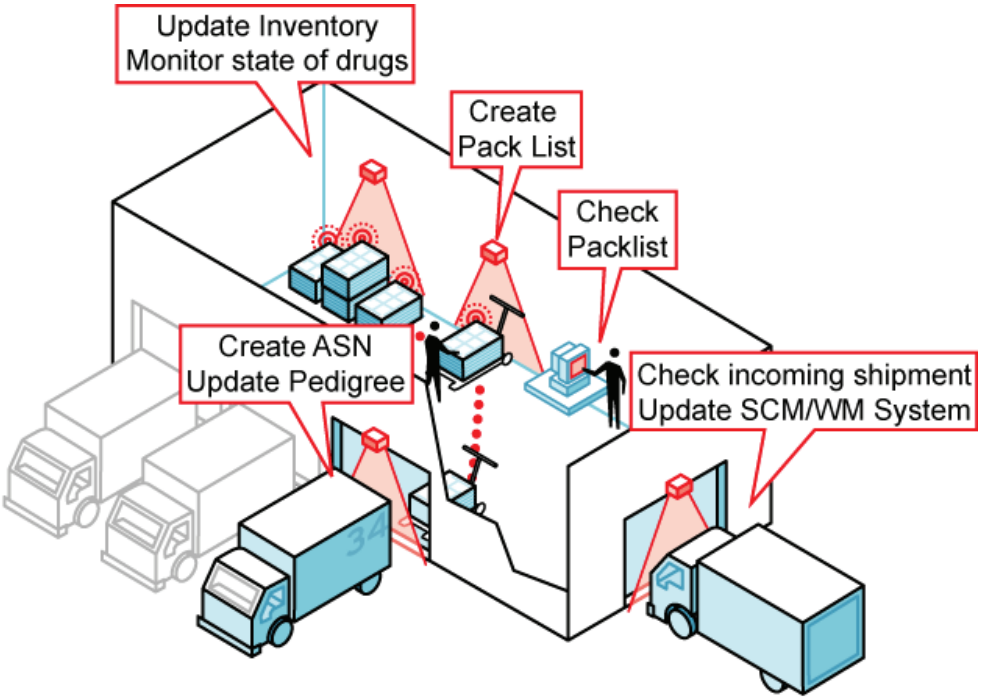


Figure 3-2 Auto-ID readers in a distribution center feeding captured data to different applications (source: EPCglobal)

(R1) Object Representation

Applications that deal with physical objects or human beings that are automatically identified are commonly not interested in Auto-ID specific details such as readers, tags or tag memory; instead they would like to deal with representations of the physical objects. The system should therefore provide an object representation of the physical objects in which an application is interested. The system has to keep the object representation in sync with the corresponding physical objects in the real world.

Physical objects that are monitored by an Auto-ID system are for example products that move through a supply chain such as the drugs in the scenario above, movable assets such as reusable containers or pallets, or animals on a farm. Human beings could be monitored for access control or security reasons, or in healthcare environments such as hospitals or old people's homes.

(R2) Object Persistency

The objects in the system that represent the physical objects with their data have to be stored persistently to keep a history of objects and allow applications to query the data. For example, to detect counterfeit drugs the SCM application queries the system to retrieve the track and trace information about certain drugs in question from production to their current location. If the trace is not valid or broken, the drug could be a potential counterfeit. The system has to provide both a persistency mechanism for the object representations and a query mechanism for the application to retrieve persisted data.

(R3) Object Relationships

Many applications are interested in information related to several objects in a certain relationship. One important relationship is the neighboring objects relationship, i.e. objects are neighbors if they are together at the same location. A chemical monitoring application might, for example, be notified when containers with hazardous chemicals are stored together in a room.

Another important relationship is the containment relationship, that means, objects that are contained in another object. Examples for containment are pharmaceutical goods that are contained in a shipping case or retail items in a shopping cart.

(R4) Location Information

The system has to provide a mechanism to enrich Auto-ID and sensor data with location information since applications are interested in locations of objects, for example, when a certain object was at a certain location. The query of a track and trace application should return a list of times, locations and object states for that given object.

For many applications, symbolic location names (e.g., “incoming goods dock door 110” to specify the location of the dock door number 110 of the incoming goods section) are sufficient in the business context definitions. Since the symbolic location names have to be manually defined for an application, the system has to provide mechanisms to add and modify the location names.

(R5) Object Identification

Object representations in the system have to be automatically synchronized with the physical objects they represent via Auto-ID and sensor technologies. The object representation should reflect the physical objects in near real time to allow applications quick interactions with the physical world. The system has to support different Auto-ID and sensor technologies and needs to be designed in a way to allow integration of future technologies.

(R6) Object Identifier

Physical objects have to be uniquely identified in object representation of the system. A variety of identifier schemes are possible ranging from simple numbers to article or product numbers, to symbolic names. The system has to support different numbering schemes; however, the numbering scheme has to be meaningful to the application.

To automatically determine the ID for a physical object, the ID stored in the memory on the Auto-ID tag can be used, either directly or by translating the ID to an object ID. One prominent example for an object identifier schema is the electronic product code (EPC) [50] that comprises three parts, namely the product manufacturer ID, the product type ID, and the serial number. The EPC is available in different representations. This includes a representation that is suitable for storage in tag memory as well as representations as uniform resource locators which can be easily processed by applications and humans. To convert between the different representations, a tag identifier translation mechanism is required [51].

(R7) Object Data Enrichment

In many cases, identified objects can be enriched with additional related data that is of interest to the applications. This could, for example, be information about the color, size or weight of a product, an expiry date for a

drug, the current temperature inside of a container, or the number of usages of a tool. The additional object data should be retrieved from different sources such as databases, dynamically provided by sensors, or even stored on memory of the tag attached to the object. The information can be specific for an object class or an object instance, and it can stay the same or change over the life cycle of a physical object (i.e. static or dynamic information).

(R8) Physical World Interaction

The system has to provide the means to allow applications the interaction with the physical world. In addition, it has to provide the means to allow the physical world to trigger functionality in the system respectively the system to react to certain conditions in the physical world.

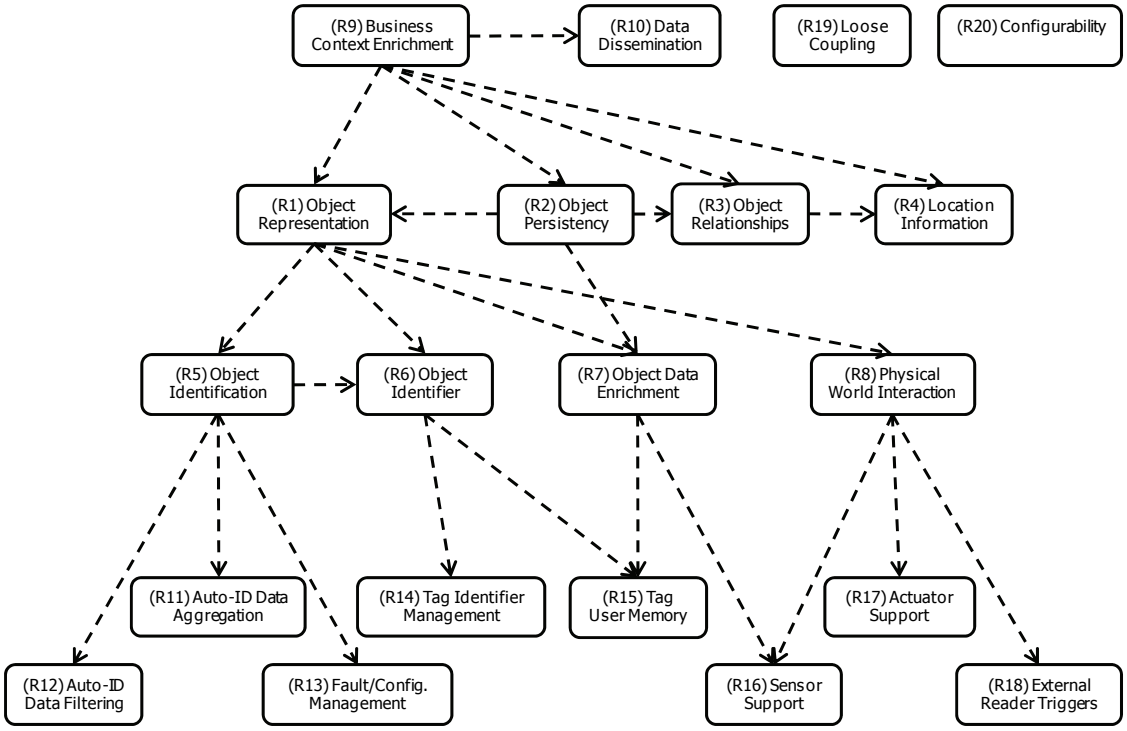


Figure 3-3 Overview of Auto-ID infrastructure requirements including their dependencies

(R9) Business Context Enrichment

The dynamic changes in the object representation of the physical objects correspond to the changes in the physical world. Many applications are not interested in all these changes but only in exceptional states related to

objects, their properties and relationships. Examples in the presented scenarios are the completion of an incoming shipment, reaching the expiry date of drugs in a shelf, or exceeding a temperature threshold inside a cool truck.

The system has therefore to provide a mechanism that interprets the captured Auto-ID data respectively the object representation in specific business context over time.

(R10) Data Dissemination

Many benefits commonly associated with Auto-ID require data sharing across the supply chain [22]. The information in the object representation and the data enriched by the business context is usually of interest not only to a single application, but to a diverse set of applications across an organization and its authorized business partners.

The data must thus be broadcast to the entities that indicated an interest in the data, such as the SCM system. Due to the event-driven nature of many processes monitored with the help of Auto-ID systems, there is a need to support asynchronous messaging as well as a query-response model. In the example above, the health care regulatory authority executes a daily query to retrieve data, whereas the pick and pack application is notified asynchronously whenever a new package of drugs arrives. Different applications also require different latencies. Applications that need to respond immediately to local interaction with the physical objects, such as the pick and pack application, require short notification latency that is comparable to the observation latency. Legacy applications that are not designed to handle streaming data might need to receive batched updates on a daily schedule.

(R11) Auto-ID Data Aggregation

Auto-ID systems generate a significant amount of data that can be aggregated in a number of different ways. Auto-ID data can be aggregated in the time domain, e.g., by generating entry and exit events, and in the space domain, e.g., by combining data from different readers and reader antennas that observe the same location or by detecting the movement of a tagged object.

In our scenario, a single dock door could be monitored by two Auto-ID readers. The Auto-ID events are aggregated and the original reader ID is

replaced by the ID of a logical reader that was defined for the dock door location. Since Auto-ID permits identification at the instance-level rather than at the class-level, there is also the possibility to report the quantity of objects belonging to a specific category. This kind of an aggregation is needed for those legacy IT systems that cannot handle instance-level data.

(R12) Auto-ID Data Filtering

In addition to aggregation, filtering can greatly reduce the amount of data that is generated. Different applications are interested in a different subset of the total data captured, based on the reader, reader antenna, and tag involved. In the supply chain scenario, certain application might only be interested in tags attached to pallets that can be filtered using the tag type.

(R13) Fault and Configuration Management

The proliferation of readers and sensors mandates fault and configuration management. This includes monitoring the health of Auto-ID readers and accessing the Auto-ID reader configurations remotely. The result is that Auto-ID readers can be integrated into IT service management, just like any other computing hardware.

(R14) Tag Identifier Management

Auto-ID permits the unique identification of objects through the identification of a tag attached to that object. The object can therefore be identified either by an object identifier stored in the memory on the Auto-ID tag or an ID that is unique for the tag. Different numbering schemes exist for such identifiers and have to be supported by the system.

(R15) Tag User Memory

Many tags feature not only memory space for an identifier, but for additional data. The system should thus provide means to write to and read from this additional memory. This additional memory can then be used to store application or object data such as expiry dates in order to facilitate data exchange where no network access is available.

(R16) Sensor Support

In many applications it is not sufficient to only identify objects. The current state of the objects in the physical world has also to be detected. This is important for many types of goods such as perishable goods that have to be cooled, drugs that have to stay within a predefined temperature range, chemicals that should not be exposed to light, or fragile goods that easily break if not handled with care. The system has thus to provide the means to integrate sensors such as temperature, humidity, light or shock sensors, process their data and make the object state accessible by the applications.

(R17) Actuator Support

Applications often have to quickly interact with the physical world using different kinds of actuators. For examples, a smart medical shelf application has to keep a shelf with anesthetics locked if an unauthorized person identifies himself and only unlock the shelf for an authorized medical doctor. Other examples are actuators to give feedback to users such as traffic lights to signal the state of a shipment to a fork lift operator.

(R18) External Reader Triggers

In many applications, there is no need to operate Auto-ID readers continuously. Due to the limited bandwidth available, it is even undesirable to have readers transmit, while no tags are present [66]. To initiate the tag inventory process at a reader when there are tagged objects arriving in the read range, external sensors, such as motion sensors, should thus be able to trigger the readers. In our example, the readers at the dock door only operate when items enter their read range. The arrival is detected by a motion sensor in front of the dock door.

(R19) Loose Coupling of Components

The setup of an application scenario involves a variety of different kind of components, both physical components such as readers or sensors and software components such as databases, low level and high level Auto-ID data processing and persistent data storage, and ERP systems. During runtime some of these components might get unavailable or new components are added. In order for the system to be flexible, the system and the components should be loosely coupled.

(R20) Configurability of System

Since the setup of an application scenario involves configuring and instantiating a variety of different kind of components, both physical and software components (see also section (R19)), the system has to provide the means to easily configure a complete application setup and instantiate all components. Application domain experts, i.e., non-software engineers, should be able to perform such a configuration and instantiation.

(R21) Privacy

The intended deployment of Auto-ID-based tracking solutions in today's retail environments epitomizes for many the dangers of an Orwellian future: unnoticed by consumers, embedded tags in our personal devices, clothes, and groceries can unknowingly be triggered to reply with their ID and other information, potentially allowing for a fine-grained yet unobtrusive surveillance mechanism that would pervade large parts of our lives [72, 106].

An Auto-ID infrastructure should consider these consumer fears and the legal guidelines that apply for data collection. If Auto-ID communication protocols support dedicated privacy enhancing features [70], the Auto-ID infrastructure will also need to support their use. Requirements concerning privacy are not part of the scope of this thesis and are discussed in great detail in [72, 105, 106].

(R22) Other Application Requirements

Other application requirements, for example requirements that relate to security and scalability, are not discussed here in detail, since they are not unique to Auto-ID system design.

4. Auto-ID Infrastructure Concepts and Implementation

An Auto-ID infrastructure consists of *Auto-ID readers*, *sensors* and *actuators* that are deployed in an application context, for example, across the different locations of interest in a supply chain. The filtering and aggregation of the captured data (i.e. the Auto-ID tag observations and the measured physical properties of the sensors) and the management of the readers and sensors are the responsibilities of the *Filtering and Aggregation* layer. Different filter and aggregation concepts components [28, 53, 47, 13, 66, 67, 131] guarantee that only relevant data is provided to the higher layers resp. the applications. Figure 4-1 shows the different layers of an Auto-ID infrastructure including their relationships.

Filtering and Aggregation provides many concepts that facilitate the development of Auto-ID applications. However, the level of abstraction is still close to the concepts of Auto-ID readers and sensors. As stated in the system requirement (R1), applications are commonly not interested in Auto-ID specific details.

It is therefore important to provide appropriate abstractions and representations of Auto-ID and sensor specific details and offer the necessary services, support for simple application logic and the appropriate level of reuse to bring the development of Auto-ID applications on the next level. This would allow system and domain engineers to build Auto-ID applications and reduce the need for custom software development to a minimum.

In an overall architecture of an Auto-ID infrastructure we therefore propose the *Data Enrichment, Representation and Persistency* layer that provides application-level support with abstractions appropriate for many *Applications and Information Systems*. The responsibilities of this layer can be grouped into the following main categories (see Figure 4-1):

- Enriching Auto-ID data with business logic (e.g., location information or information about the state of a physical object)

- Providing adequate representation of data for an application (e.g. as physical objects with their relationships)
- Keeping a data history (i.e., persistency of data) including query capabilities which is important for track and trace applications
- Providing Auto-ID related business process support to allow fast processing of the data in a business workflow

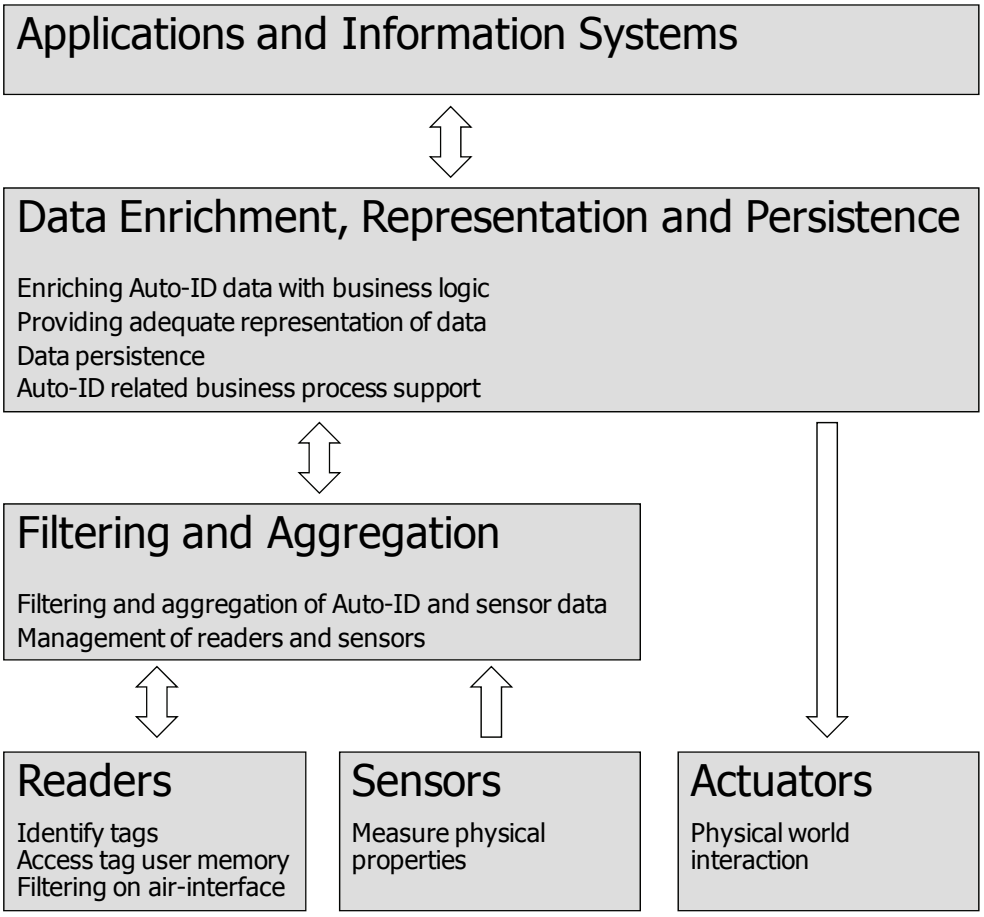


Figure 4-1 Layers of Auto-ID infrastructure

Since the Filtering and Aggregation layer is the base of the higher layers, the filtering and aggregation concepts are shortly presented in section 4.1. The concepts the Data Enrichment, Representation and Persistency layer is based on are the *Auto-ID Object Model* which provides an object representation of the physical objects in an application setting including their properties, containment relationships and the services related to it. The Auto-ID Object Model (see section 4.3) is based on a symbolic, hierarchical location model which is discussed in a short overview of location

models in section 4.1. Based on the Auto-ID Object Model, a state machine-based model to provide business process support is presented. Section 4.4 presents an overview and discussion of an implementation of the Data Enrichment, Representation and Persistency layer. The related work in the fields of Ubicomp Infrastructures and Auto-ID middleware is presented in section 4.5.

4.1. Filtering and Aggregation of Observations

Auto-ID readers and sensors produce an enormous amount of data that is highly redundant or not of interest to an application or higher layer. By applying appropriate filtering and aggregation of the data, the amount can be significantly reduced [38, 121, 23]. The following filtering and aggregation concepts have been developed in cooperation with C. Flörkemeier and a detailed description in conjunction with a proposal for an implementation can be found in [66].

For example, a palette with 10 packages of different products that arrives at a dock door that is observed by an RFID reader might reside in the read range for approximately 10 seconds. Since a typical UHF RFID reader has tag read rates of up to 1000 tags per seconds [88], the readers would generate about 100 observations per seconds of the packages. This would result in approximately 1000 observations during the 10 seconds in the read range. Since the application is only interested if the packages arrived, filtering out the palette tags and aggregating the tags of the packages to eliminate duplicate reads can reduce the observations to only a couple.

Table 4-1 Filter types

Filter by	Description
Reader Identifier	This filter type allows the application to specify that it is only interested in data from a particular set of readers.
Tag Identifier and Data	The application can define the tag population that it is interested in, e.g., the restriction to tags attached to pallets.

The removal of certain tag observations based on the reader which generated the observation and the tag data captured is usually referred to as filtering. Table 4-1 shows the most common filter types. Many readers

and RFID protocols support filtering that is carried out on the air interface for bandwidth considerations.

Aggregation is desired to reduce the flood of raw tag reads to more meaningful events such as the first appearance of a tag in the read range and its subsequent disappearance. Aggregation is also needed to address the problem of temporary false negative reads and to smooth the data accordingly. The aggregation types we propose are listed in Table 4-2.

Table 4-2 Aggregate types

Aggregate types	Description
Observed	Over a certain duration all duplicate reads of tags are eliminated by this aggregate type.
Entry & Exit	This aggregate type reduces a number of successful reads of a tag to the best estimate when the tag appeared and disappeared from the read range.
Count	Applications can prefer to receive information about the total number of items of a specific category detected rather than the individual ID of each object.
Passage	This observation indicates the direction, in which a tagged object is moved, as a tag moves from one reader to another. Applications prefer receiving a passage observation rather than being forced to interpret a sequence of entry and exit observations from two individual readers.
Logical Readers	When an application does not distinguish between two readers, this aggregate type allows it to logically join their read range.

Figure 4-2 gives an example for filtering and aggregation of Auto-ID data. In the example scenario, the dock door is monitored by two Auto-ID readers *Reader1* and *Reader2*. The EPCs and a timestamp of the detected objects are given. The observations of the two dock door readers are combined into observations of a logical reader *DockDoor*. Duplicate EPCs are eliminated by applying the Observed aggregation and the EPCs of palettes are filtered (in this case EPC 11.49.40). In addition, the quantities of product categories are calculated.

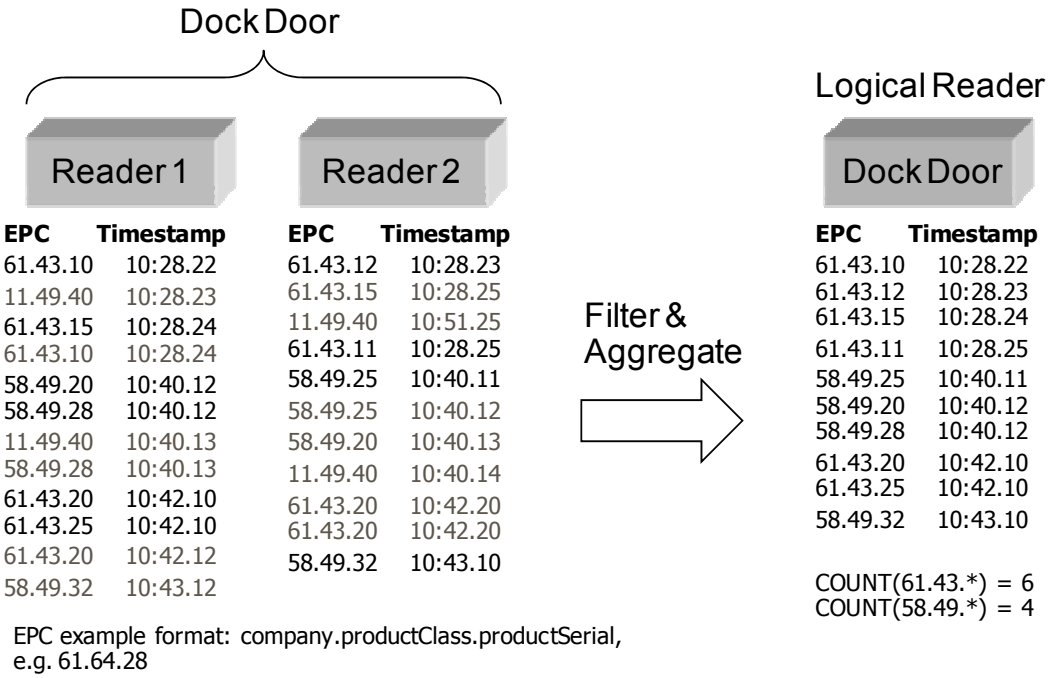


Figure 4-2 Example for filtering and aggregation of observations

4.2. Overview of Location Models

Since many Auto-ID applications are interested in the locations of physical objects, the concept of location of physical objects takes an important part in the Auto-ID Object Model. It is important to understand the different concepts of location models that exist to be able to better understand the decision to base the Auto-ID Object Model on a symbolic location model.

Location models provide the representation of locations of objects. The representation of spaces, locations and the objects they contain is very important to location aware systems and Auto-ID application domains. Location models can be classified in the following four categories: Geometric, symbolic, hybrid and semantic location models as described in the following sections.

4.2.1. Geometric Location Models

Geometric location models [107] contain points, areas and volumes represented as n dimensional tuples of coordinates. A point in a geometric space does not have the relationship to what it points to. A physical model

is a geometric model that specifies the position of places or points based on a global coordinate system [124]. A geometric model can have one or more reference coordinate systems. In this case the coordinate systems are not absolute but relative to some reference point. For example, to retrieve the distance information it is necessary to perform coordinate system transformations. The accuracy of the information of this model is high, but the information acquired has to be translated to one of the reference coordinate systems.

4.2.2. Symbolic Location Models

Symbolic location models [107] identify locations with human-readable names and abstractions such as room, building, city, etc. There are three kinds of the symbolic model: cell, zone and a location domain model. They differentiate in the permission to the overlapping of locations. This model lacks the precision but it models the containment relationships very well (e.g. building contains a room). This model requires the manual definition of the names used in the model. The set of names depends on the application domain and the spatial resolution of the model.

4.2.3. Hybrid Location Models

Hybrid location models [107] combine advantages of two previous models. They give the needed precision to the symbolic description (i.e. locations are represented with both coordinates and symbolic names). However, the mapping between a specific space and its symbolic representation has to be provided.

4.2.4. Semantic Location Models

Semantic location models [124, 75] focus on the relationships between spaces allowing location definitions that do not physically exist. For example, the department of computer science has offices in different buildings but from an organizational point of view they are part of the same department. There is a semantic connection between them even though they are not physically connected.

4.2.5. Summary and Discussion

One important requirement on a location model is its scalability. It has to be possible to add new locations into the location model that can monitor

their areas, as well as to enable monitoring of many mobile objects in a system. The location information can be stored in a tree, but sometimes it is important to model an object that has more than one parent (e.g. a door between two rooms has both rooms as parent). In that case the lattice structure is more appropriate to model the location data. A graph approach enables modeling of complex relationships which is also useful for semantic models to represent logically connections between locations.

The location models use two different ways to associate a located object with a location [44]: containment and positioning. Containment determines positions of objects by identifying regions which contains these objects. Symbolic models use containment. Positioning tracks objects by acquiring their coordinates in the reference coordinate system. Each object is represented by its current position and the relations between objects can be represented by distance between them. Positioning is used by physical and geometric location models.

The choice of a specific model depends on the application domain that should be supported [83], on the resolution needed by the application and on the accuracy and precision required. For example, if we need to compute distances between objects it is better to use a coordinate location model, since it is much easier to compute distances using coordinates.

4.3. Auto-ID Object Model

The objective of the proposed Auto-ID Object Model is to provide an abstraction from the low-level Auto-ID and sensor concepts (e.g., tag observations or tag memory) and the means for an application to model its domain as the base for the application logic. However, the Auto-ID Object Model does not intend to provide a world model that allows describing any potential application domain. The model focuses on the domain of Auto-ID applications, that is, applications whose application logic is based on an implicit or explicit model of the physical world and is triggered by (near) real time observations of the physical world through Auto-ID readers and sensors.

For a specific Auto-ID application, the model has to be instantiated, that is, the Auto-ID application creates an instance of the model using an implicit (e.g., hard-coded in the application) or explicit (e.g., file-based) description of the instantiation (see Figure 4-3). Such an instance of the model contains static information such as the locations of the application

domain and dynamic information that changes during the life-time of the application such as objects and object state information observed by Auto-ID readers and sensors.

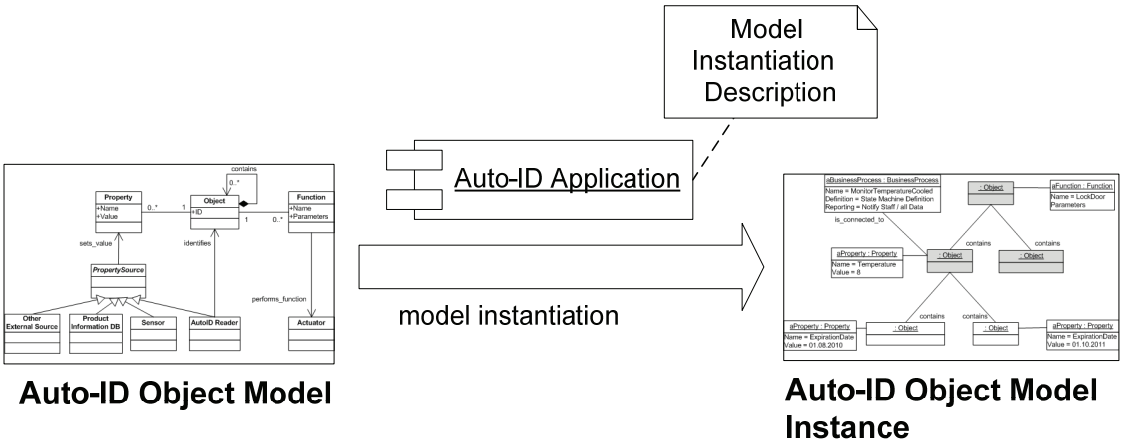


Figure 4-3 Auto-ID Object Model instantiation

The Auto-ID Object Model is based on a symbolic location model, in which physical objects also define locations (see Figure 4-4). The term *object* stands for the entity in the model that represents physical objects in the real world (e.g., the object Fork-Lifter12 in an object model instance represents the fork lifter #12 in the physical world). Auto-ID readers indentify physical objects by their Auto-ID tags. The representation of physical objects using a symbolic location model are discussed and described in detail in section 4.3.1.

Properties represent additional dynamic or static business information about objects (see section 4.3.2). Such information is received and related to an object in the model using property sources (e.g., product information databases, sensors or data on Auto-ID tags read by Auto-ID readers).

Functions (see section 4.3.3) are the abstraction for the interaction of the system with the physical world. The interaction is then performed by actuators such as locking a door or setting a signal light to red.

The entities in the presented Auto-ID Object Model are the concepts with which an application and the business process support are interfacing.

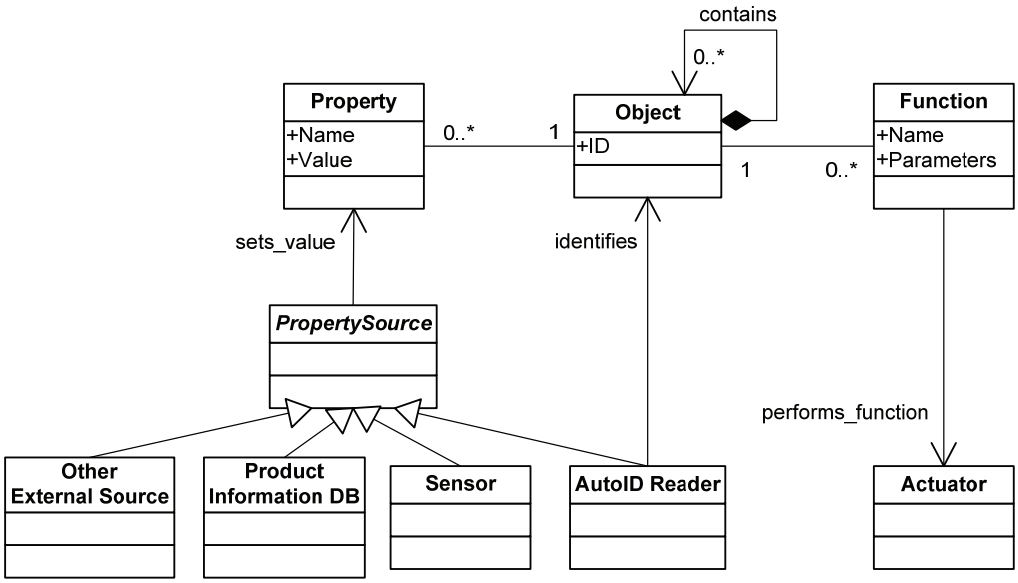


Figure 4-4 Overview of Auto-ID Object Model

4.3.1. Physical Object Representation

Physical objects of the real world play the central part in the model and are represented by the model entity *Object*. As a reference for the application an Object has an identifier that is unique for the application or application domain. A prominent candidate for such an identifier is the electronic product code (EPC) as specified by EPCglobal [50]. It provides a container for different number schemes that are extendable. Currently, the standard includes the following number schemes:

- Serialized General Trade Identifier (SGTIN)
- Serial Shipping Container Code (SSCC)
- Serialized Global Location Number (SGLN)
- Global Returnable Asset Identifier is (GRAI)
- Global Individual Asset Identifier (GIAI)
- Department of Defense (DoD) Universal ID

All EPC identifiers have a symbolic representation which can be used as the ID for objects in the Auto-ID Object Model. For example, a SGTIN EPC could look like `urn:epc:id:sgtin:0037000.112345.400`. It contains information about the producer, product class and the product serial number for a product which can be used by the application to count the number of products of a certain class in a shipment or filter on products of a specific producer. However, any other ID schema can be used by an application.

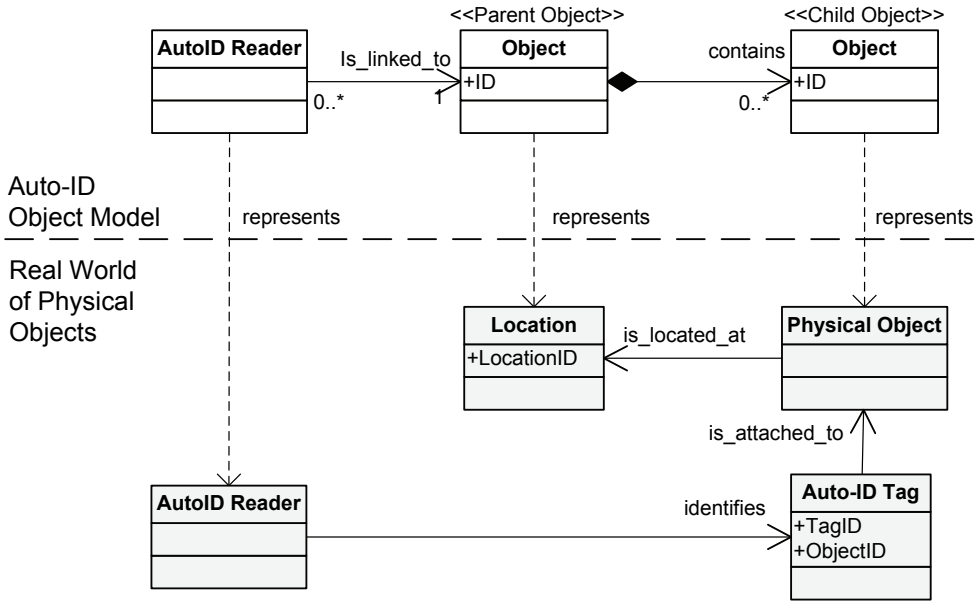


Figure 4-5 Auto-ID Object Model entities representing physical objects in the real world

Figure 4-5 presents a different look on the Auto-ID Object Model. On top of the striped line, the entities Object and Auto-ID Reader of the model are shown with their relationships. Below the striped line, the physical objects of the real world which the model entities represent are shown.

The Auto-ID Object Model does not differentiate between physical objects and locations. They are both represented by the model entity Object which is shown in Figure 4-5 by making parent and child Object types in the model explicit. The containment relationship between Objects implicitly expresses that a physical object is located at a location, that is, the containment relationship between a child and a parent Object can be also be read as “the child object is located at the parent object”.

For example, two bottles of water which are located in the cooled storage of the backroom of a retail store are represented in the model as shown in Figure 4-6: The parent Object Backroom contains the two child Objects CooledStorage and FrozenStorage. CooledStorage itself contains the two water bottles as child Objects (MineralWater1L.130 and MineralWater1L.230). Figure 4-7 gives a more simplified representation of a model instance using a floor plan of the different locations and physical objects and a tree view (the Auto-ID readers are denoted by green rhombus, the yellow triangle denotes sensors).

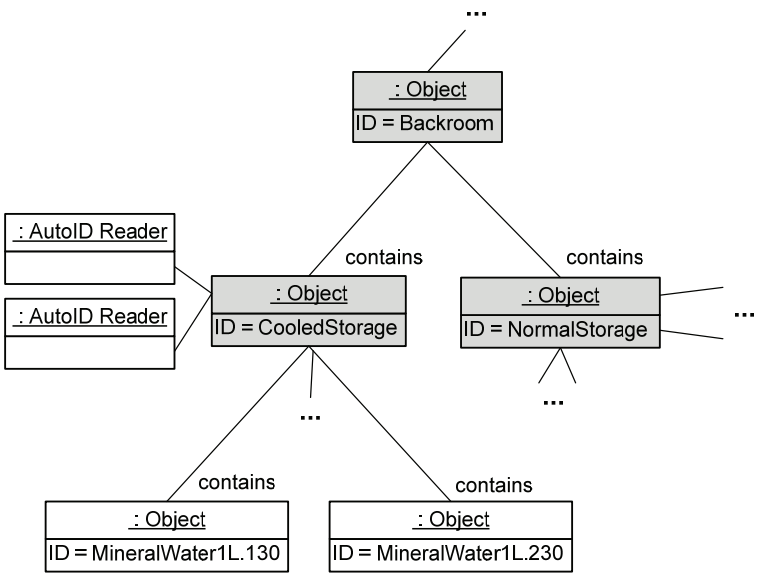


Figure 4-6 Example instance of the Auto-ID Object Model (UML object diagram representation)

As shown in Figure 4-6 and Figure 4-7, in an Auto-ID Object Model instance, the Objects and the containment relationships between all the Objects are represented as an object tree in which an Object can have zero to n child Objects. The Auto-ID Object Model is therefore based on a symbolic location model (see section 4.2.2). The reason for choosing a symbolic location model and representing physical objects and locations as Objects is the way Auto-ID readers monitor their environment.

An Auto-ID reader detects and identifies the Auto-ID tag that is attached to a physical object. Physical objects can be detected only in the read range of an Auto-ID reader and, except for specialized active RFID technology, a reader cannot determine the exact location of a physical object but only that it is contained in the read range. Auto-ID readers can therefore automatically determine a containment relationship if they are positioned in such a way that the read ranges of the readers cover the whole area of a location. The Auto-ID readers are linked in the model to the parent Object for which they determine its containment relationships.

Another possibility to monitor a location that has a limited number of entrances is to place readers at these entrances to determine if a physical object entered or left a location (see the placement of the readers on the floor plan in Figure 4-7). Such a multiple reader installation is often combined with a light barrier to be able to determine if a physical object just passed a reader or really entered or left a location. In the Filtering and

Aggregation layer such installations are specially processed and so called passage events are generated (see section 4.1).

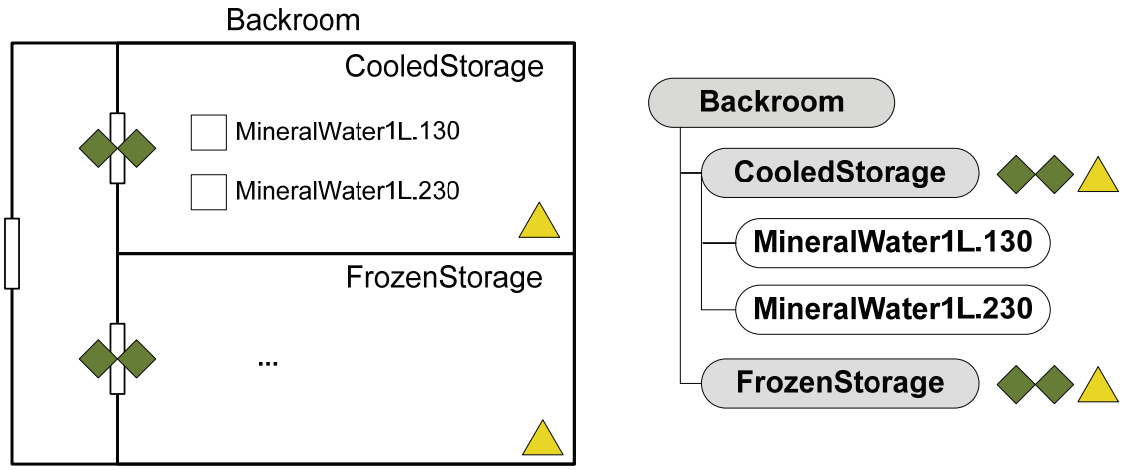


Figure 4-7 Example instance of the Auto-ID Object Model (simplified floor and tree representation)

Summarizing the above, the core of the Auto-ID Object Model is defined by:

- Physical objects and locations are represented by Objects.
- An Object has a unique identifier (ID) in the context of an application which is the symbol by which it is represented.
- An Object can contain other objects (i.e. the child objects). It may also be contained by one object (i.e. the parent object).
- An object may have a link to one or more Auto-ID readers, which automatically identify the child objects of the object to which the reader is linked (i.e. the reader determines the containment of objects).

The Auto-ID Object Model definition has the following implications:

- A location cannot exist without an object that defines the location, that is, the two concepts of object and location are merged into one. This greatly simplifies an instance of the model since the application only has to define and manage its objects. Many of these definitions happen automatically since objects carry their IDs which are obtained automatically by Auto-ID readers of the container object.
- The application has to define a strategy for automatically retrieving the IDs of Objects. The possibilities are: (a) using the TagID of the Auto-ID tag that is attached to the physical object, (b) using an Ob-

jectID stored on the Auto-ID tag (e.g. EPC), (c) providing a mapping table or algorithm for mapping TagIDs to ObjectIDs.

- The static structure of the object tree, i.e. the static objects such as buildings, rooms, shelves and their containment relationships have to be defined by the application.
- Links to Auto-ID readers have to be defined by the application.

Moreover, there are the following implicit Object types that can exist in an application domain:

- *Static Objects* such as buildings, rooms or loading areas that scarcely change over the lifetime of an application. If a building is remodeled or a new one is constructed the model entities have to be adapted accordingly.
- *Semi-static Objects* such as shelves, cool boxes, promotion sale areas that frequently change over the lifetime of an application. However, the change cannot be determined automatically and the model entities have to be adapted accordingly.
- *Mobile Objects* that defining a location such as fork lifters, transportation units, shopping carts. They are mobile in the physical world (i.e., they can easily change their location) and have Auto-ID readers linked to them to determine their contained objects.
- *Simple Objects* such as products that frequently move through the Object tree of the application domain.

Static and semi-static Objects have to be managed manually by adapting the model instance of the application accordingly. Simple Objects on the other hand are automatically managed by the model implementation. Mobile Objects have to be partly managed manually, for example, the Auto-ID reader link has to be set. However, the location change of the Mobile Object is managed by the model implementation. If a Mobile Object leaves the monitored locations of the Object tree of the application domain, the model implementation either has to cache the Mobile Object and its contained Objects or initiate an update of the containment relationship when the Mobile Object enters a monitored location again. Otherwise the information about the contained objects would be lost.

For example, an RFID reader mounted on a fork lifter monitors the palette and its content which the fork lifter is carrying. In the model instance, the detected Objects will be contained in the fork lifter Object. By itself the fork lifter might be detected by a passage reader of the backroom. In

the model instance, the fork lifter Object with its containing objects will be contained by the backroom Object.

In order for an application to define the static structure of the Object tree, the Auto-ID Object Model provides the following tree construction/ destruction operations to an application:

- `addObject(childId, parentId)` adds a new object with `childId` under the object with `parentId` in the object tree.
- `deleteObject(objectId)` removes an already defined object with `objectId` including all child objects from the object tree.
- `move(objectId, newParentId)` reconstruct the location tree. It moves the object with `objectId` and all child objects to the new parent object with `newParentId`.

To attach or remove links to Auto-ID readers the following operations are supported:

- `addReader(objectId, readerId)` which attaches a reader with `readerId` to the Object with `objectId`.
- `removeReader(objectId, readerId)` which detaches a reader with `readerId` from the Object with `objectId`.

4.3.2. Properties of Objects

A physical object may have properties that further describe the object and its state. These properties can be static, that is, they do not change over the lifetime of an object (e.g, expiry date or color), or dynamic if they frequently change their value (e.g., temperature or usage counts). Moreover, properties can belong to a single object (e.g. usage counts of a certain tool) or to a class or group of objects (e.g. volume or size of water bottles).

In the Auto-ID Object Model a *Property* of an *Object* simply consist of a name-value pair (see Figure 4-8). The Properties of an Object are in a common application setting distributed over different sources (e.g. sensors or tag memory). The concepts of Properties and *Property Sources* are introduced to abstract both from the different methods to access the values of a property and the different types of Properties (i.e. belonging to a single object or object class). For example, the access to a temperature sensor incorporated into an Auto-ID tag is quite different from accessing object class related information from a product database or ERP system. In the Auto-ID Object Model the access is always the same through a Property of an Object.

Figure 4-8 illustrates the different Property Sources and how the model entities represent the physical entities of the real world and Figure 4-9 presents an example model instance with several Properties attached to the Objects. In the simplified representation of a model instance, sensors are denoted by a yellow triangle (see Figure 4-7). The most common Property Sources are:

- *Auto-ID tag user memory*. Auto-ID tags such as RFID tags (active and passive) often contain user memory from several bytes to kilo bytes. The application can define which properties are stored for which object class on the user memory. Auto-ID readers read the user memory of interest that is on an Auto-ID tag attached to a physical object.
- *Sensors*. The values of environmental parameters that are measured by a sensor can be linked to a Property of an Object. The sensor can be connected to a location or a physical object.
- *Product Information Databases*. Global product databases such as SYNFOS [150] and 1SYNC [20] store all related information about products. Companies that produce these products or other partners in the supply chain can subscribe to get access to these databases. The information is typically static information about objects such as ingredients of a food product, color or the measures of a product.
- *Internal Property Storage*. Properties connected to an Object can be defined by the application at startup or during runtime. These properties can, for example, describe static Objects or add any other information needed by the business processes.
- *Other external sources*. Many other potential data sources in the network or the Internet exist where information about Objects can be retrieved. For example, Object information in legacy or ERP systems within the application domain, or information in public Internet databases such as book information at Amazon.com.

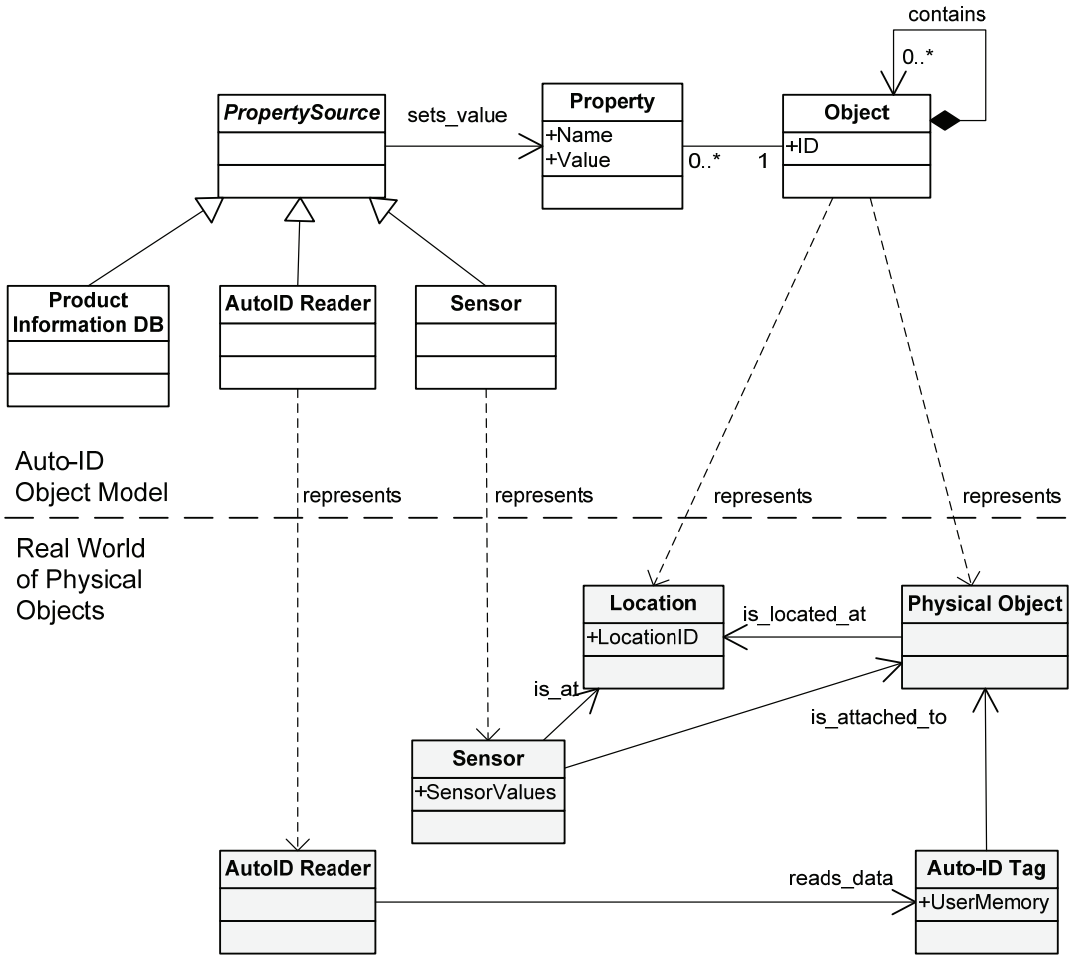


Figure 4-8 Auto-ID Object Model entities representing physical objects in the real world (emphasis on entity Property)

Summarizing the above, the concept of Property in the Auto-ID Object Model is defined by:

- A Property is a name-value pair.
- An Object can have zero to n Properties.
- Each Property has a Property Source that sets its value.
- A Property Source abstract from the method to access a property and can be Auto-ID tag memory, a sensor value, product information databases, and internal or external data sources.

The Property definition has the following implications:

- The application has to define the link between Property Sources, Properties and Objects. This can be based on Object classes or special groupings based on the Object ID.

- The application has to provide a definition of the Auto-ID tag user memory structure for the stored property values. Since the Filter and Aggregation layer typically allows naming user memory areas, the application has to simply define the link between name of the Property and name of the user memory.
- Internal Properties connected to an Object have to be defined by the application during instantiation of the model in order for the model implementation to allow access to such a Property.
- Connection and access information for external information sources have to be provided at instantiation time for the model implementation to access these sources.

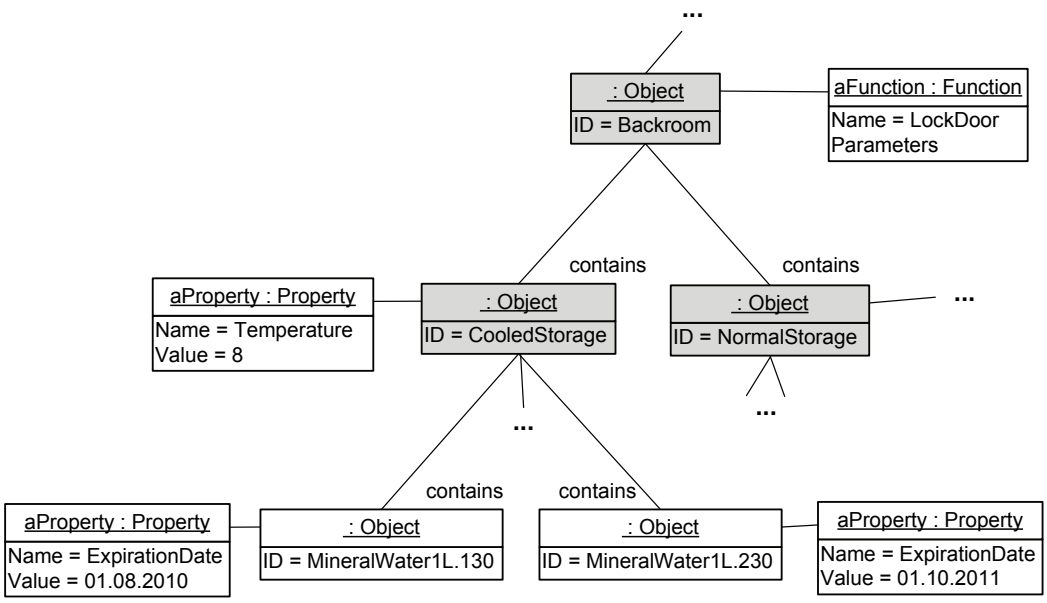


Figure 4-9 Example model instance with emphasis on Property and Function

4.3.3. Functions

Functions of Objects in the Auto-ID Object Model are defined by a name and a list of parameters that are interpreted by the Function (see Figure 4-10). They act as the abstraction for actuator control (e.g., locks or light signals) and can be executed by the application or by business processes of the model. Figure 4-9 presents an example model instance with a Function to lock the doors attached to the Object Backroom.

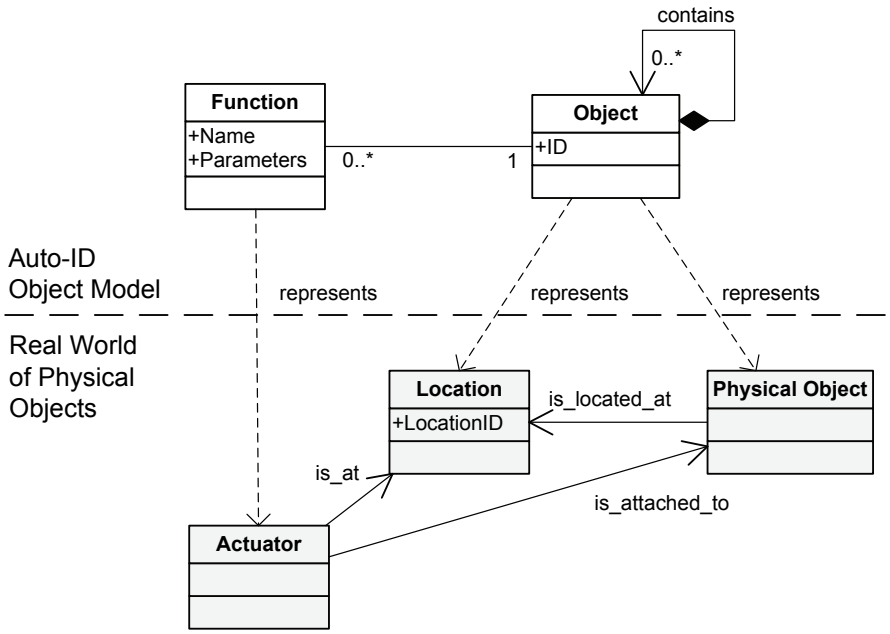


Figure 4-10 Auto-ID Object Model entities representing physical objects in the real world (emphasis on entity Function)

Functions therefore offer fast interaction with the physical world. For example, in a smart medical shelf application, a business process detects the badge of the person accessing it and only unlocks the doors via the lock actuator if the person is authorized. Actuators can either be attached to physical objects (e.g., lights, buzzer or sound) or to locations (e.g. door lock, light signal) depending on their usage.

Actuators can be used to interact with human beings in the real world or control processes where no human beings are involved. For example, a signal light that gets green if all products of a shipment arrived at a dock door and the fork lift driver is allowed to drive with the shipment into the storage area.

Summarizing the above, the concept of Functions in the Auto-ID Object Model is defined by:

- A Function has a name and takes a list of parameters.
- An Object can have zero to n Functions.
- A Function abstracts from physical actuators to allow interaction with the real world.

The Function definition has the following implications:

- The application has to define the link between Functions and Objects. This can be based on Object classes or special groupings based on the Object ID.
- Since Functions are specific for the application domain, the application also has to provide the implementation for the provided function in the Auto-ID Object Model implementation.
- The application has to provide the connection and access information for the actuators needed by the Functions.

4.3.4. Object History and Queries

The Auto-ID Object Model does not only provide a representation of the current states of physical objects, but also information that depends on time, i.e. we include time as an important concept in our model. Applications are usually interested in the following two history information about physical objects:

1. The history of a located-object, that is, all locations at which the physical object was located, e.g., a supply chain management application needs to know all places of interest at which a certain product has been to get a track and trace of the product.
2. The history of a location, that is, all physical objects that were located at the given location, e.g., an application monitoring chemicals needs to know for an audit which chemicals have been at a storage location over a certain time span in the past, to determine if any safety violation happened.

Figure 4-11 illustrates another example for the history of the located Object MineralWater1L.130 over a certain duration in the past: (1) First the Object arrives with a shipment at DockDoor-12, (2) next it is moved by a fork lifter (ForkLifter-12) into the Backroom, (3) and is stored in the CooledStorage, (4) from which it is finally moved to the CooledShelf-2. All parent Objects are shown until the root Object RetailStore.

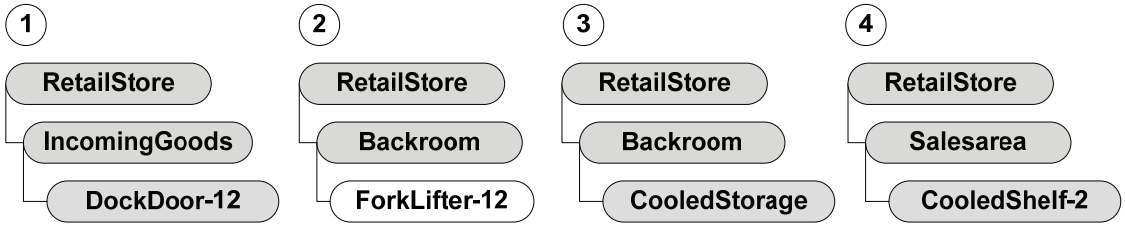


Figure 4-11 History of located object MineralWater1L.130

Figure 4-12 illustrates an example for the location history of the Object CooledStorage over a certain duration in the past: (1) two water bottles are contained in CooledStorage, (2) a fork lifter driver (Staff-2342311) drives with a fork lifter (ForkLifter-12) and four loaded water bottles into the CooledStorage, (3) he unloads the bottles and leaves the storage (omitted in the figure), and (4) the final state where the extra four bottles are contained together with the first two bottles in the CooledStorage.

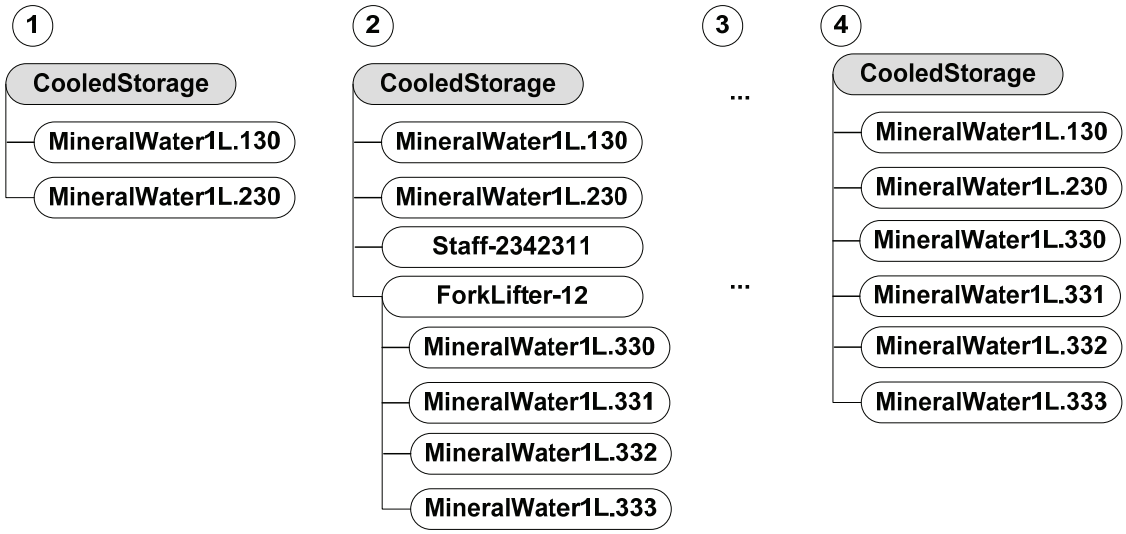


Figure 4-12 History of the location CooledStorage

The Auto-ID Object Model provides querying of location information of Objects. Since time is included in our location model the application can query not only the present location information but also changes that happened in the past. The application can make the following requests:

- Where is physical object X located at?
- Which physical objects are at location X?
- Where was the physical object X located at time T1?
- Where was the physical object X located during the time interval T1-T2?
- Which physical objects were located at location X at time T1?
- Which physical objects were located at location X during the time interval T1-T2?

These requests can be performed using the following two queries with their parameters:

1. `GetLocationsOfObject`. Retrieves all parent Objects for the Object in question (`objectID`) from the start time (`timestamp`) for

the given `duration` where `duration` can be zero for a point in time. An optional parameter `level`, specifies how many parent Object levels should be returned for the results (e.g., if all the levels to the root Object should be retrieved, `level` would be `all`).

2. `GetLocatedObjects`. Retrieves all child Objects for the parent Object in question (`objectID` from the start time (`timestamp`) for the given `duration` where `duration` can be `0` for a point in time. An optional parameter `level`, specifies how many child Object levels should be returned for the results (e.g., if only the direct children should be retrieved, `level` would be `1`).

The queries for the results shown in Figure 4-11 and Figure 4-12 could look like:

- `GetLocationsOfObject(objectID=MineralWater1L.130, timestamp=10.10.2009 06:30:00,duration=1d, level=all)`
- `GetLocatedObjects(objectID=CooledStorage, timestamp=10.10.2009 12:30:00,duration=30min, level=all)`

4.3.5. Business Process Support

The dynamic changes in the Auto-ID Object Model correspond to the changes in the physical world. Many applications are only interested in exceptional states related to Objects and their Properties such as the completion of an incoming shipment, reached expiry dates of products in a shelf, or the temperature inside a cool box that exceeds a threshold.

In most cases a business process that also needs business knowledge about the application domain can be defined to determine such exceptional states. Applications would need to implement such business processes that would involve querying the Auto-ID Object Model instance quite frequently. The disadvantages of such an approach are the following:

- High query traffic on the Auto-ID Object Model instance with the increased network traffic involved for the communication.
- First, system engineers have to define the business processes, and then software engineers have to implement them often re-implementing similar concepts over and over again.

We therefore propose a business process support combined with the Auto-ID Object Model. A business process can be defined using a state machine-based model with transition conditions and actions based on the

state of Objects in the Auto-ID Object Model instance. The application is only notified in cases of exceptional states.

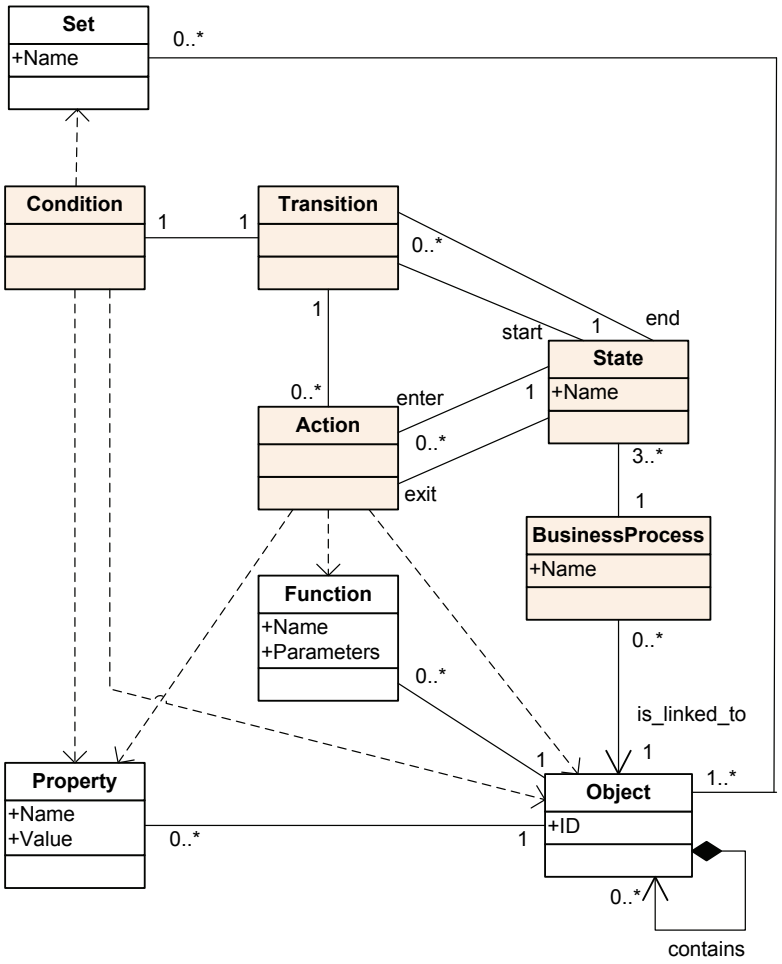


Figure 4-13 Auto-ID Model with emphasis on Business Process Support

Overview of Business Process Support

The analysis of Auto-ID applications has shown that an event mechanism based simply on conditions of Object and Property configurations is not sufficient to describe many exceptional states. The decision if an exceptional state occurred often depends on a related state that has happened a certain time before the state in question. This dependency of states over time can very well be expressed and modeled using state machines. In addition, state machines express a kind of business process flow that is well known in industry and state machines can be modeled using available design and modeling tools such as UML [118].

In the Auto-ID Object Model (see Figure 4-13), a business process is always linked to an Object representing a location or physical object in which the business process is interested. A business process comprises of several *States* (at least three, i.e., for start, end and the application defined state) which have *Transitions* between them. State transitions are coupled to certain configuration of Objects and their Properties. A transition happens if the *Condition* describing the transition is true. Conditions can involve presence or absence of a set or number of objects or object properties and time constraints. It is also possible to logically combine different conditions to describe a transition. The conditions are defined using an Auto-ID Object Model specific condition language that allows access to the model and provides several operators to formulate the logical conditions (see below).

Actions such as reporting to the application can be defined to take place in the following three cases: (i) when a transition is performed, (ii) when a state is entered, and (iii) when a state is exited. The following Actions can be used in the Auto-ID Object Model:

- *Model Action*. A model action provides a business process access to the Auto-ID Object Model instance, for example, to execute a Function, set the value of a Property or even allow changes in the Object tree.
- *Application Reporting*. The most common action of a business process is a report sent to the application. For example, after a shipment arrived a notification is sent to the application with information taken from the model instance such as the Objects of the shipment and the parent Object where it arrived.
- *Information System Reporting*. A business process of an Auto-ID Object Model instance often has to report information into an information system in an application domain. Many different systems exist such as ERP systems, databases, or application servers. Among others, the following information system reporting are supported:
    - *EPCIS Event Generation*. As an interface to EPC Network components such as the EPC Information Services (EPCIS) Repository, EPCIS events can be generated and stored in a given EPCIS Repository. From an EPC Network point of view, the implementation of the Auto-ID Object Model then acts as an EPCIS Capturing Application. The relationship be-

tween the Auto-ID Object Model and the EPC Network are discussed in detail in the related work section 4.5.
    - *SAP Connector*. As one of the most prominent ERP System, SAP comprises many modules where information about an application domain is stored. The application has to customize a connector to an SAP system to its needs.
    - *Custom Information System Reporting*. In many application domains other information systems, databases or legacy systems exists where information has to be stored. An application has to provide such custom components if they need to be used as actions in a business process.
- *Custom action*. In a highly distributed and connected application domain, many more actions can be imagined. The following list gives some examples for custom actions:
    - Reporting to external partners, for example, in a supply chain scenario, a retail store can be notified about an outgoing shipment via an advance shipping notice using EDIFACT [43].
    - Notification of workers, staff, etc. on their mobile devices (e.g. SMS to a retail store staff if the temperature in a cool box leaves the allowed limits).
    - Reporting using Web 2.0 technology such as RSS feed or Twitter.

A Business Process Example

Figure 4-14 illustrates an example of a state machine definition of the business process for an incoming shipment. In the example, the business process is linked to the Object IncomingGoods which represents the location of all incoming shipments. The application is interested if an expected shipment that arrived is complete (i.e., all products specified in an advance shipping notice were shipped), or incomplete. In the first case, the application can mark the shipment in the ERP system as done, in the second case, the application marks the shipment to be followed up and opens an issue for a service staff.

In the first state *Awaiting Shipment*, the process has a list of expected products for the shipment and awaits the start of the shipment. If the first products are detected (i.e., the Objects representing the products become child Objects of IncomingGoods) the next state *Shipment Arriving* is en-

tered and as an enter action a timer is started. When all products of the shipment have been detected (i.e., all Objects are child Objects of IncomingGoods) the state *Shipment Complete* is entered and the enter action is executed that notifies the application of the complete shipment. The process then ends.

If after 5 minutes not all products have been detected the state *Shipment Potentially Incomplete* is entered and the enter action is executed that notifies a local staff on his mobile device to check what the state of the shipment is. If new products of the shipment are detected the state *Shipment Arriving* is entered again, otherwise if after 10 minutes not all products have been detected the state *Shipment Incomplete* is entered and the enter action is executed that notifies the application of the incomplete shipment. The process then ends. Further more detailed examples are presented with the case studies in section 6.

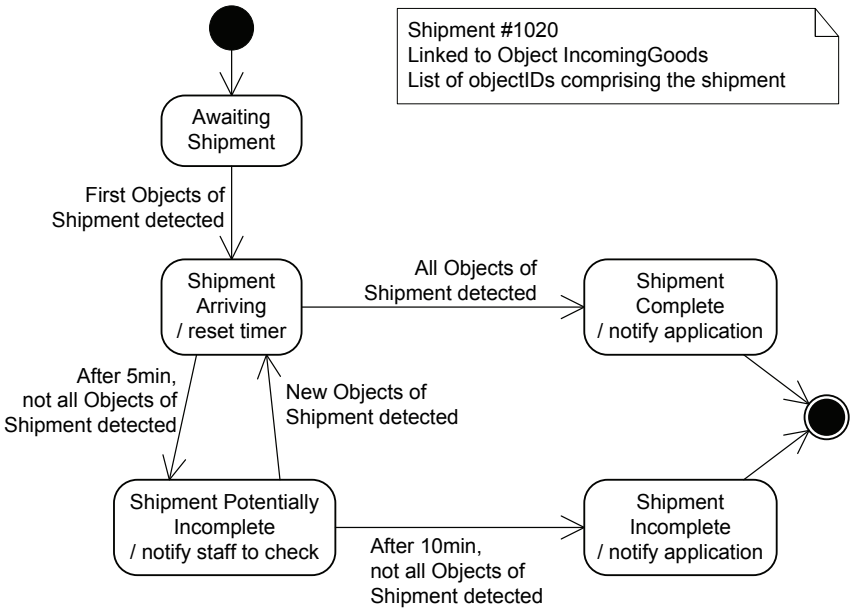


Figure 4-14 State machine to monitor the incoming shipment #1020

Formal Definition of State Transition Conditions

The conditions in the example above are described informally to illustrate the overall process. The business process support of the Auto-ID Object Model, however, provides a domain specific language to formally define the conditions that must be true for a transition between two states. The transition condition definition language consists of the following two parts:

1. A definition language for sets of Objects based on their IDs and Properties. These sets can be defined at instantiation time or also dynamically by the application. A formal definition in EBNF is given in Appendix 8.1 and Appendix 8.2.
2. A definition language for conditions that is based on set comparison of predefined sets (using the set definition language) and sets of Objects in the Object tree of the Auto-ID Object Model instance (using query operators of the model). A formal definition in EBNF is given in Appendix 9.3

Defining Sets

There are three different types of sets in that can be defined using the set definition language:

a) *Object Set* that is defined by enumerating all IDs of the Objects which comprise the set, for example, all the IDs of the staff of a retail store. The enumeration can be simply a list of IDs or could be generated automatically using a database table or query.
b) *Pattern Set* that consists of Objects whose IDs match a pattern defined for the given set. The widely used and available Perl regular expressions have been chosen to define the pattern. Pattern sets are of importance in application domains where IDs of Objects are structured, for example, using the EPC a pattern set can be used to define a set of Objects of a certain producer.
c) *Property Set* that comprises Objects whose Properties match given property conditions, for example, a set of Objects whose expiry date has been passed. Property sets can be defined by giving a condition for the property values using standard comparison operators or by the condition for the existence of a specific property.

For example, the object set definition of the staff of a retail store that is allowed to drive a fork lifter named `ForkLifterDrivers` would be:

```
OBJECTS 'Staff-2342110', 'Staff-2341512',
'Staff-2342220', 'Staff-2342311', 'Staff-2342488'
```

The pattern set definition of all one liter mineral water bottles (of the examples above) named `MineralWater1L` would be:

```
PATTERN '/^[MineralWater1L]/'
```

The property set definition of all Objects whose expiry date passed (i.e., whose Property expiryDate is less than today, taken that today is 10.10.2009) named `ExpiredProducts` is:

```
PROPERTY expiryDate LT 10.10.2009
```

The Auto-ID Object Model provides the following two set operations:

- addSet(SetName, SetDefinition) to add a new set defined in the SetDefinition.
- removeSet(SetName) to remove the set with the given name.

Defining State Transition Conditions

The definition of a transition condition for states is based on the comparison of a predefined set as described above with a set of Objects in the Object tree (i.e., currently observed physical objects). In addition, count operations on certain sets can be performed and conditions based on the properties of the Object to which the business process is linked can be defined. A state transition condition can be construction by logically combining different condition expressions using parenthesis and the logical operators `AND`, `OR` and `NOT`. Table 4-4 gives an overview of the definitions for the state transition conditions.

1. Set Comparison

The set comparison expression is defined in two ways, where the first set is specified either by the level of containment relative to the Object to which the business process is linked or by a set built by combining a containment set `Set1` and a predefined set `Set2`:

```
Set1 SetComareOp Set3
(Set1 SetOp Set2) SetComareOp Set3
```

where the set comparison operator *SetCompareOp* can be `ISEQUALSET`, `ISINTERSECTION`, `ISSUBSET`, `ISSUPERSET` or `ISEMPTYSET`. For the last operator, the second set `Set3` is omitted. They are defined as:

- `SetA ISEQUALSET SetB` is true, iff `SetA` is equal to `SetB` (i.e. they contain exactly the same Objects).
- `SetA ISINTERSECTION SetB` is true, iff `SetA` and `SetB` have at least one common Object.
- `SetA ISSUBSET SetB` is true, iff all objects from `SetA` are contained in the `SetB`.
- `SetA ISSUPERSET SetB` is true, iff all objects from `SetB` are contained in the `SetA`.
- `SetA ISEMPTYSET` is true, iff `SetB` contains no objects.

All of the set operators, except `ISEMPTYSET`, operate with object sets as a second parameter, whereas pattern and property sets can be used only with `ISINTERSECTION` and `ISSUBSET`.

`Set1` is the set of currently observed Objects which is dynamically created. It represents the current containment of the Object to which the business process is linked. The set can be specified by indicating the level of containment:

- `CHILDREN`. Only the direct child Objects comprise the set.
- `LEVEL n` (where `n` is a natural number > 1). All Objects that are included in the `n`th level in the Object sub tree hierarchy comprise the set. `Level 1` is equivalent to `CHILDREN`. If `n` exceeds the actual level of the Object sub tree, the set is empty.
- `LAST`. All Objects that are included in the last level in the Object sub tree hierarchy comprise the set.
- `LEVEL n-m` (where `n`, `m` are natural numbers > 1 and m > n). All child Objects from and including the `n`th level up to and including the `m`th level in the Object sub tree hierarchy comprise the set. If `n` exceeds the actual level of the Object sub tree, the set is empty. If `m` exceeds the actual level of the Object sub tree, the Objects are taken up to the last level.
- `ALL`. The Objects of the entire Object sub tree comprise the set.

The first set in the set comparison expression can be built using the set operators `INTERSECTION` or `COMPLEMENT`. `INTERSECTION` defines a set that contains the objects that are members in set `Set1` and `Set2`. `COMPLEMENT` defines a set that contains the objects that are members of `Set1` but not of `Set2`.

For example, in the business process that is linked to the Object IncomingGoods (see Figure 4-14), the following set comparison expression can be used to check if all Objects of the expected shipment (i.e. the set `ObjectsShipment1020`) are located at IncomingGoods:

```
ALL ISSUPERSET ObjectsShipment1020
```

The comparison operator `ISSUPERSET` is used and not `ISEQUALSET` since other objects such as fork lifters, staff or other shipments can be also located at IncomingGoods.

Another set comparison expression allows checking if a set contains a certain Object. The set in question is specified by the level of containment relative to the Object to which the business process is linked:

```
Set1 HASELEMENT ObjectID
```

where `Set1` is defined as described above.

2. Count Comparison

The count comparison is used to compare the number of Objects in a set with a given natural number. A count comparison expressions can be defined in the following two ways:

```
COUNT (Set1) CompareOp number
COUNT (Set1 SetOp Set2) CompareOp number
```

The set `Set1` is either a set specified by the level of containment relative to the Object to which the business process is linked (i.e. `CHILDREN`, `LEVEL` *n*, `ALL`) or a set built by combining a containment set `Set1` and a predefined set `Set2` using the set operators `INTERSECTION`, `COMPLEMENT` or `SUBSET`. `INTERSECTION` defines a set that contains the objects that are members in set `Set1` and `Set2`. `COMPLEMENT` defines a set that contains the objects that are members of `Set1` but not of `Set2`. With `SUBSET` we can check if `Set1` is a subset of the `Set2`, in which case the `COUNT` operator returns the size of the `Set1`. The result of the count operations has to be compared to a given number.

For example, in a business process that is linked to the Object Backroom (see Figure 4-7), it is required to test if the number of fork lift drivers exceeds 4 which would result in a warning to the drivers. The count comparison expression would be:

```
COUNT (ALL INTERSECTION ForkLifterDrivers) > 4
```

By building the intersection of all Objects in the sub tree below Backroom with the set `ForkLifterDrivers` only the fork lift driver Objects that are actually contained in the Backroom can be counted, all other Objects are not taken into consideration.

3. Property Comparison

The property comparison allows comparing the current value of a Property of the Object to which the business process is linked with a given value. The property comparison expressions is defined in the following two ways:

```
PROPERTY PropertyName CompareOp fnum
PROPERTY PropertyName = string
```

In the first definition the Property with the name PN can be compared with a floating point number. For Properties that are not represented by numbers, the second definition allows testing for equality with a given string.

For example, if the business process is linked to the Object CooledStorage (see Figure 4-7) the following property comparison expression can

be used to test if the Property of CooledStorage with the name Temperature has a value that is higher than 10 degrees Celsius:

```
PROPERTY Temperature > 10.0
```

Our location model provides the following operations to manage business processes:

- addBusinessProcess(ObjectId , businessProcessName, businessProcessDefinition)
- removeBusinessProcess(ObjectId , businessProcessName)

where businessProcessDefinition includes a list of states with their actions and a list of state transitions with their conditions and actions as shown in see Figure 4-13 (colored entities belong to a business process).

Detailed Definition of Business Process Example

The formal definitions of the state transition conditions given in the description of the example above (i.e. Monitoring Incoming Shipment #1020) are shown in Table 4-3. Note that the second transition defines in its model action a new temporary set with the objects of shipment #1020 that have not yet been detected named `MissingObjectsShipment1020`.

Table 4-3 Formal state transition conditions of example process

Transition condition	Definition of condition
First Objects of Shipment detected	`ALL ISINTERSECTION ObjectsShipment1020`
After 5min, not all Objects of Shipment detected	`TIMER GE 0:05.00` `NOT(ALL ISSUPERSET ObjectsShipment1020)`
New Objects of Shipment detected	`ALL INTERSECTION MissingObjectsShipment1020`
After 10min, not all Objects of Shipment detected	`TIMER GE 0:10.00` `NOT(ALL ISSUPERSET ObjectsShipment1020)`
All Objects of Shipment detected	`ALL ISSUPERSET ObjectsShipment1020`

Section 6 presents several detailed examples of business process definitions as proof-of-concept of the presented business process support in the Auto-ID Object Model.

Table 4-4 Overview of definition language for state transition condition

Expression	Description
`StateTransitionCondition`	Expression that consists of a number of `ConditionExpression` that can be logically combined using `AND`, `OR`, `NOT` and parenthesis.
`ConditionExpression`	Condition expressions can be set, count or property comparison expressions.
`Set1` *`SetComareOp`* `Set3` with *`SetComareOp`* as `ISEQUALSET`, `ISINTERSECTION`, `ISSUBSET`, `ISSUPERSET`, `ISEMPTYSET` and `Set1` as `CHILDREN`, `LEVEL` *`n-m`*, `ALL` and `Set3` is Object, Pattern or Property set	Set comparison expression that compares two sets of Objects.
(`Set1` *SetOp* `Set2)` *`SetComareOp`* `Set3` with *SetOp* as `INTERSECTION`, `COMPLEMENT` and *SetCompareOp*, `Set1`, `Set3` as given above	Set comparison expression that compares two sets of Objects where the first set is constructed.
`Set1 HASELEMENT ObjectID` with `Set1`, as given above	Set comparison to check if a set contains a certain Object.
`COUNT (Set1)` *CompareOp* `num` with *ComareOp* as >, <, =, >=, <=	Count comparison expression which compares the count of Objects in the set to a natural number.
`COUNT (Set1` *SetOp* `Set2)` *CompareOp* `num` with *SetOp* as `INTERSECTION`, `COMPLEMENT`, `SUBSET` and *ComareOp* as >, <, =, >=, <=	Count comparison expression which compares the count of Objects in the constructed set to a natural number.
`PROPERTY PropertyName` *CompareOp* `fnum` with *ComareOp* as >, <, =, >=, <=	Property comparison expression that compares the value of a property to a given floating point num-

Expression	Description
	ber.
`PROPERTY PropertyName = string`	Property comparison expression that tests the value of a property to equality with a given string.

4.4. Auto-ID Infrastructure Implementation

The concepts presented in the previous section, i.e. the Auto-ID Object Model, have been prototypically implemented as a software component called the *Object Monitoring System (OMS)*. The overall design and implementation are briefly described in this section. We also discuss possible implementation approaches of the Auto-ID Object Model including temporal databases, tree database extensions and active databases. Moreover, an outlook is given on a distributed implementation of the model.

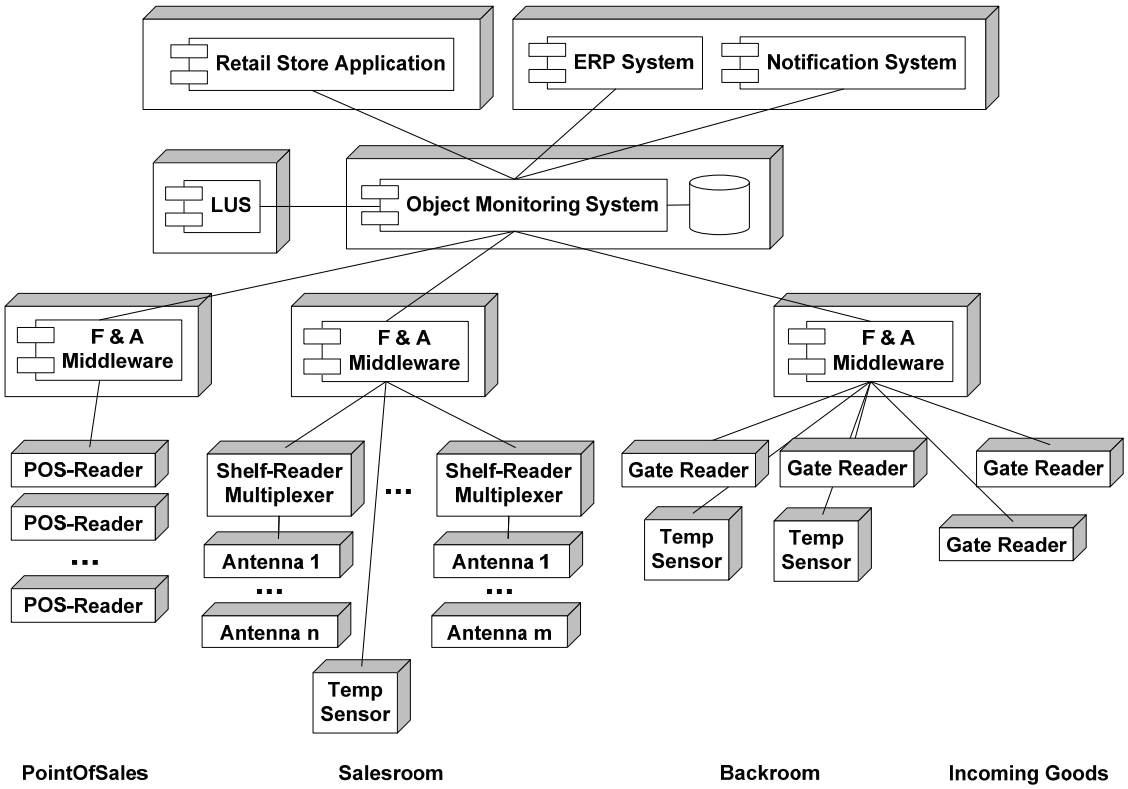


Figure 4-15 Auto-ID Infrastructure deployment for retail store example

The OMS implementation relies on the following components of other layers of the Auto-ID Infrastructure (see Figure 4-1):

- *Filtering & Aggregation (F & A) Middleware* that implements the concepts described in section 4.1 and that provides adequate interfaces to the OMS such as a publish/subscribe mechanism to retrieve the filtered and aggregated Auto-ID observations. In addition, interfaces to the lower layers allow retrieving data from readers and sensors.
- *Auto-ID Readers* that either get polled by or push their data to the F & A Middleware. A reader can have 1 to N antennas connected to it which are managed by an integrated or attached multiplexer.

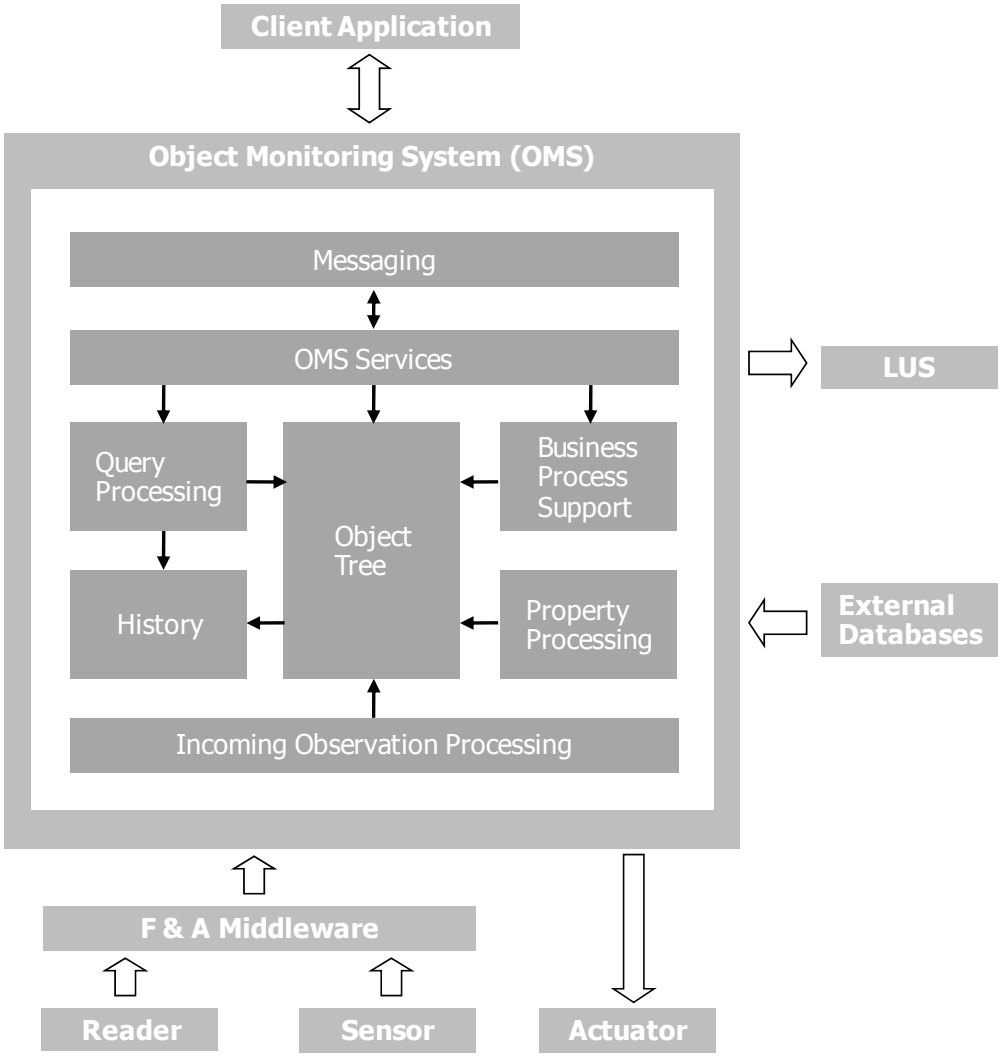


Figure 4-16 OMS Architecture Overview

- *Sensors* that observer environmental parameters which are retrieved by the F & A Middleware.
- *Look-Up Service (LUS)* that acts as a directory to dynamically retrieve connection information of existing components of the Auto-ID Infrastructure. Each component has to register itself using a unique ID after starting up (i.e., each component of the Auto-ID Infrastructure can be addressed with its ID).

The OMS itself provides services to the application layer, e.g., to ERP systems or application domain specific software. For example, Figure 4-15 illustrates an Auto-ID Infrastructure deployment for a retail store. In the example, several instances of the F & A Middleware balance the load of data and processing and provide the filtered and aggregated data to the OMS. Different business processes of the OMS send notification and data to a retail store application that provides a management cockpit for the store manager, to an ERP system, and to a notification system that relays notifications to mobile devices of the retail store staff.

4.4.1. Object Monitoring System

We prototypically implemented the Auto-ID Object Model and the business process support in a software component called the OMS which provides persistency of an Auto-ID Object Model instance (i.e. Object history), query capabilities and services to access and modify the Auto-ID Object Model instance (i.e. the Object tree) and the business processes. The OMS is implemented using the Java Platform, Enterprise Edition (Java EE) [149]. Figure 4-16 gives an overview of the overall architecture of the OMS including its subsystems. Important components and OMS subsystems are described in the following sections.

F & A Middleware

The OMS implementation supports the following Filtering and Aggregation Middlewares:

- The RFID Stack Event Layer (EL) [119] is a filtering and aggregation software developed in conjunction with OMS. The EL supports all the filtering and aggregation concepts described in section 4.1. It provides a flexible framework for building chains of processors (i.e. filters or aggregators) that allow for a high degree of flexibility. In addition, it supports processing of sensor data. Since there are no software interface standard for sensors, sensor adaptors have to be

developed for all different types of sensors that are distributed in an Auto-ID Infrastructure.

- EPC Filtering and Collection Middleware using the ALE interface standard [47] by EPCglobal supports not all of the filtering and aggregation concepts described in section 4.1: Count and Passage aggregations are not supported. Moreover, since it is aimed only at RFID infrastructures, sensors are not supported. Some of the application scenarios implemented with the OMS make use of the Fosstrak Filtering and Collection middleware [69].

An Auto-ID infrastructure deployment for a specific application setup might therefore use a mix of the two supported Filtering and Aggregation Middlewares.

LUS

The LUS is typically a directory server such as LDAP. The prototypical implementation of OMS uses an instance of OpenLDAP [120] where all software components have to be registered at application setup or components such as the RFID Stack Event Layer (see above) can register themselves dynamically. The OMS use the `javax.naming.directory` package to access the OpenLDAP instance. For each software component the following information has to be stored in a LDAP record:

- ID of the software component that acts as the key in LDAP (e.g. POS_EL for the RFID Stack Event Layer that manages all point of sales readers)
- Connection information (URL=tcp://10.0.128.20:6000) that contains all the necessary information for OMS to connect to such a component.

In the Auto-ID Object Model, Auto-ID readers are linked to Objects using an ID for a reader. Since the OMS does not connect directly to readers it needs to know the mapping between a reader and the Filtering and Aggregation Middleware that manages the reader. This mapping information is also stored in LDAP since it can change dynamically.

Incoming Observation Processing

All the incoming data from the readers and sensors via the Filtering and Aggregation Middlewares is stored in a queue that is ordered by the time of the incoming event. It is assumed that all components in the Auto-ID Infrastructure have the correct global time. This can be achieved by a dis-

tributed time synchronization algorithms such as the Network Time Protocol (NTP) [113, 114]. For example, a network connected from a sensor might be slow which leads to events that reach the OMS later than reader observations even if they happened earlier in time. Ordering the events by their time stamp provides a best effort to process the events in the order the occurred in the physical world. The time window of the processing of the events in the queue can be adjusted depending if the application should be closer to real time or if network latencies should be smoothed.

Auto-ID reader observations contain the following type of information:

- Timestamp
- ID of the reader
- Type of Observation (i.e. entry or exit)
- List of IDs of the observed physical objects resp. tags (e.g. tag IDs or EPCs)

If the IDs of the physical objects are already object IDs such as EPCs, they can be directly mapped to Objects in the Object tree. Otherwise they have to be translated from tag IDs to object IDs using a mapping table or rules for the specific application domain. Such mapping tables have to be provided as part of the OMS instantiation/configuration. The reader ID defines the parent Object of the observed Objects under which they are added as child Objects. For example, the following observation results in the two new child Objects MineralWater1L.130 and MineralWater1L.230 under the Object CooledStorage to which the reader GateReader-2 is linked:

```
10.10.2009 12:30:00, GateReader-2, ENTRY,
[MineralWater1L.130, MineralWater1L.230]
```

Sensor observations and Auto-ID tag memory reads are delegated to the property processing module of the OMS.

Object Tree

The Object tree is the core of the OMS. It is a tree data structure that can also be described as an instance of the Auto-ID Object Model which contains all Objects and their containment relationships that represent the current state of the physical world. All the described entities, their relationships and services of Auto-ID Object Model are implemented in the Object tree. The tree provides services to add, move and remove Objects resp. Object sub trees, and to add and remove a link to a reader or sensor.

The whole Object tree is kept in an in-memory data structure to provide fast access (i) for adding Objects by the processing of incoming observations, (ii) for the evaluation of state transition conditions, and (iii) for queries of the current state of the Objects.

Property Processing

Sensor observations and Auto-ID tag memory reads are processed specially since they result in the change of property values of Objects in the Object tree. In addition, properties that are retrieved by external databases are also handled in the property processing module.

A sensor observation contains the following type of information:

- Timestamp
- ID of the sensor
- Pair of sensor property and value

Using the ID of the sensor, the corresponding Object in the Object tree to which the sensor is connected can be found. The processing of sensor observations is then handled by the corresponding instance of the property source that manages the value change of the property of the Object including the mapping of sensor property to Object property. For example, the following sensor observation results in the change of the property Temperature of the Object CooledStorage to which the sensor tempSense-15 is linked to 10.2 degrees Celsius:

```
10.10.2009 13:30:00, TempSense-15, temp=10.2
```

An Auto-ID tag memory read contains the following information:

- Timestamp
- ID of the reader
- ID of the physical object resp. tag from which memory was read
- List of IDs with field name-value pairs that represent memory sections of the read tag

Using the reader ID and tag ID resp. Object ID, the Object in the Object tree to which the property belongs can be found. The processing of sensor observations is then handled by the corresponding instance of the property source that manages the value change of the property of the Object including the mapping of fields to Object properties. Tags used in applications to store fields on them have been either EPC Gen2 transponder (i.e. UHF tags) or Philips I-Code transponder (i.e. HF tags that offer 46 bytes of user memory). For example, the following tag memory read results in

the change of the properties ExpiryDate and Weight of the Object MineralWater1L.130:

```
10.10.2009 12:30:00, ShelfReader-10,
MineralWater1L.130, [ExpiryDate=12.10.2009,
Weight=1]
```

In the current implementation only SQL databases that are accessed using JDBC connections are supported to retrieved property values from external databases. In the OMS configuration, the application has to specify for which type or class of Objects a certain mapping of a property to an attribute in a SQL database applies. In addition, the polling frequency has to be defined for refreshing a property value.

History and Query Processing

The object history contains all past object data including properties and object containment relationships. Since the emphasis of the OMS prototype was not a performant implementation of the history, it is implemented by simply using relational database tables similar to the approach of [156] which are stored in the open source RDBMS MySQL database. OMS uses the following two tables to store the history of properties and objects (see Figure 8-1 Appendix 8.4 for the SQL DDS):

- Table `property_history`:
  `(ObjectID, PropertyName, PropertyValue, Timestamp)`
- Table `object_history`:
  `(ParentObjectID, ObjectID, EnterTime, ExitTime)`

The MySQL database accessed is wrapped by a HistoryManager that provides services to update, remove and query the database. When the incoming observation processing handles an Auto-ID reader observation, in addition to updating the Object tree, the corresponding record in the database is update. Property value changes are handled equivalently.

The HistoryManager also allows querying the history of objects and properties. The queries are translated into SQL queries and are executed by the HistoryManager. The application has the option to wait for the query results or get the query result through a notification channel specified by the application for larger more time consuming queries. The history allows querying the database using two kinds of queries that can be parameterized (see also section 4.3.4):

- *LocationsOfObject* query that gets all locations of a given Object (i.e. all the parent Objects) for a certain time range.

- *LocatedObjects* that gets all Objects situated at a given location (i.e. the child Objects) for a certain time range.

Since most queries are interested in a structured result in the form of Object subtrees, the results for both queries consists of recursive SQL queries.

For the LocationsOfObject query, the SQL base query (see Figure 4-17) is executed first to retrieve all the parent objects where the object in question was located. For each parent object the query is executed again with a narrowed down time range and so on for their parent object until the specified level or the root object is reached.

```
SELECT LocationID, EnterTime, ExitTime
FROM object_history
WHERE ObjectID = 'objectID' AND
 ((ExitTime >= 'timeFrom'
   AND EnterTime <= 'timeTo')
  OR EnterTime <= 'timeTo')
```

Figure 4-17 SQL base query for `LocationsOfObject` query

Similarly for the LocatedObjects query, the SQL base query (see Figure 4-18) is executed first to retrieve all the child objects that have been located at the location object in question. For each child object the query is then executed again with a narrowed down time range and so on for their child objects until the specified level or a leaf object is reached.

```
SELECT ObjectID, EnterTime, ExitTime
FROM object_history
WHERE LocationID = 'locationID' AND
 ((ExitTime >= 'timeFrom'
   AND EnterTime <= 'timeTo')
  OR EnterTime <= 'timeTo')
```

Figure 4-18 SQL base query for `LocatedObjects` query

Business Process Support

The implementation of the business processes support in the OMS consists of the following functions:

- Processing the definitions of business processes and activating the state machine of a business process
- Managing the state transitions of all state machines

- Executing the actions of state transitions or states

Business processes in the OMS are defined using XML. For the business process example presented in section 4.3.5 and illustrated in Figure 4-14 and Table 4-3, a snippet of the definition is (the complete definition is given in Appendix 8.3):

```
<business-process-definition>
  <states>
    <state name="START"/>
    <state name="Awaiting Shipment"/>
    [..]
    <state name="Shipment Incomplete">
      <action type="enter"
              kind="report"
              client="wms-app"
              objects="ALL INTERSECTION
                           ObjectsShipment1020"/>
    </state>
    [..]
  </states>
  <transitions>
    [..]
    <transition start="Shipment Arriving"
                end="Shipment Complete">
      <condition>
         CHILDREN ISSUPERSET ObjectsShipment1020
      </condition>
    </transition>
    [..]
  </transitions>
</business-process-definition>
```

The state machines are implemented based on a generic Java finite state machine framework. After processing the definition of the business process and its state machine, a new state machine with all states, transitions and actions is instantiated and initialized into the start state. The set and transition condition definition language is defined and parsed using JFlex [95], a Lexical Analyser Generator for Java and CUP [87], a LALR Parser Generator for Java.

The state machine management mainly consists of evaluating the state transition conditions. For all possible state transitions of all state machines, so called hints are set in the Object tree that signal the condition evaluator that a certain part of a condition has changed and needs to be re-

evaluated. Thus, only expressions where the underlying data has changed are evaluated which increases the performance of the state machines.

If a certain state transition takes place, all the actions involved are executed. Actions in the OMS are modeled as a framework and can therefore be easily extended (i.e. client application can implement new subclasses of an abstract class `Action` and load this class into OMS at runtime). The following actions have been implemented in the OMS: Model Action, Application Reporting and EPCIS Event Generation (see also section 4.3.5).

OMS Services and Messaging

The OMS Services are the collection of all services the Auto-ID Object Model provides which are described in section 4.3. Summarizing it consists of services to

- access and modify the Object tree and link Auto-ID readers and sensors to objects
- define properties and property sources
- define functions and link actuators
- query the history
- set and remove business process definitions
- set or change OMS properties and configuration (e.g. tagID-objectID mapping tables or OMS specific runtime parameters)

Client applications can access these services through the messaging layer of the OMS which manages all the connections and communication with the client applications. The OMS supports an arbitrary number of client applications that identify themselves with a client ID. Client applications can have two roles:

- Applications that perform administrative tasks (e.g. adding sets or business processes, or performing changes in the Object tree)
- Applications that request or receive data from business processes or Object tree queries.

Messaging with the client application is handled using XML and TCP or HTTP message-transfer-bindings. All OMS services are represented through XML messages that can be sent to the OMS by a client application. In addition, messages are defined for queries and query results and business process notification actions.

4.4.2. Alternative Implementation Approaches for the Auto-ID Object Model History

The current prototypical implementation of the history of the Auto-ID Object Model simply uses a relational database management system (RDBMS). The disadvantage is the gap between the object-oriented model of the hierarchical Object tree and the relational model of the history. Moreover, the temporal aspect of the history has to be taken into account in the SQL queries. Other database concepts such as object-oriented or temporal databases provide an alternative approach for the implementation of the Auto-ID Object Model. Other interesting approaches are the RDBMS extensions such as the PostgreSQL extension for tree support.

Object-oriented Database Approach

The great advantage of the object-oriented database approach is that on the one hand an application such as the OMS can store objects in their entirety in a persistent manner. On the other hand it can access objects retrieved from the database (i.e. stored persistently) in the same way as it accesses run-time objects in memory. This adds greatly to the transparency of the whole application, considering that there is only a single data type to handle.

When discussing object databases, the following two kinds of approaches can be taken: Object-oriented [125, 33, 93] or Object-relational DBMSs [145, 146]. Table 4-5 gives a short overview of the advantages and disadvantages of the two approaches.

Table 4-5 Comparing OODBMS and ORDBMS

Object-oriented Database Systems (OODBMS)	Object-relational Database Systems (ORDBMS)
• Integrate seamlessly with an object-oriented programming language • Database itself handles an object as an entity, compared to data rows as the entity of RDBMSs • Same data model can be used to design an application and the underlying database	• Trade-off between traditional RDBMSs and OODBMS • Core database is relational • Additional layer on top of database performs the object-relational mapping • Mapping layer translates OO-style data modifications written in the Object Definition Language and queries Object Query Language to

Object-oriented Database Systems (OODBMS)	Object-relational Database Systems (ORDBMS)
• Objects created within an object-oriented programming language can persist on disk in a transparent way, complete with their structure, procedures and values	the relational definition/query language of the core database • Needs OO application specialists, RDBMS specialists, and object-relational mapping specialists

Object-oriented functionality would facilitate the implementation of the history of the Auto-ID Object Model. It would allow storing an Object directly, including the references to its parent Object and its child Objects, the linked Auto-ID readers and sensors, and its properties. An extensive overview over commercially and freely available object-oriented databases can be found at [30].

Temporal Database Approach

Temporal databases (TDBs) are an extension to relational DBMS. The TDB glossary [45] defines a TDB as follows: "A temporal database is a database that supports some aspect of time, not counting user-defined time." As a synonym we could also speak of Time-oriented Databases. A more in depth introduction to TDBs can be found in [60, 151].

Storing the object history of the Auto-ID Object Model in a database means that the same object will be stored in a lot of records. Each time an object moves from one location to another, the exit time of its currently valid record is set, and a new record is created in order to reflect the new location of the object. First of all this introduces a difficulty in terms of the primary key used for the object history table. It would be natural to use the Object ID as primary key within a table of objects and their locations. However this is not possible, since the object IDs are not unique in the temporal dimension. Two possible solutions are conceivable: Either we take a composite primary key, for example the Object ID and its enter time, or we introduce an additional column with a unique ID. Second we introduce the problem of time-integrity. If we want to have available a continuous history of every object, it is very important that there be no gap between the exit time of an object's previous location and the enter time of its new location. Checking this on the application level is not very comfortable; instead it would be more transparent to indicate only the

new location of an object, and having the DBMS set both timestamps automatically. All these requirements would be met by a TDB.

Table 4-6 Available time-relational DB layers

Product Name	Target RDBMS
TimeDB	All major RDBMS
JTemporal	All major RDBMS
Timetier	Oracle
Tiger	Oracle
TAU	BerkeleyDB, MySQL
BtPgsql	PostgreSQL

As of today, no native TDB exists, instead there are layers and modules available for all major RDBMS systems that emulate some temporal features (see Table 4-5). They offer their own temporal-augmented data query/definition languages and translate these to standard SQL. For example, JTemporal [25] is a Java framework that manages temporal associations and relieves the developers of many time related issues.

PostgreSQL Extension: The module ltree

ltree [139] is one of the extension of the PostgreSQL [11] RDBMS. It provides support for storing and querying hierarchical data by introducing new PostgreSQL data types ltree, ltree[], lquery, and ltxtquery, operators on them, and predefined functions for easier handling of these new data types.

A hierarchy tree created through ltree is stored in a single table with a column named label path that is constructed for each node/row according to a rule. A simple example would be a tree where `Retailstore` is the root node, `Backroom` one of its children, and `CooledStorage` a child node of `Backroom`. The label path of the node `CooledStorage` would then be `Retailstore.CooledStorage.CooledStorage`. The operators and functions defined on objects of the type ltree allow querying the tree to a specified depth, matching node names according to a pattern which can be defined by special modifiers, or testing whether a tree T1 is a descendant/ancestor of another tree T2.

The functionality provided by ltree would facilitate storing the Object tree in a DBMS and would enable the OMS to query the tree with a single call to the ltree module, instead of querying the database for each and

every parent object of an object in question as it is needed, for example in the *LocationsOfObject* query (see section 4.4.1).

4.5. Related Work and Discussion

The need for Auto-ID data management and an Auto-ID infrastructure including hardware abstraction and middleware have been discussed in a number of publications [28, 38, 63, 66, 121, 129, 131, 132, 152, 123]. They all agree in structuring an Auto-ID infrastructure in different layers, which can be generalized as followed:

1. A *hardware layer* that contains the readers, sensors and tags and the networking hardware. Readers and sensors can have integrated software that performs low-level Auto-ID data processing. Readers and sensors have a common interface towards the higher layer or some kind of hardware abstraction / device controller approach is offered.
2. A *middleware layer* that contains software that collects Auto-ID raw data from different readers. It filters and aggregates the raw data to prevent redundant information and provides meaningful Auto-ID observations to the higher layer. Auto-ID middleware is sometimes referred to as Auto-ID Edgeware.
3. An *application and information system layer* that contains applications responsible for enriching Auto-ID observations with application and business relevant context. In addition, persistency of the enriched data is provided. Existing enterprise applications, information systems and business workflows are integrated with the enriched Auto-ID data.

The layers group hardware and software components for an Auto-ID infrastructure. The layers can also be associated with a corresponding data model or representation: (i) The hardware layer models data simply as tag IDs and tag memory (i.e. raw data), (ii) the middleware works with Auto-ID data as events or observations that include the source of the event (i.e. the reader or sensor), a list of tag or object IDs or sensor values and a list of fields with their values and (iii) the application and information system layer has an implicit or explicit model of the physical objects and their state which can simply be an extension to Auto-ID events including business context information such as location or business activity or a more complex object-based model of the physical world.

In the last years much research has been performed concerning the first two layers to provide an adequate interface to Auto-ID hardware and to provide "clean" Auto-ID data to applications in an efficient manner. The main objective of approaches for filtering and aggregating Auto-ID data is the elimination of duplicate observations of tags and dealing with false-positive and false-negative observations.

Since some approaches for Auto-ID data processing and middleware include concepts of layer three (e.g. enriching data with business information), we propose to separate layer three into a layer for data enrichment, representation and persistency and a layer for applications and information systems (see Table 4-7).

Table 4-7 Layers, data models and data processing of an Auto-ID infrastructure

Layer	Data Model	Data Processing / Functions
Hardware / Hardware Abstraction	Raw Auto-ID Data	Low-level processing related to air-interface (identification, memory access, filtering)
Filtering and Aggregation	Auto-ID observation/ event	Collection, filtering and aggregation of data from multiple readers
Data Enrichment, Representation and Persistency	Auto-ID specific data model / Object representation	Enrichment of Auto-ID data with business context Representation of physical objects Persistency of data model Business process support
Applications and Information Systems	Application specific, associated with master data in ERP systems	Application-specific functions, e.g. track and trace of objects, visualization of object flows, management cockpit

In the following sections, we relate existing research to our approach using different criteria including Auto-ID data models and representations, data processing, provided functionality and services to applications and support for application logic and business processes. Related work can be

grouped into: EPC Network approach (see section 4.5.1), Auto-ID middleware and infrastructures (see section 4.5.2) and approaches in Ubiquitous Computing infrastructures (see section 4.5.3).

4.5.1. EPC Network

One of the major research and standardization efforts in the area of Auto-ID was the Auto-ID Center, founded in 1999 at the Massachusetts Institute of Technology (MIT) with several industrial sponsors [130]. The goal of the Auto-ID Center was to foster research to provide low cost RFID tags and readers [133], a standardized numbering scheme for objects identified via RFID, the Electronic Product Code (EPC) [29], and an infrastructure for the RFID data management and sharing across the supply chain [22]. The research of the Auto-ID Center was supported by a network of universities and research centers, the Auto-ID Lab.

In 2003, the Auto-ID Center was transformed into EPCglobal Inc., a nonprofit organization with the objective to commercialize and manage the EPC standards and to continue the development of standards [64, 153]. The Auto-ID Labs, in cooperation with EPCglobal, continue to perform research in a variety of Auto-ID related research areas.

The family of EPCglobal standards related to EPC, air interface protocols, software interfaces and directory services are combined in the EPCglobal Architecture Framework [52, 153, 38, 69], also called the EPC Network. The software standards of the EPC Network describe interfaces and data structures for communication and roles that software components fulfill when implementing an interface (see Figure 4-19). The standards the EPC Network can be grouped according to the layers described above (see Figure 4-20). Starting from the hardware layer, the standards are:

- Tag Data Standard [50] describing the EPC
- Low Level Reader Protocol [53] to communicate with RFID readers
- Application Level Events (ALE) [47], the interface to communicate with the RFID Middleware
- EPC Information Services (EPCIS) [48] describing interfaces to store to and query data from an EPCIS Repository
- Object Naming Service (ONS) [54] a look-up service for EPCIS repositories

The *EPC* acts as a meta-schema that allows integrating existing number schemes widely used in different application domains such as the Serialized General Trade Identifier (SGTIN) or the Serial Shipping Container Code

(SSCC). Another supported schema is the Serialized Global Location Number (SGLN) to identify locations. The Tag Data and the Tag Data Translation [51] standards define different representations of an EPC: A binary representation which is stored in memory of RFID tags and a symbolic representation in a URN format. For example, a SGTIN EPC could look like `urn:epc:id:sgtin:0037000.112345.400`. It contains information about the producer, product class and the product serial number for a product which can be used by the application to count the number of products of a certain class in a shipment or filter on products of a specific producer.

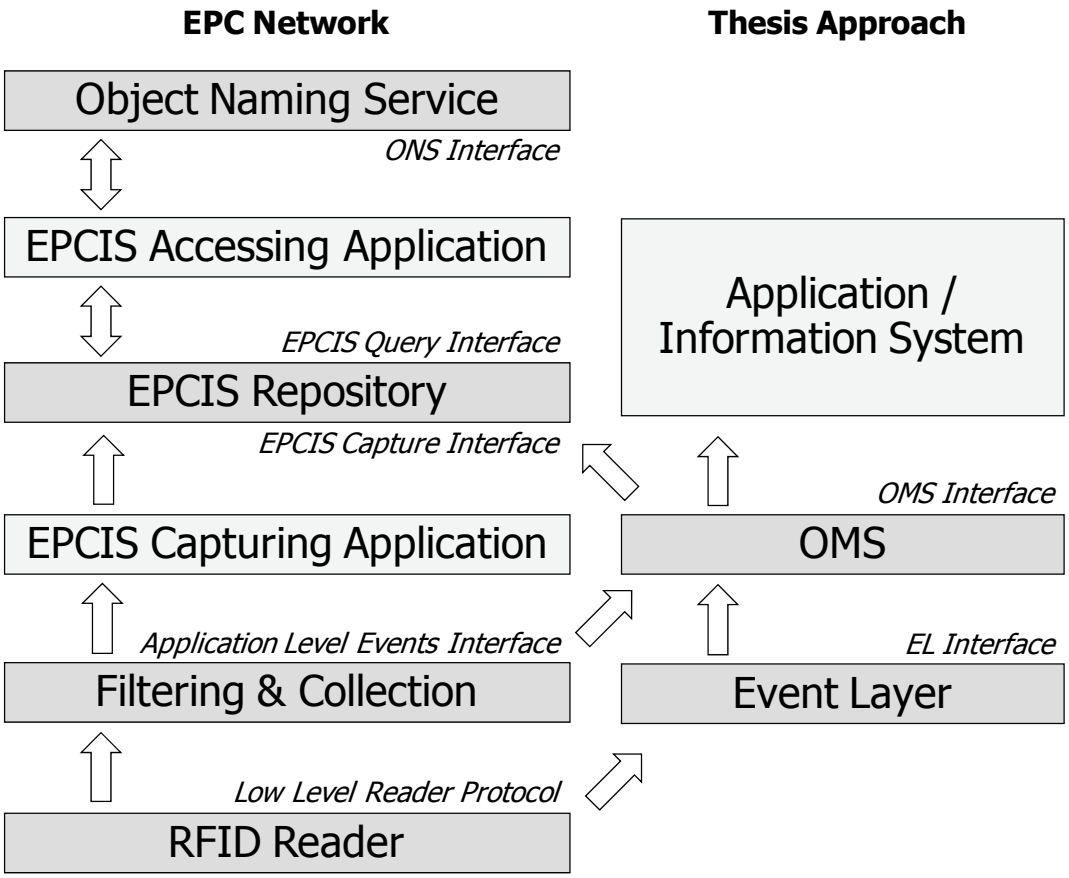


Figure 4-19 EPCglobal Architecture Framework compared to OMS

The *Filtering and Collection Middleware* role defined trough the ALE interface provides filtering and aggregation of raw RFID data from readers and acts as a single interface to the potentially large number of readers that make up an Auto-ID system deployment. The ALE interface hides Auto-ID technology details from the applications and provides a pub-

lish/subscribe mechanism for applications to define in which filtered and aggregated data they are interested. Filtering can be performed on EPC-level, that is EPCs can be filtered based on producer or object class for SGTIN EPCs. In general, the filtering is achieved via bitmasks on the EPC. Aggregation allows summarizing all EPCs observed over a certain period of time, called an Event Cycle, and eliminating duplicate observations. In addition, reports can be generated based on all observed EPCs or only the new ones (i.e., additions) resp. the ones that are not observed any more (i.e., deletions). Another form of aggregation is the bundling of all observed EPCs of several readers into the observations of a so called logical reader. Once the readers capture relevant tag data, they notify the middleware, which combines the data arriving from different readers in a report that is sent to the subscribed applications according to a predetermined schedule. The ALE specification defines a SOAP message transport binding for the subscription communication channel and an XML and TCP/HTTP message transport binding for the notification channel.

The *EPC Information Services (EPCIS)* consists of three roles and two interfaces. The roles are the *EPCIS Repository*, a database for Auto-ID events enriched with business information (called EPCIS Events), the *EPCIS Capturing Application* that stores EPCIS Events in the EPCIS repository and the *EPCIS Accessing Application* that queries the EPCIS Repository for EPCIS Events.

The EPCIS Repository is defined through the two interfaces for storing EPCIS Events in the repository, via the EPCIS Capture Interface, and for retrieving them via the EPCIS Query Interface. The EPCIS Query Interface provides synchronous and asynchronous queries for the applications. Both capturing and querying data is based on the data model of EPCIS Events that contain all relevant data describing an event. There are the following four types of events:

- Object Event describing an observation of one or many objects identified via an EPC
- Aggregation Event describing the physical aggregation of several objects (e.g. packages loaded on a palette)
- Quantity Event describing the number of objects of a certain object class
- Transaction Event connecting EPC observations with business transactions

EPCIS Events contain data about the date and time the event happened, the main information what happened (i.e. the list of EPCs, parent and child EPCs for aggregation events or quantity for quantity events), where the event happened (i.e. the read point identifying the reader and the business location) and data why the event happened (i.e. the business step or the business transaction). Data that enriches an EPCIS Event with business information such as business location or business step is based on master data that is organized into vocabularies to structure the master data for different application domains. For example, business context for fast-moving consumer goods (FMCG) is described in its own vocabulary that defines the possible business steps including picking, shipping or storing.

Since the data of EPCIS Events highly depend on the business context and the application domain, implementations of the EPCIS Capturing Application role differ significantly from each other and will thus have to be developed on a case-by-case basis. EPCIS Events could be generated automatically only for trivial cases such as simple EPCIS Object Events describing a simple observation of a physical object.

Applications that retrieve data from the repository act as EPCIS Accessing Applications. As such an application (e.g., a warehouse management system) is not a middleware component and resides in the application and information system layer.

In a distributed application environment such as supply chain scenarios, EPCIS Accessing Applications need the access path to EPCIS Repositories in order to retrieve data about required EPCs. EPCglobal provides two different look-up services for EPCIS Repositories: First, the Object Naming Service (ONS) and second the Discovery Services. ONS is an existing standard that is merely an extension of the existing Domain Name Service (DNS) infrastructure. ONS only allows look-up of EPCIS Repositories of manufacturers of goods identified with an EPC (i.e., the organization that brings an EPC into life). The Discovery Service standard is still in work and will provide information about all or selected EPCIS Repositories that store information about an EPC in question.

Comparison of the Thesis Approach with the EPC Network

Since the hardware layer is out of scope of our approach, the OMS merely uses Auto-ID readers with different interfaces including readers with the LLRP. Likewise, we also incorporate the symbolic representation of the EPC as a numbering and identification scheme for objects in the Auto-ID

Object Model. The main advantage is its standardized structure that has the advantage of filtering and aggregating on owner or object class level simply based on the EPC. Other advantages are its widely usage and acceptance among all Auto-ID middleware approaches. For example, a SGTIN EPC in its symbolic representation could look like `urn:epc:id:sgtin:0037000.112345.400`. Since the EPC is not limited to identifying consumer goods using the GTIN, we also apply it to identify location objects, readers and sensors in the Auto-ID Object Model. Using the EPC also allows for a seamless integration of the OMS and an EPC Network deployment (see Figure 4-19).

Compared to the filtering and aggregation concepts presented in section 4.1, the Filtering and Collection Middleware resp. the ALE interface provide the filter types of reader identifier, tag identifier and data and the aggregation types observed, entry and exit, and logical readers. Since these filtering and aggregation types are sufficient to generate clean Auto-ID data as input for the higher layers, a Filtering and Collection Middleware implementation can act as a feeder for the OMS. Only the passage aggregation type does not exist and has to be adapted in the OMS based on enter and exit aggregation types.

Analyzing business context enrichment, the Filtering and Collection Middleware only provides very limited mechanisms to incorporate business logic (i.e., using logical readers as locations). It does not allow extending filtering and aggregation or providing own mechanisms via plug-in. It also does not provide persistency of ALE events.

The EPCIS data model, that is the EPCIS Events, does not provide a complete model of physical objects. The EPC as an object identifier can be seen as a representation of a physical object. However, object properties cannot be modeled in EPCIS Events. Locations can be attached to any type of EPCIS Events as business locations. This allows for a track and trace of objects identified via EPCs. The EPCIS meta data also provides child relationships between locations. The major drawback in the EPCIS data model is the emphasis on EPCIS events describing observations related to EPCIS. It does not provide a model that represents the physical world at a certain moment in time. The EPCIS is merely a collection of events that can be used to reconstruct certain observation of the physical world using a varying number of queries depending on the needed information. In contrast, the Auto-ID Object Model provides a structured representation of the physical worlds of objects, their properties and their

relationships. Since the Auto-ID Object Model provides already a certain level of business context (i.e. locations and certain properties), it is partly on the same level as the data model of the EPCIS. On the other side, certain actions of the business process support of the Auto-ID Object Model also correspond with EPCIS Events.

The EPCIS Repository is simply a database only for EPCIS Events. It does not allow storing other kind of events such as ALE or custom events. Moreover, it does not automatically generate EPCIS Events nor does it provide any business or application logic support. Information about business processes is only supported as data fields identifying business steps or transactions related to an EPC observation. Business context enrichment has to be implemented in an EPCIS Capturing Application that links the Auto-ID observations from the Filtering and Collection Middleware and other information from additional application dependent sources with EPCIS Events to be stored in an EPCIS Repository. Since the OMS provides actions for creating and storing EPCIS Events, the OMS can take on the role of an EPCIS Capturing Application. The OMS can therefore integrate into an EPC Network deployment and provide non-standardized generic capturing application support (see Figure 4-19). The business process definitions provide the means to define how to transform the state of observed physical objects and their relationships into events relevant for a higher business workflow. An EPC Network deployment can easily be setup using the components of the open source project Fosstrak [67, 69]. In the model of the OMS the EPCIS Repository would reside in the application and information system layer.

The ONS and the proposed Discovery Service only provide a look-up for EPCIS Repositories. In an Auto-ID infrastructure deployment many more hardware and software components exist that have to be connected. This connection should ideally happen via loose coupling in case components have to be exchanged or an access path to a component is changing. A more general look-up service as presented in our approach provides for such a loose coupling among components of an Auto-ID infrastructure.

Generally analyzing the EPCglobal Architecture Framework, it only supports Auto-ID with EPC as the identifier. No sensor or actuator support is provided in the different standards. Only sensors such as light barriers that act as triggers for readers can be modeled in the EPC Network on a low-level in the LLRP and ALE. The Auto-ID Object Model abstracts from sensors through object properties and actuators through func-

tions. Research concerning sensor support for RFID tags (i.e., EPC class 3) and RFID tags that can act as sensor nodes in a network (i.e., EPC class 4) is ongoing [156, 49].

Another analysis of the EPC Network [126] argues that in an EPC Network deployment the EPCIS Repositories offering persistence to EPCIS Events are not sufficient and that a new component called Enterprise Location Services is needed. Such a service would provide advantages related to Auto-ID in an enterprise and would not be targeted to the benefits related to the exchange of Auto-ID data such as EPCIS Events among enterprises in a supply chain. The Enterprise Location Services would store all Auto-ID related events simply with their location before interpreting these events and generating EPCIS Events. Such a service would provide in exceptional cases, such as a safety incident or an incomplete shipment, valuable information about the cause of such events that are not available when only considering EPCIS Events [121]. The OMS providing the persistence of the Auto-ID Object Model fulfills most of the requirements of the proposed Enterprise Location Services in an EPC Network deployment. Only the efficiency and the access control requirement are not fulfilled since they are out of scope of our research.

4.5.2. Auto-ID Middlewares/Infrastructures

Besides the EPC Network there are a number of open source and commercial Auto-ID middleware approaches and research in Auto-ID data management available. The Auto-ID middleware approaches [123, 148, 28, 97, 4, 46, 136] are either database-centric [156, 86] or extend existing enterprise application servers. The emphasis of most of these approaches is to provide data filtering, aggregation and simple data enrichment. Very little generic application-level support is offered. Other research in Auto-ID data management exists in the field of complex event processing [156, 162, 79] and in applying agent-based approaches [111].

Among the database-centric approaches of RFID data management Wang [156] presents a temporal data model for RFID using a Dynamic Relationship ER Model (DRER). The temporal aspect of their model is related to proposed temporal aspects in SQL [143]. The presented DRER model describes dynamic relationships between the following entities: Readers, locations, objects and transactions. For example the relationship between location and object is expressed as the object-location relationship containing the location, the object and a start and end time at which

the object was at the location. Similarly the reader-object relationship is expressed as an observation containing the reader, the object and the timestamp at which the object was observed by the reader. The model is instantiated in a RDBMS as tables for the entities and tables for the relationships. In addition to the temporal data model, effective query support for complex queries such as tracking of objects and automatic data acquisition and rule-based transformation to filter and aggregate data is described. The transformation mechanism includes a simple declarative rule definition language and allows data filtering of observations, location transformations based on observation and data aggregation resulting in containment object-object relationships and object-transaction relationships.

Compared to our approach the presented model mixes Auto-ID specific information (i.e., readers, reader-location and observation), with object-related information (i.e. object, object-location and transaction). The transformation mechanism is on the one hand responsible to filter and aggregate raw data and on the other hand to fill the model (i.e. for the object-location, object-containment and object-transaction relationships). The second part is data enrichment with context whereas the first part simply is filtering and aggregation. The model therefore combines data representation and processing of two layers. Our approach clearly separates Auto-ID specific data processing relying already on filtered and aggregated data by lower layers and providing a clear object model to applications abstracting from Auto-ID specific details. We use a similar approach of database model and queries for the object history, since it follows accepted methods in temporal database approaches [143]. Business process support can only be seen in automatically determining a transaction that is related to an object observation. This results in updates in the model which applications have to query. Our approach on the other side provides full business process support with actions to notify applications or integrate with information systems.

A complementary approach for RFID data management is presented by Hu et al. [86]. The emphasis is placed on the representation of the object identifier EPC as a bitmap data type in the RDBMS. The advantage is the support for queries on part of the EPC such as producers or object class which would not be possible if the EPC would be stored as a string or integer data type. Several possible database table representations for different applications and the related queries are presented, in addition to an implementation of the proposed EPC data type in a RDBMS. The presented approach would allow implemented the OMS with a much higher

performance than the current prototype implementation which uses a simple string data type to store the object identifier in the object history.

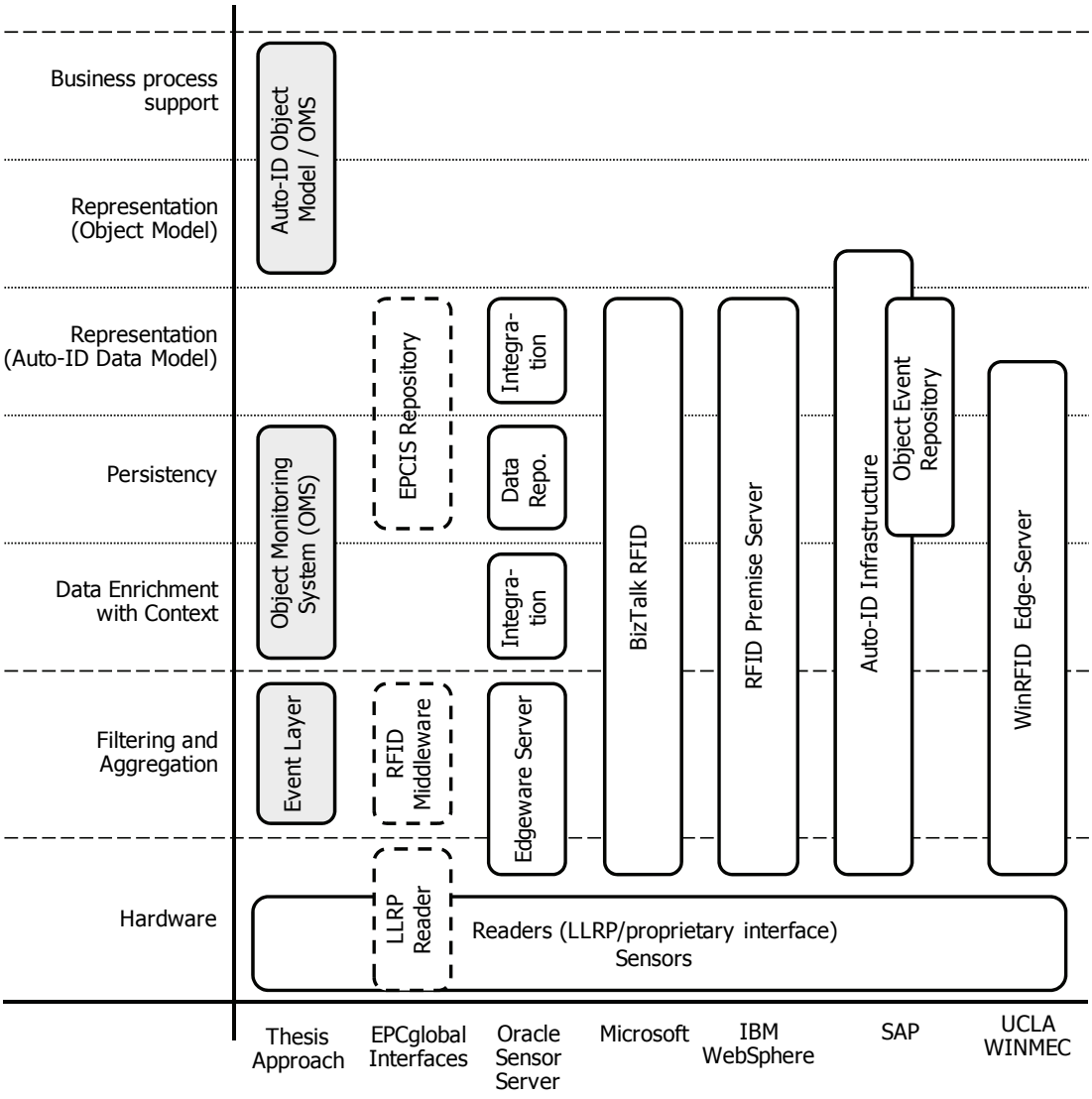


Figure 4-20 Comparison of different Auto-ID middleware/ infrastructure approaches

Several approaches in managing Auto-ID data are based on complex event processing [109] which has its root in active database research [36]. These approaches [156, 162, 79] are based on an Auto-ID data specific event model. The rules that can be defined in such event processing range from simple event condition action (ECA) rules to complex petri net based rule definitions that also incorporate temporal data processing. Since these rules are based on Auto-ID specific concepts (i.e., Auto-ID

observations) and events and not a model representing the state of the physical world, formulating more complex business rules such as the arrival of a shipment becomes a tedious task and requires deep knowledge of the complex rule definition languages.

Comparing our approach to the more complex rules including temporal aspects, we provide a data model that abstracts from Auto-ID specific details and allows for rule definitions that are much closer to the language of business processes than rules based on Auto-ID observations. Conceptually, our approach is based on state changes of the objects in our model that results in events to applications and not on events that are the result of Auto-ID observations. From a more abstract point of view, our approach can also be seen as complex event processing, since the object model of an application is dynamically generated based on Auto-ID observation. However, the main advantage of our approach is (i) the Auto-ID Object Model which represents the real world more intuitively than Auto-ID observations and (ii) the simpler definition of business processes which provides a language with which a system engineer or domain expert is more familiar.

At the University of California LA (UCLA) research concerning RFID and middleware issues is performed under the umbrella of the Wireless Internet for Mobile Enterprise Consortium (WINMEC) [12]. Their approaches resulted in an RFID middleware called WinRFID [123, 148]. The architecture of WinRFID is layered in five layers that can be mapped to some of the layers presented above (see Table 4-7). The hardware and protocol layer abstracts from different RFID hardware and air and reader communication protocols. WinRFID supports the integration of a variety of different readers. The data processing layer performs filtering and aggregation of RFID data. Applications can create their own data processors using rules to define filters and aggregators. Data processors can be linked in a chain which allows for a flexible definition of filtering and aggregation mechanisms. RFID data and other information can be modeled as different events such as system events (indicating the system status), device events (indicating device status), data events (transporting RFID data) or user events (indicating status determined by user defined rules). The rule approach to filtering is related to complex event processing described above and allows the generation of events based on the input of other events. The data presentation layer allows creating different XML-based representation of event data in formats that can be processed by applica-

tions. Different so called connectors are available that perform the data representations and communication with applications. Two connectors are provided: the database connector that allows populating an RDBMS with processed events (i.e. data persistency) and the portal connector that provides data to web portals using a publish/subscribe mechanism.

The WinRFID approach provides Auto-ID application support starting from the hardware layer up to the persistency layer (see Figure 4-20). The filtering and aggregation mechanism is similar the approach of the Event Layer that can be used to feed events into the OMS. The flexibility to combine events in a chain and the ability to provide additional event processors using the rules allows the enrichment of the data with business context. The data representation, however, is only based on events and no consistent Auto-ID data or object model is provided. Incorporating business rules from the application into a WinRFID deployment using the provided rules is therefore not a straightforward task. The rules contain many Auto-ID specific concepts and will mix these concepts with application domain concepts. Maintaining these rules for an application will be complicated and costly. Moreover, no specific business process support is provided. Events notifying an application about exceptional states of the physical objects have also to be implemented using rules and low-level events.

A multi-agent based RFID middleware approach is presented in [111]. The presented RFID middleware is based on Agent-oriented Software Engineering concepts and uses the Process for Agent Societies Specification and Implementation (PASSI) methodology [39] which facilitates the design and implementation by providing tools, patterns and automation. The multi-agent approach has the advantage of distributed and concurrent problem solving and new patterns of interactions such as cooperation, coordination and negotiation. The different types of agents in the presented approach have certain responsibilities such as controlling a reader, generating events or reporting events. The agents are grouped into the following layers: Device management, data management and interface layer similar to our presented layers above. Inter-agent communication is based on an ontology that describes the different entities such as reader, antenna, tag or event. The event generation and event data model is based on the EPCglobal ALE standard (see section 4.5.1). As a case-study an asset management application is presented.

Since the filtering and aggregation mechanism is based on the ALE interface, only limited business context enrichment and no business process support can be delivered. The approach using multi-agents has the mentioned advantages and provides a valuable contribution to the development of RFID middleware. Especially for a highly distributed Auto-ID deployment the agent-based approach allows for an easy distribution and scalability. However, since the communication with the application is still based on traditional interfaces there is no advantage in Auto-ID application development.

Several commercial Auto-ID middleware and infrastructure approaches are extensions of existing database and application servers. The Oracle Sensor Edge Server [10, 4], the IBM WebSphere RFID Premises Server [16, 46] and the Microsoft BizTalk RFID Server [1, 136] provide all the EPCglobal ALE and EPCIS standard interfaces and allow integrating LLRP readers and readers with proprietary interfaces using adapters that are provided for the most common readers. In addition to the EPCglobal interfaces, proprietary interfaces of the different application servers can be used by applications, for example to plug-in filters or aggregators or special reporting tools. By extending the filtering and aggregation capabilities, applications are provided more flexibility than the ALE interface would allow which provides basic data enrichment with business context (see Figure 4-20). All applications provide a data model based on Auto-ID concepts.

All three approaches provide high performance filtering and aggregation mechanisms and an Auto-ID data model, however, they are still based on Auto-ID specific concepts and none of the approaches provide a clear object model. Business process support is not offered and has to be implemented by special applications or by the integration of workflow enterprise software.

The SAP Auto-ID Infrastructure (AII) is based on research on smart items [28, 97]. As the three other commercial approaches, the AII provides the EPCglobal interfaces of ALE and EPCIS. However, partly it is still based on an outdated Auto-ID data model of the Auto-ID Center, the Product Markup Language (PML) [65]. In the business enrichment, the AII goes one step further than the other commercial approaches and offers a hierarchal location model in addition to Auto-ID events. In the location model, detected objects can be added based on rules that the application can define similar to the approach of Wang [156] described above. The AII therefore partly provides an object model with location and objects represented

through their EPC. However, the rules can only be used to construct the location-object model which is persisted but cannot be applied on the model itself to provide business process support. The AII does not provide a consistent object model to which all of the stored data has to comply.

4.5.3. Ubiquitous Computing Infrastructures

Ubiquitous Computing (Ubicomp) infrastructures such as GAIA [127, 82], Aura [144, 73], or Nexus [71, 24] provide general models and application abstractions that focus on concepts such as persons and devices, user mobility, intelligent environments, and the processing and modeling of context information. The goal of these very general and abstract approaches is to support the construction of a great variety of Ubicomp applications that follow the vision of Marc Weiser [158]. An overview of existing Ubicomp middleware and infrastructures is presented in [135].

However, due to the generality of the Ubicomp approaches, the effort to build applications is still high. Identification and monitoring of objects is interwoven in these infrastructures and not clearly separated against other services. Each infrastructure defines a different object model for the identification and monitoring service. For example, the data model of Nexus is a distributed location model that supports the development of spatial aware applications, that means location is the most important entity in this model.

Our approach focuses only on the identification and monitoring of objects as a basic building block to construct Auto-ID applications. The Auto-ID Object Model provides a more restricted but also more focused object model on which an application or infrastructure can be build and which can be extended. We do not need to take many different physical entities into account (e.g. distinguish between persons, devices or products), the main objective of our model is to represent the physical world as objects that can be automatically identified and their relationships. Applications that need to differentiate objects of certain types can easily add properties to these objects or define object IDs to include object types or classes.

Compared to symbolic location models presented in [107, 26] our model is extended by time, i.e. we provide a history of objects and their properties, which is also provided by Nexus. The Auto-ID Object Model merges the concepts of object and location into the single concept of ob-

ject. In addition, state machines with conditional state transitions based on a certain state of the objects in the model are offered to provide business process support and notify applications only in exceptional cases. In contrast to the complex rule language in the GAIA infrastructure we operate only with sets of objects, number of objects, and object properties.

Some Ubicomp infrastructures such as the Cooltown infrastructure [94, 124] use the semantic location model. Since our location model is based on Auto-ID technology, that means object information is provided by Auto-ID sensors, it is difficult to automatically represent the symbolic grouping of objects. For our purposes of representing object containment it suffices to detect the real physical containment relationship.

Römer et al. and Schoch propose a smart identification framework for Ubicomp applications [129, 134]. The goal of the framework is to contribute towards the realization of the vision of a world with 'smart' everyday items. Smart everyday items differ from regular everyday items insofar as they know their whereabouts, perceive their environment, and are able to communicate with other smart things. The framework includes a model to describe the world of collaborating everyday items. The model consists of locations, things, and tags attached to them and a representation of the thing in an IT system. The framework also includes a service infrastructure that allows for the coupling of the tag and its representation and that provides a history of the smart things and query capabilities.

Compared to our approach, the smart identification framework also provides an object model where objects, their locations and their relationships can be described. The representation of physical objects is called virtual counterparts. However, the model includes Auto-ID concepts of tags thereby mixing several concepts. Moreover, it differentiates between the physical object and its representation in an IT system which makes the implementation of an object transparent to the application and not only the abstraction. In the Auto-ID Object Model we only provide the abstraction of physical objects (i.e., Object, Property and Object relationships) to the application. The smart identification framework places its emphasis on the management of smart things and provides only services of these smart things to the application. There are no mechanisms that allow applications to plug-in application logic directly into the infrastructure that manages the smart things comparable to the business process support of the Auto-ID Object Model.

The approach of smart things goes beyond the scope of Auto-ID applications where physical objects are identified and applications are interested in the whereabouts and state of these objects. Smart things take a more active role and are able to actively communicate with each other and the environment. The application setup is therefore far more complicated than for Auto-ID applications which involve application logic on the smart things. Our approach therefore reduces the object model to provide simplicity to the applications and emphasizes more on the support for applications.

5. Visual and Generative Tool-based Auto-ID System Development Process

The previous section described how the gap between the applications or information systems on the one side and the Auto-ID systems on the other can be bridged by providing appropriate data representation, application logic support and services to the applications and information systems. Applications can build on interfaces that abstract from Auto-ID specific details. In an Auto-ID application deployment, applications interface with the Data Enrichment, Representation and Persistency layer, represented by the OMS component, that has to be instantiated and configured according to the required needs of the specific application domain. Moreover, the other components in a deployment such as look-up services, filtering and aggregation components, readers, sensors and actuators and their connections have to be configured.

Typically these configurations have to be defined in component specific formats (e.g. XML or text based configurations) and often certain parts of components have to be implemented and integrated into these components. These tasks require specific knowledge of all software and hardware components of an Auto-ID application deployment in addition to software development skills.

We propose to bridge this gap between Auto-ID application developers on one side and the components of an Auto-ID infrastructure on the other by an development process based on generative and visual programming concepts. The visual tool-based process provides a concrete representation of the application domain and supports non-software developers in the different tasks over the lifecycle of an Auto-ID infrastructure. During design time, it allows to instantiate the Auto-ID application model towards a certain Auto-ID application (e.g. a retail store management application) and facilitates the configuration of all components of an Auto-ID infrastructure. For the deployment phase, it offers a different view to visually setup and to connect the different hardware and software compo-

nents. During runtime it can act as a "management cockpit" to dynamically visualize the state of the infrastructure and allow changes to the setup and configuration. In addition, it can also be a testing tool to simulate the movement of objects to test the business logic. As shown in Figure 5-1, the proposed tools support all the different layers of an Auto-ID infrastructure.

Since the Auto-ID Application Development Process is based on different concepts of generative and visual programming, an overview of these concepts is given in section 5.1. The process and the visual tools are presented in section 5.2 and compared to related work in section 5.3.

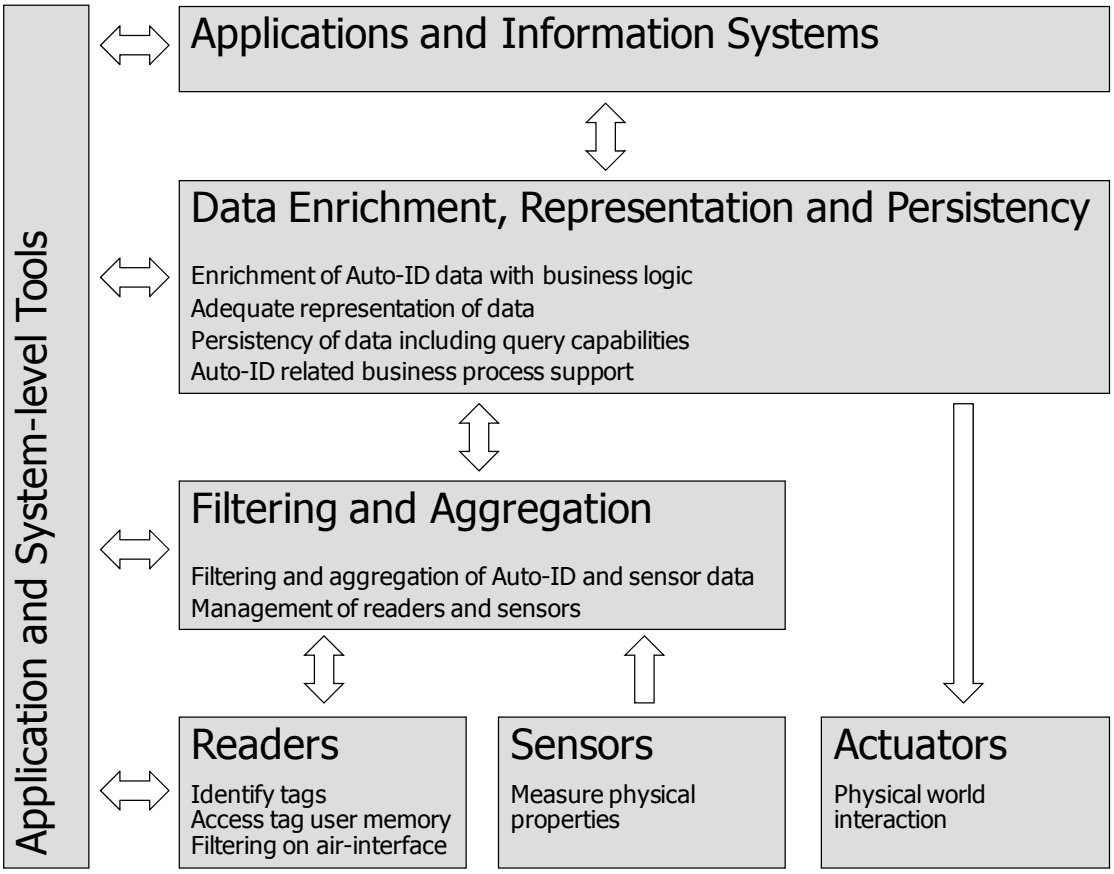


Figure 5-1 Tool support for the Auto-ID infrastructure layers

5.1. Generative and Visual Programming Concepts

5.1.1. Generative Programming

Czarnecki defines Generative Programming as "a software engineering paradigm based on modeling software system families such that, given a particular requirements specification, a highly customized and optimized intermediate or end-product can be automatically manufactured on demand from elementary, reusable implementation components by means of configuration knowledge." [40]

From this definition, two requirements for Generative Programming can be extracted. First, a set of elementary, reusable components is needed from which a family of systems can be generated and second, through the use of automation, these components can be combined to produce a software system. A family of systems can be defined and generated from a common model, a generative domain model, which includes three elements (see Figure 5-2):

1. A family of systems, which forms the *problem space*. The problem space consists of the domain-specific (i.e. application-specific) concepts and features needed to define the implementation components.
2. The components from which the systems can be built, the implementation components. These, in composition with each other, form the solution space. The *configuration knowledge* specifies how the implementation components may be combined, and more importantly, defines defaults for as many features as possible. This is important since it minimizes the work of application generators.
3. Constraints and rules on how the components and their features can be put together, the configuration knowledge. The *solution space* is composed of all possible combinations of implementation components that adhere to the restrictions given by the configuration knowledge. Optimally, the implementation components are designed such that the number of possible combinations is maximized while keeping the implementation intersections between components at a minimum. To implement the generative domain model, generators are used. They represent the configuration knowledge and assemble the implementation components specified by a system specification.

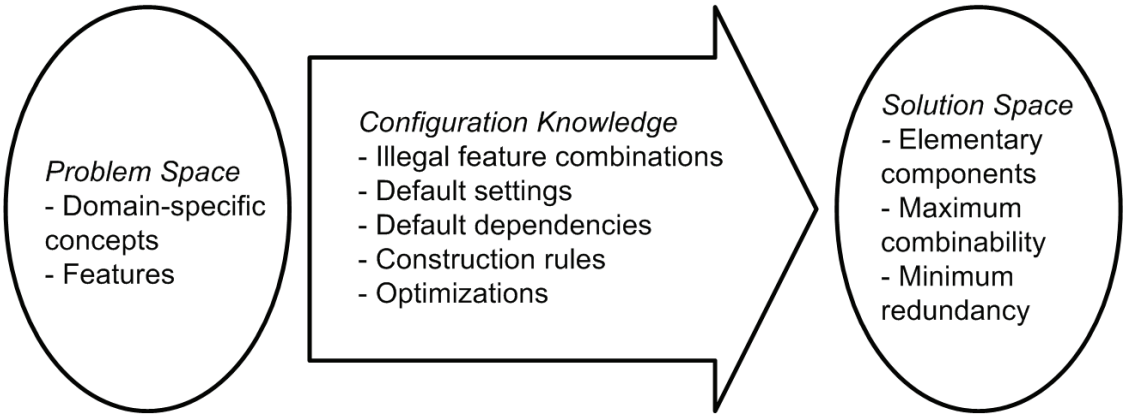


Figure 5-2 Generative domain model (from [40])

Generative Programming offers many advantages, but at certain costs. The most important characteristic of Generative Programming is that all advantages depend on the fact that it is comprised of two complete development cycles: The generative domain model, developed for reuse, and the usage of the model, actually reusing it.

From this characteristic stem many advantages. First of all, since the implementation components and the configuration knowledge can be reused for each member of the solution space. Development costs are reduced with each member produced. If new members need to be produced, most of the implementation components and configuration knowledge stay the same and the implementation components being improved continually instead of being replaced by specialized components.

However, the reliance on the reusability of the components and the model means also that the monetary and temporal cost to develop said components is higher than creating one system. From this we conclude that Generative Programming should be deployed if it is possible to guarantee a high reusability rate and a big enough strategic time window in which to place the products generated with Generative Programming.

As opposed to conventional software engineering, which concentrates on single systems, *Domain Engineering* focuses on providing reusable solutions for a family of systems. It does this by collecting, organizing, and storing constituents of systems in a particular domain in the form of reusable concepts and units, but also by providing the means to reuse/assemble/adapt/combine these assets (e.g., domain models, software architectures, design standards, communication protocols, code components and application generators), when building new systems. An exam-

ple for this is a generic system from which concrete systems and/or components are instantiated, which are then reused in different systems.
Domain engineering consists of three main process parts (see upper part of Figure 5-3) [41]:

- *Domain Analysis*. Analyzing the domain and defining a set of reusable, configurable requirements for the systems in the domain.
- *Domain Design*. Developing a common system family architecture and formulating a production plan.
- *Domain Implementation*. Implementing the reusable parts of the system and the generators.

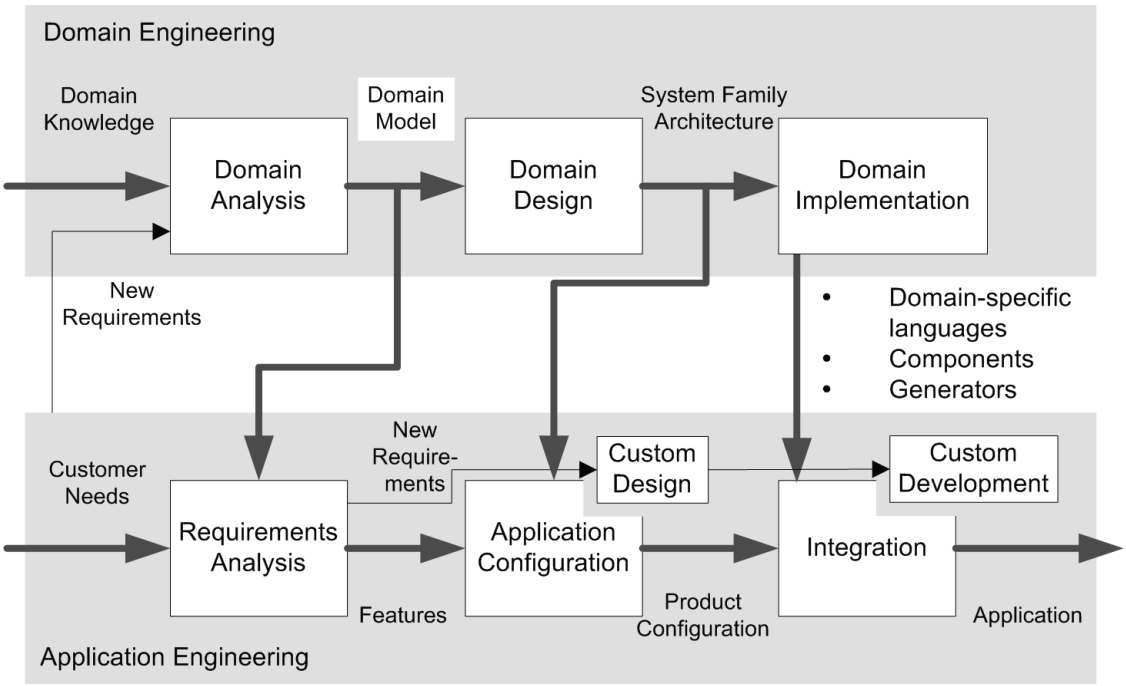


Figure 5-3 Software development based on Domain Engineering (from [41])

Based on the results of domain engineering, systems can be built. This process is called *Application Engineering* [41]. Figure 5-3 illustrates how domain engineering and application engineering interact. The requirements analysis for new applications takes advantage of existing domain analyses and the resulting domain model. In the case a customer require special features not allowed for in the domain model, these requirements are fed back into the domain knowledge but also passed on as a custom design to the product configuration created from the domain design. The custom design is then implemented by a custom development and inte-

grated with the result from a generated or manual composition of the domain implementation to compose the final application.

5.1.2. Visual Programming

The definition of visual programming encompasses conventional flow charts and graphical or visual programming languages. A visual programming language (VPL) is defined as any programming language that allows the user to specify a program in a two-(or more)-dimensional way [31]. Conventional textual programming languages (TPL) are not considered two-dimensional since the compiler or interpreter processes them as one-dimensional streams of characters [142]. The most common goals of visual programming are best summarized in [31]:

- Make programming more accessible.
- Improve programming correctness, that is, reduce the error rate of programmers.
- Improve programming speed.

Five common strategies are usually used to achieve these goals [31, 32]:

- *Directness*. In the field of human computer interaction (HCI), directness refers to the distance that has to be covered from the action of formulating a goal and the achievement of the goal. In the context of object manipulation, it refers to the feeling that the user is directly manipulating the object [138].
- *Explicitness*. Semantics are inherently contained in the graphical structure. For example, a system could express the containment relationship between objects by drawing them inside another.
- *Liveness*. Liveness refers to immediate visual semantic feedback of program edits.
- *Concreteness*. The converse of abstractness. This means that programs are expressed on particular instances.
- *Simplicity*. VPLs use fewer concepts. For example, many VPLs do not need the user to worry about pointers or memory allocation.

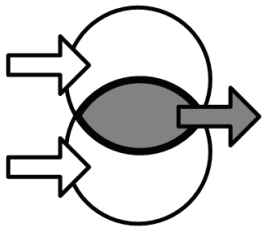
The advantages of visual programming are obvious. The human visual system is very well suited for two- or three-dimensional visual processing. For example, whereas the textual representation of a program relies on the syntactical representation of structures, a visual representation could make use of various gestalt laws [37, 80, 159] to display program structure and functionality.

Current programming practice is to design program structure using graphical aids, for example UML diagrams and flow charts. But as soon as the implementation and instantiation stage is reached, the graphical structures are translated into a textual representation, which only uses rudimentary visual aids like textual indentation or syntax highlighting. Easily human-decoded visual representations make way for more abstract representations, which need more time to be processed.

Despite current practice, theories that improve upon the discrepancy between representations have been stated already a long time ago. David Canfield Smith writes almost thirty years ago: “The most articulate representation for a program requires the least translation between the internal representation in the mind and the external representation in the medium.” [142] He draws upon ideas by Sloman, who distinguishes two kinds of data representation, “analogical” and “Fregean”. In an analogical system, parts and their relations are represented in such a way that “the structure of the representation gives information about the structure of what is represented.” [141], whereas Fregean systems are based on Frege’s syntactic and semantic theories [74], for example represented by textual programming languages. Although current programming languages also represent the structure of what is represented, there exists a huge gap between the textual representation of a program and the program as a functional unity. Object oriented classes were successful in improving the representation, but only modestly.

```
import java.math.*;
public interface SetIntersectComponent extends Component
{
    public void setA(Set a);
    public void setB(Set b);
    public int getIntersectResult();
}
```

Fregean representation



analogical representation

Figure 5-4 Fregean versus analogical representation

One of the goals of Visual Programming is to enable humans to program without having been trained to be able to read and write the textual representation of a program (accessibility). To that end, most VPL use analogical representations whose structures have functionalities that are as close as possible to those of the structures represented. An example for the two representations is given in Figure 5-4. Both parts of the figure de-

scribe the same functionality. The left part uses a Fregean representation, using a Java interface as a description, whereas the right part describes the program in analogical "terms". Both representations need to be decoded. But it is assumed that the Fregean representation needs more abstract knowledge (i.e. about the programming language Java) to be decoded than the right part, whose representation requires knowledge about mathematical symbols (Venn diagrams [137, 155] in this case).

A very similar approach is presented in [137], where linguistic and diagrammatical representations of logic are compared. The key distinction is that the diagrammatical representation relies on the reader's perceptual inferences, whereas the linguistic representation relies more on the conventions of the associated representation system known to the reader.

The theories mentioned above are mainly concerned with the information transfer from the representation to the user. Fairly recently, researchers have investigated in user studies [122, 160] on how programs or parts thereof are represented mentally. In these studies, programmers were asked to describe their mental imagery during the design of programs, solving given problem cases. The importance of using mental imagery has been shown previously [81]. However, it has been found that the mental imagery varied considerably between test subjects (e.g. most used textual and graphical representations intermixed, but some exclusively used a textual representation). Despite that, a few common elements in using mental imagery could be extracted [122]:

- Mental imagery was dynamic and subject to control by the user.
- Information was not uniformly distributed, but focused and more detailed at the point the user chose to concentrate on.
- Information was spread over multiple levels.
- Areas that had not yet been solved were marked specifically, using for example a fuzzy representation.
- Users readily used multiple representations for a given structure.
- Most of the structures were labeled textually, suggesting a multimodal thought process.

Although these findings suggest a smaller translation distance from mental representation to visual programming language representation than textual representation, this point is still disputed [77].

Recently, papers that investigate visual programming in specific contexts have indicated that most of the advantages and disadvantages of visual programming are closely related to the area in which they are used

[117]. One of these areas where the usability of VPLs compared to TPLs varies with the context is whether they are used in programming-in-the-small or programming-in-the-large [76]. Programming-in-the-large is the activity of programming systems that consist of many modules, or small programs, interacting with each other [42]. Programming-in-the-small is about exclusively writing these modules. Software engineering as a discipline originated from the realization that the two programming styles are handled fundamentally different: "structuring a large collection of modules to form a 'system' is an essentially distinct and different intellectual activity from that of constructing the individual modules." [42]

Despite the fact that most of the visual programming research effort has been concentrated on visual programming languages for programming-in-the-small, they do not seem to be well suited for the task. To compare the performance of VPLs and TPLs on a low level, [117] defines a graphic metric (using graphic tokens) based on the assumption that there are three ways (tokens) of combining graphic units: adjoinment, when graphical units touch each other, linkage, where units are linked through special graphical devices; and containment, where one unit is enclosed in another. Although the graphical representation handles graph relations better (i.e. has a lower token count), the textual representation is much better suited to tree-like relations (e.g. "4 * 3 + 1") which occur very often at this programming level.

Stemming from the same source is the problem of the large space used to display a program [96]. Possible solutions for this problem are scrolling, introducing an abstraction mechanism like a hierarchical structure, where only the current level is visible, or as presented in [76], a zooming mechanism, that only displays features around a certain depth. There are a few advantages, for example that fewer concepts are required to program: "de-emphasizing issues of syntax and providing a higher level of abstraction" [115]. The programmer does not have to deal with pointers, memory allocation, and the like. Still, it seems that on a low programming level, the mental abstractions performed by the user are closer to the textual representation than to a graphical representation.

On the other hand, VPLs appear to perform very well in the area of programming-in-the-large, an example of which is the object oriented programming paradigm. Application examples for this type of VPL are described in [21, 34, 76]. The same graphic metric that favors TPLs in programming-in-the-small, seems to favor VPLs in programming-in-the-

large. The reason for this is that with programming-in-the-large, the semantic expressions are often graph-based. Also, since the modules are most of the time encapsulated in files, the textual expression of relations is costly, compared to the graphic representation between modules, which can be expressed using one of the abovementioned three types of relations [117]. VPL environments utilizing high-level basic elements have been quite successful in context of a specific application domain, which is often small. In the last decade, a few VPL environments have been researched, which use user-customizable high-level components, for example [76, 96]. Many have been developed commercially, with mixed success, for example Sanscript, or Khoros.

Despite the fact that many VPLs have been designed and implemented, and generally favored by test users [160], usage has been marginal [58]. A reason for this might be the fact that changing software design has a high cost-of-entry due to investments made in older, better established paradigms, and also due to the fact that software developers are hesitant to learn new paradigms.

5.2. Auto-ID System Development Process

In a specific application scenario (e.g., a retail store or a whole supply chain), the Auto-ID infrastructure consisting of its hardware and software components has to be initially instantiated. This instantiation involves the following categories of tasks:

1. Setup of hardware, that is the readers, sensors, actuators and servers
2. Configuration of hardware, for example reader parameters to tune the anti-collision algorithm or the dense reader mode
3. Setup of software, for example installing databases or software components
4. Configuration of software, for example, definition of objects in the OMS or filters in the Filtering and Collection middleware

The first category of tasks has to be performed in the physical world, for example readers with their antennas have to be deployed at the required locations and connected to a network. The second category is typically done by the system integrator that provides the reader and sensor which are Auto-ID technology specialists. The third category involves the setup of standard software components which is typically done by system administrators of the involved parties of the application scenario.

In the proposed Auto-ID Application Development Process we concentrate on the fourth category. The configuration of the Auto-ID software components is specific for the application scenario and brings the Auto-ID infrastructure in a state where it can be used by the applications. The software components in question are:

- Reader and sensor software
- Filtering and aggregation components such as Event Layer or EPCglobal Filtering and Collection middleware
- Object Monitoring System (OMS)
- Lookup Service (LUS)

The specific configuration and instantiation tasks for the software components, grouped by the layers of an Auto-ID infrastructure, are listed in Table 5-1.

Table 5-1 Configuration and instantiation tasks for the Auto-ID infrastructure software components

Layer	Taks
Hardware	• Setup and configuration of readers, sensors and actuators with their correct IDs and registration of these IDs with the LUS • Definition of read cycles and reporting of readers (e.g. how often and for how long will a read cycle be performed) • Definition of measuring and reporting of sensors
Filtering and Aggregation	• Setup and configuration of filtering and aggregation components with their correct IDs and registration of these IDs with the LUS • Definition and configuration of filtering, aggregation and reporting of the filtering and aggregation components (e.g. event cycle definitions for Filtering & Collection components or filter chain definitions for Event Layer components) • Definition of field names for user memory on Auto-ID tags • Configuration of the connections between filtering and aggregation components and the readers and sensors that should report to these components

Layer	Taks
Data Enrichment, Representation and Persistency	• Definition of the static and semi-static objects (e.g. buildings, rooms or shelves) and their containment relationships in the Auto-ID Object Model instance (i.e. the object tree) • Definition of mobile objects (e.g. fork lifters or shopping carts) • Definition of existing functions of objects • Implementation and integration of application specific functions • Definition of existing properties of objects and their property sources • Implementation and integration of application specific property sources (e.g. properties taken from legacy information systems) • Definition of the readers, sensors and actuators in the OMS and their links to objects in the object tree • Definitions of object sets that are used in the business processes • Definitions of all business processes specific for the application scenario which are managed by the OMS (i.e. definitions of state machines, state transition conditions and actions) • Implementation and integration of application specific actions (e.g. custom information system reporting or notifications to mobile devices) • Configuration of connections to filtering and aggregation components such as the Event Layer or EPCglobal compliant Filtering and Collection middlewares including the definition of user memory fields on Auto-ID tags
Applications and Information Systems	• Definition of the configuration of the structure (i.e. the schema) of the look-up service (e.g. LDAP schema to store the different component records)

The Auto-ID Application Development Process defines certain steps and provides visual tools to support non-software developers in the differ-

ent tasks over the lifecycle of an Auto-ID infrastructure (see Figure 5-5). The process corresponds to the application engineering steps in the software development based on domain engineering described in tion 5.1.1 (see also Figure 5-3).

In the *analysis phase*, the application scenario is analyzed by Application Analysts. The result of the analysis is an Application Specification that contains an extensive, semi-formal specification of the application including the business processes that are triggered by the physical objects in the real world, the locations of interest and their hierarchical structure, and properties of physical objects that are of interest to the application or the business processes. The analysis corresponds to business analysis and requirements engineering in traditional software engineering.

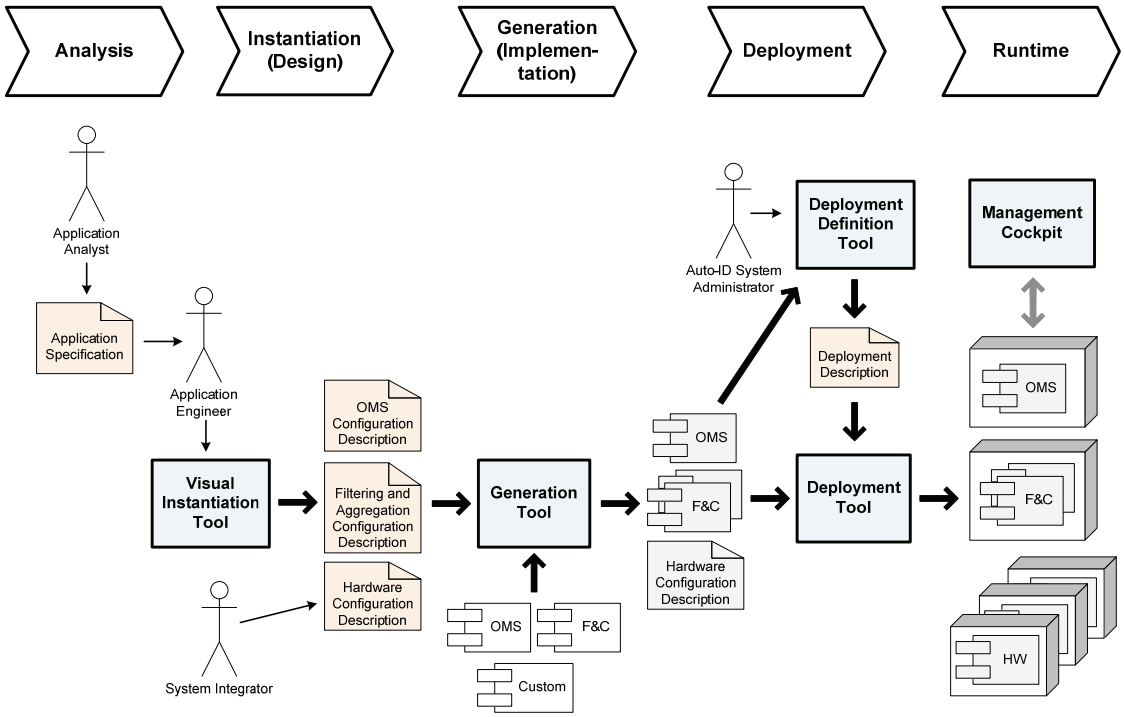


Figure 5-5 Auto-ID Application Development Process

This Application Specification is the input for the Application Engineer in the *instantiation phase* (i.e., the design phase in traditional software engineering). The Application Engineer formally specifies the instantiation of the Auto-ID infrastructure using the Visual Instantiation Tool (VIT) (see section 5.2.1). The VIT provides the Application Engineer with a concrete representation of the application definitions (e.g. the Auto-ID Object Model instance is represented as a floor plan) and no soft-

ware development knowledge is needed. The output of the VIT is a description of the configurations of the OMS and the filtering and aggregation components. In addition, the System Integrator provides the configuration descriptions of the readers, sensors and actuators.

These configuration descriptions and the bare software components are the input for the Generation Tool (see section 5.2.2) in the *generation phase* (i.e., in traditional software engineering the implementation phase). The Generation Tool automatically generates instantiated software components, that is software components that contain the specific configuration, and a description of the hardware configuration in the format required for the Deployment Tool.

In the deployment phase, the Auto-ID System Administrator who has the information about the different servers and the deployed hardware defines the actual deployment of the software components using the Visual Deployment Definition Tool (VDDT) (see section 5.2.3). The VDDT takes the information about the instantiated software components as input and provides the Auto-ID System Administrator with a visual toolkit to create a deployment plan for the Auto-ID Infrastructure. The output of the VDDT is the Deployment Description. The Deployment Tool takes the Deployment Description as input and automatically deploys the instantiated software components to their servers. This requires a deployment infrastructure in place on the servers. The Auto-ID Infrastructure is now fully instantiated, deployed and ready for the application to be used.

During the runtime phase, a visual tool similar to the VIT, provides a Management Cockpit to interested actors. For example, the store manager in a retail store scenario can use the tool to get an overview of the physical objects and their movements in the store, the running business process and their reporting actions. The Management Cockpit was not in the scope of this thesis and is not described further.

5.2.1. Visual Instantiation Tool

The VIT is the main tool in the Auto-ID System Development Process. The Application Engineer can perform most of the instantiation and configuration tasks required for the OMS (see Table 5-2). The Application Engineer does not need any programming skills or software development knowledge.

Table 5-2 Configuration and instantiation tasks supported by VIT

Tasks (see Table 5-1)	Function in the VIT
• Definition of the static and semi-static objects and their containment relationships	Draw and define objects in Floor Plan View or define Objects in Tree View
• Definition of mobile objects	Define objects in Tree View
• Definition of existing functions of objects • Integration of application specific functions (implemented outside of VIT)	Add functions to selected object in Functions View
• Definition of existing properties of objects and their property sources • Integration of application specific property sources (implemented outside of VIT)	Add properties and property sources to selected object in Property View
• Definition of the readers, sensors and actuators in the OMS and their links to objects in the object tree	Add reader, sensor or actuator to objects in Floor Plan or Tree View
• Definitions of object sets that are used in the business processes	Add sets with Set Management
• Definitions of all business processes specific for the application scenario which are managed by the OMS • Integration of application specific actions (implemented outside of VIT)	Add business process including state and transition definitions to selected object using Business Process Management

As stated in section 5.1.2, the main goals of visual programming are to make programming more accessible to some particular audience and to improve the correctness and speed with which people perform programming tasks. The VIT fulfills these goals for application engineers. It relieves them from defining extensive XML configuration files which require knowledge about their complex XML schemas. Using XML-based editors would improve the syntactic correctness, however, the semantic

correctness would still suffer. The accessibility of the visual representation using a floor plan of the application scenario supported by a tree view that emphasizes the object hierarchy is very high and many tasks can be fulfilled in less time.

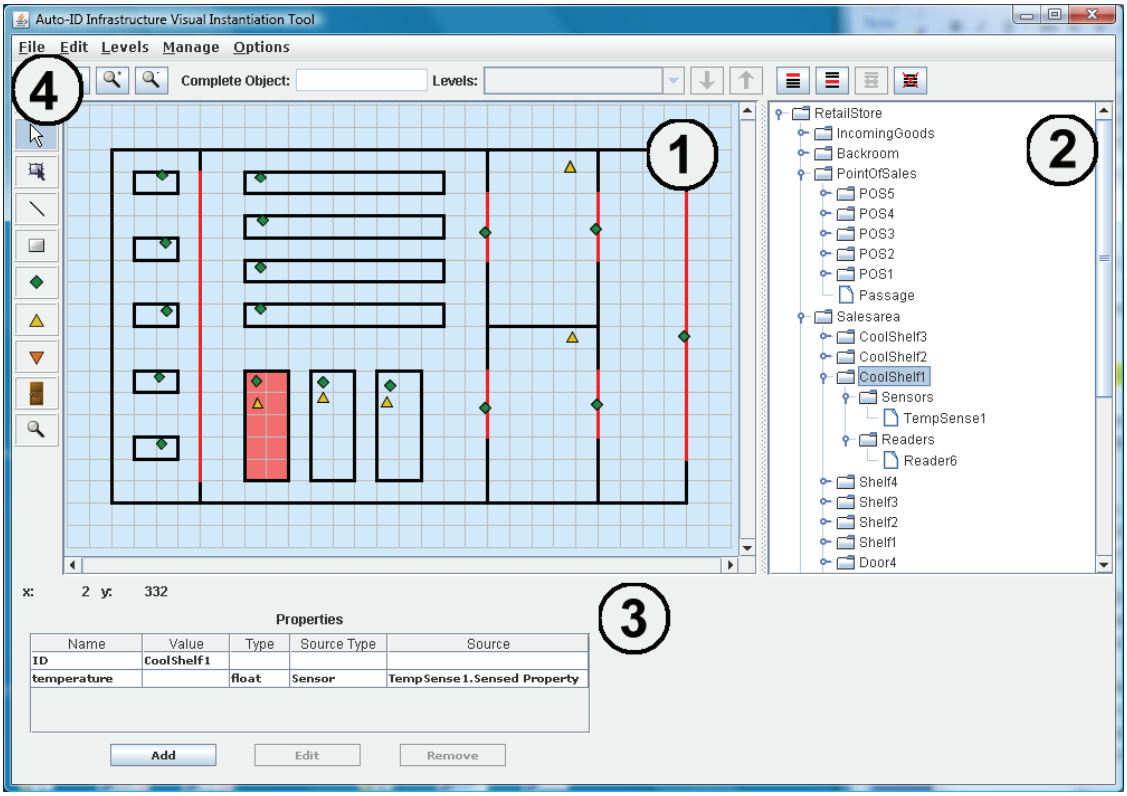


Figure 5-6 GUI of the Visual Instantiation Tool

The VIT uses the above mentioned strategies (see section 5.1.2) of visual programming: The visual representation of the Auto-ID Object Model instance provides a *concrete* representation where the application engineer can *directly* manipulate the objects and their relationships. Some aspects of semantics (e.g. object containment relations or reader linked to objects) are inferred without the application engineer needing to *explicitly* state them. During the manipulation of the model instance, the application engineer gets immediate semantic visual feedback (i.e., *liveness*). For example, adding or removing an object is shown both in the floor plan and tree view. The definition of the model instance becomes *simpler* since the application engineer can think in application concepts and does not need to worry too much about XML specific syntax or semantics.

Figure 5-6 visualizes the graphical user interface (GUI) of the VIT. The GUI is structured in the following areas: (1) The *Floor Plan View*, (2) the *Tree View*, (3) the *Properties and Functions View*, and (4) functions of the VIT provided as menus or toolbar actions. In addition, the *Set and Business Process Management* is provided in dialogs accessible via menu items.

Floor Plan View

The Floor Plan View visualizes the Auto-ID Object Model instance as a floor plan. The view allows the user to intuitively draw a 2-dimensional floor plan of a building or other structure.

The user can modify the plan by dragging or deleting previously drawn line segments, defining objects through an area enclosed by line segments, and add readers, sensors or actuators by dragging the corresponding symbol onto a defined object. The VIT automatically determines the containment relationship of objects while drawing and defining objects. For example, drawing a room and then drawing three shelves inside the room will result in automatically placing the three shelves inside the room in the tree hierarchy.

Although the view represents a 2-dimensional view, the 3rd dimension is supported by objects that can have different levels in the 3rd dimension (e.g., a shelf with different levels of storage). The visible object on the view then stands for the currently chosen level of the complete object. A complete object represents a container that includes all levels of an object. Levels in the Auto-ID Objects Model instance are simply represented as child objects of the complete object.

Tree View

The Tree View shows the Auto-ID Object Model instance emphasizing the object hierarchy. This view presents the user with complementary information since objects are represented by their object ID and clearly showing the object containment relationships.

The Tree View also shows the links to readers, sensors or actuators of objects. Changes in the Floor Plan View and Tree View are synchronized. However, not all functionality of the Floor Plan View is available in the Tree View due to the different representation (e.g., objects can only be defined through drawing them on the Floor Plan View).

Properties and Functions View

The Properties and Functions View provides a tabular view on the properties and functions of the currently selected object. The user can add new properties and define the property source and modify existing properties. In a similar manner, the user can add and modify the functions of an object.

Menu and Toolbar Actions

Several functions of the VIT are provided as menu or toolbar items. There are several menus that group functions such as file functions (e.g. load and save of a model instance), typical edit functions such as copy, cut, paste and remove, management of levels of objects, starting of set and business process management, and user options.

The two toolbars contain quick access buttons that support carrying out actions with only one click and are short cuts for frequently used actions. The toolbar on the left of the Floor Plan View includes actions to select and define objects and add links to readers, sensors and actuators. The toolbar on the top provides quick access to load, save, zooming of the floor plan and level management of the currently selected object.

In addition, a context pop-up menu exists for actions on the Floor Plan View.

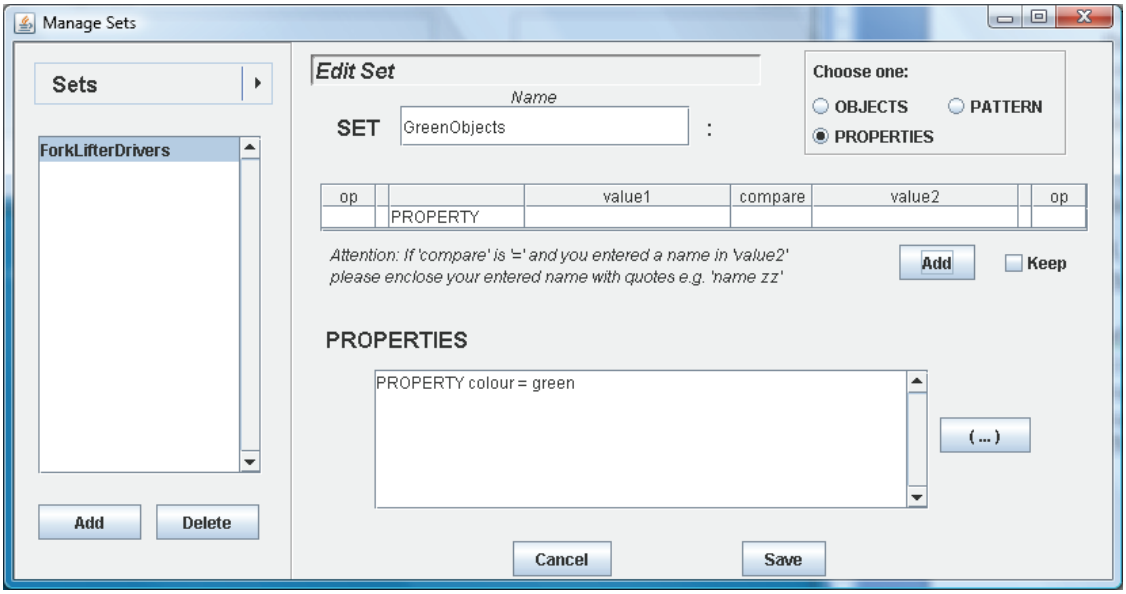


Figure 5-7 Set Management dialog

Set and Business Process Management

Since the sets in the Auto-ID Object Model are not bound to an object, the Set Management dialog is opened via a menu item only. The dialog allows adding and modifying object, pattern and property sets. Figure 5-7 shows adding a property set. Defining the property condition is supported by a constructor that enforces the syntactic constraints of the set definition language. In addition, the definition can always be modified directly in the textual expert view of the set definition.

The Business Process Management dialog can be opened via a menu or context menu item and applies to the currently selected object. The states and their transitions can be graphically defined similar to a UML state diagram. The conditions of the state transitions can be defined using the definition language for state transition condition either in textual expert mode that gives complete freedom or using a constructor that enforces the syntactic constraints of the language.

op		level	value	compare	value
NOT		ALL		ISSUPERSET	ObjectShipment1020

Figure 5-8 Conststructing a set comparison expression

The constructor allows building a condition using expressions, logical operators and parentheses as building blocks. The expressions can be defined similar to the conditions in the property set definition. The different parts of the expression can be chosen from a drop down list that either depends on syntax of the expression (e.g., the possible compare operators) or other definitions of the user (e.g., the existing sets of objects). For example, Figure 5-8 illustrates how to construct the following set comparison expression:

```
NOT (ALL ISSUPERSET ObjectShipment1020)
```

5.2.2. Generation Tool

The Generation Tool is not a visual tool that requires user interaction. It can be started in the workflow of the Auto-ID Application Development Process either directly from the VIT or manually. The Generation Tool takes two kinds of input: First, the output of the VIT and additional configuration descriptions (i.e., about readers, sensors and actuators), and second, the OMS, Filtering and Aggregation, and custom made software components. The tool then generates fully configured and instantiated software components that are ready to be deployed and hardware configu-

ration descriptions that will be used to configure already deployed hardware (i.e., readers, sensors and actuators) if necessary.

In the concepts of the Domain and Application Engineering (see Figure 5-3), the Generation Tool provides the step from the domain implementation (i.e., the OMS and Filtering and Aggregation software components) via the integration step (i.e., configuring the domain components) towards the construction of the final application (i.e., integrating the custom made components). The Generation Tool contains the needed configuration knowledge such as default settings, default dependencies and construction rules (see Figure 5-2) to build the final application, that is, the fully instantiated and configured components.

The prototype implementation of the Generation Tool is using the technology of Extensible Stylesheet Language Transformation (XSLT) [18] to transform the XML output of the VIT into an XML document that acts as the configuration of the OMS. The configuration is packaged in the Java archive (JAR) files of the OMS. In addition, custom components are configured and integrated into the JAR files of the OMS. Based on the hardware configuration description, for each reader, sensor and actuator that needs to be configured a configuration description is created that is used by the Deployment Tool.

Table 5-3 Configuration and instantiation tasks supported by the Deployment Definition Tool

Taks	Function in Deployment Definition Tool
• Configuration of the connections between filtering and aggregation components and the readers and sensors that should report to these components	Define Auto-ID Connection between components
• Configuration of connections of the filtering and aggregation components and the OMS to which the component should report	Define Auto-ID Connection between components

5.2.3. Deployment Definition and Deployment Tool

After the Generation Tool, all software components are configured, packaged and ready for deployment. The Deployment Definition Tool provides a visual interface that allows the Auto-ID System Administrator to

specify the deployment of the Auto-ID infrastructure. The deployment information includes two kind of information: First, for each software component the server (i.e., physical node) to which it should be deployed. Second, all the connections between the software components, that means, the definition to which higher component a specific component should report its data (see Table 5-3).

The Deployment Definition Tool is based on the Graphical Instantiation Environment (GIE) which was developed at ETH. The GIE provides an extensible framework to create graphical editors for components and connections between these components. The Deployment Definition Tool extends the GIE by providing all the visual and syntactic information about the components and connection that exist in an Auto-ID infrastructure. The GUI of the Deployment Definition Tool (see Figure 5-9) mainly consists of a toolbox with the available components and connections and a panel that contains the deployment as a visual representation of the component instances and their connection.

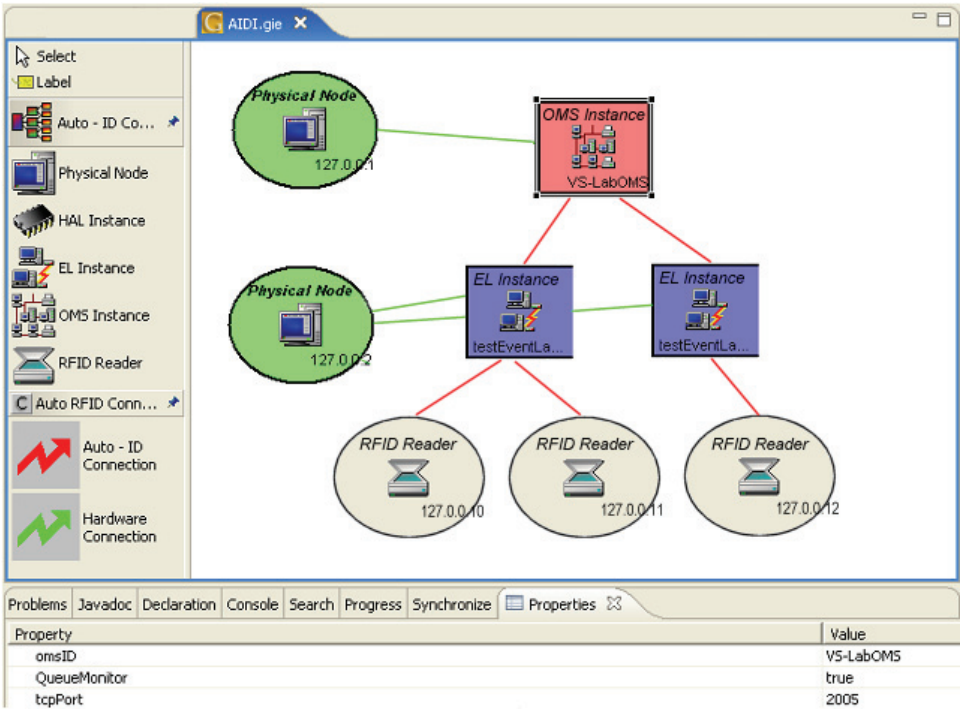


Figure 5-9 GUI of the Deployment Definition Tool

The Deployment Definition Tool enforces specific constraints on the deployment definition, for example, software components can only have Hardware Connections to Physical Nodes or RFID Readers can only be

connected with Auto-ID Connections to EL Instances. In the example, shown in Figure 5-9, the OMS will be deployed on a server and the two EL instances on another server. The same deployment as an UML deployment diagram is shown in Figure 5-10.

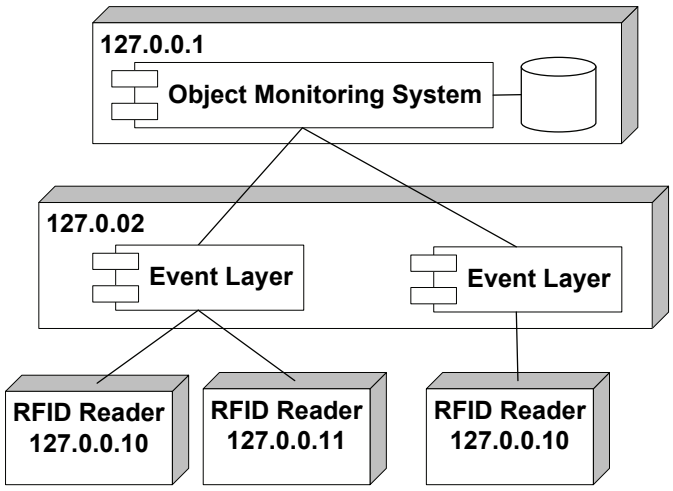


Figure 5-10 Example deployment of Deployment Tool

The Deployment Tool takes the information from the Deployment Definition Tool and performs the actual deployment of the configured and instantiated software components. In addition, using the hardware configuration description, all readers, sensors and actuators are configured. After the Deployment Tool successfully finished its deployment, the whole Auto-ID infrastructure is set-up and in an initial state, ready to be used by the applications. In our prototype implementation, the Deployment Tool uses the Java Web Start technology [8] to deploy and start the software components automatically.

5.3. Related Work

There are several related approaches based on generative and visual programming concepts such as domain-oriented design environments or component-based construction kits. The general concepts related to our approach have been described in sections 5.1.1 and 5.1.2. The following selected research projects or visual programming environments are related to our approach.

Another visual programming environment is, for example, the *Peter System* [116], a visual programming tool for programmers and non-

programmers, in which programs are put together on a tree structure from a variety of basic components.

UML modeling tools can also be seen as visual programming environments with which a user can instantiate a UML model (e.g. a UML class diagram) based on the UML meta model [118]. In many cases, even source code or source code skeletons can be created based on visual UML definitions.

Many *system modeling environments* are based on visual programming environments, for example, Simulink [14], an environment for multi-domain simulation and Model-Based Design for dynamic and embedded systems. It is an interactive graphical tool for modeling, simulating, and analyzing dynamic, multi domain systems. It lets the user describe, simulate, evaluate, and refine a system's behavior through standard and custom block libraries. Simulink is used as a system modeling tool in a variety of applications such as unmanned flight control systems, health risk predictions, agricultural research, car engineering, satellite software development and chip design.

The Board *Software Instantiation Environment (OBSIE)* [35] tackles the problem of developing systems for physical devices or facilities: Engineers who are application specialists specify the control system of a device but the implementation of the control system if performed by software specialist based on the specification. The information gap between the application specialist and the software specialist can lead to misunderstandings and is error-prone and cost-intensive.

The OBSIE is a generative programming environment, which allows the application specialist, given a clearly defined framework, whose components are preferably implemented as black box components, to compose them together and instantiate a new program without the need to be a software specialist. The OBSIE has a development process that clearly defines the workflow of domain and application engineering and the involved tools and artifacts. The domain components are the components of the existing framework that can be further customized (i.e., the framework be instantiated) to build a concrete application. The OBSIE is presented with a specific framework, the Attitude and Orbit Control Systems framework (AOCS) used for satellite control systems.

The OBSIE is based on the JavaBean component model using different XML schemas to describe the different steps in the customization process. It also uses a visual programming environment based on the Java Bean

Builder (see Figure 5-11) to customize and connect the components. The customization information is then used in a generator based on XLST to create instantiation code which together with the framework components composes the application.

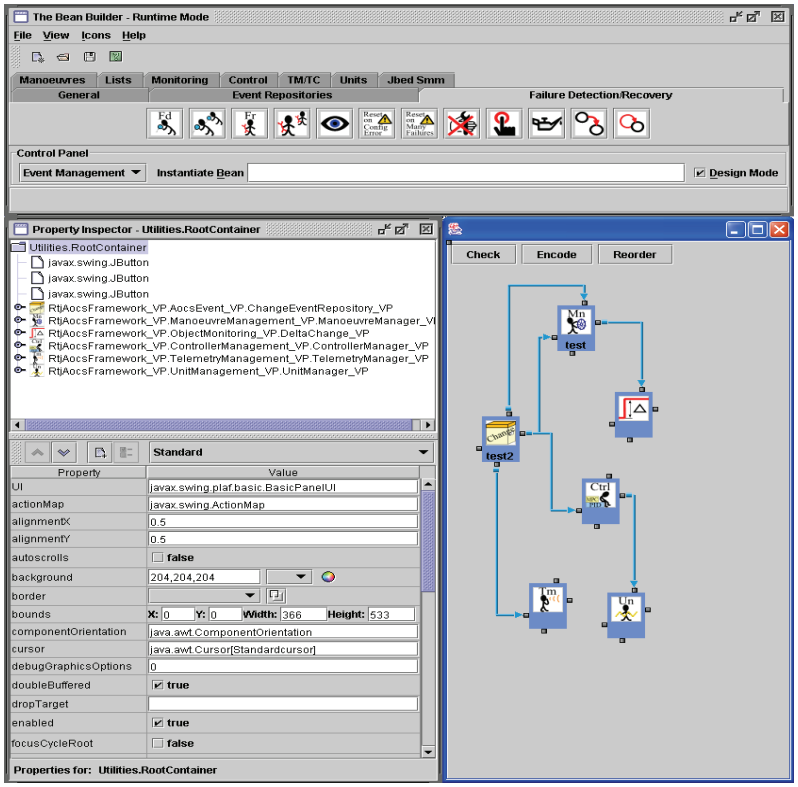


Figure 5-11 OBS Bean Bilder (from [35])

The *programming environment for the LEGO Mindstorms* [9] robots is a graphical editor and code generator to visually create LEGO Robot Command System (RCX) code. First, a visual representation of the program is designed in the graphical editor, and then the textual representation (the code) is generated and used in the RCX. The RCX building block is a programmable microprocessor, which is able to execute RCX code and to which several sensors and actuators can be connected. The new generation of Lego Mindstorms has a slightly different and enhanced programming environment based partly on LabView [92].

To represent parts of a program design, the programming environment uses special building blocks whose look resembles normal LEGO blocks as the concrete representation of the program. A wide variety of predefined blocks exists to cover a program (see Figure 5-12). Blocks are drag-and-dropped onto the lower LEGO connection part of existing

blocks. The textual representation, an ASCII representation of the RCX code, is structurally a direct translation from the visual representation into a textual representation which resembles C code.

Figure 5-12 . LEGO Mindstorms Programming Environment

However, even if the visual representation is quite directly mapped to its textual representation, this does not mean that the representations are equal from a user interface standpoint. The program structure in visual form is much faster acquisitioned due to the right level of detail shown to the user, the use of color to distinguish elements and the clear definitions of block connectivity and program flow direction.

6. Case Studies

To evaluate the proposed Auto-ID Object Model and visual tool-based approach, we applied the Auto-ID Application Development Process to several different application scenarios. Our approach is not only suitable for supply chain applications, but a variety of different applications based on the Auto-ID Object Model. The applications and an evaluation are presented in the following case studies acting as a proof-of-concept of our approach:

1. Retail store supply chain application
2. Smart Medicine Shelf in hospitals
3. Tool management in aircraft maintenance application
4. Augmented Knight's Castle, a pervasive computing play set

Since the retail store supply chain application is presented as the example application already in section 4 and 5, it is not further presented in this section.

6.1. Smart Medicine Shelf

The Smart Medicine Shelf is an automated shelf in a health care environment (e.g., in a hospital) that keeps track of different kinds of medication that require different conditions (e.g. certain vaccines have to be cooled) and require different access rights (e.g. a nurse is not allowed to access certain drugs).

The objective of using the Smart Medicine Shelf is to improve stock management of drugs in a hospital. The Smart Medicine Shelf frequently informs a hospital ERP system about its inventory and reports exceptional states to the same system or other interested applications. The benefits of the Smart Medicine Shelf are among others:

- The availability of medication can be improved and out-of-stock situations can be avoided. The Smart Medicine Shelf reports low number of drugs to the ERP system that automatically invokes replenishment orders.

- The Smart Medicine Shelf avoids the cumbersome process of checking the expiry date on each folding box and reduces the number of drugs that has to be thrown away due to reached expiry dates. A warning is sent to the responsible medical personnel such as the pharmacists if drugs are about to expire soon.
- A user interface can indicate drugs that have been recently recalled and which are still contained in the shelves. In addition, applications monitoring recalled drugs can inform medical personnel to remove them from the Smart Medicine Shelf.
- Access to restricted drugs can be enforced by only unlocking a special compartment of the shelf for authorized medical personnel only. This requires that all medical personnel can be identified, for example by Auto-ID tags in their badges.
- Since medication has to be stored under certain conditions (e.g., certain temperature or humidity), sensors monitor the correct conditions and report warnings if they are violated.
- Synchronizing different Smart Medicine Shelves in a hospital even increases the benefits. For example, medications that are needed in a certain department which are not available there might be available in another department. Instead of reordering the medications, they can be fetched from the Smart Medicine Shelf of the other department.

Figure 6-1 Setup of the Smart Medicine Shelf Application

The principle of monitoring medications using RFID and providing the above listed benefits was developed in a prototype of the Smart Medicine Shelf [68, 99] (see Figure 6-1). It can be extended to a larger setup used in hospitals. The Smart Medicine Shelf consists of four main locations resp. compartments as shown in the instance of the Auto-ID Object Model in Figure 6-2 (readers are marked by green diamonds, sensors by yellow triangles and actuators by orange triangles on the head):

- *Surrounding*. The location where medical personnel is identified. In the example a medical doctor with the ID Doctor-1020 is located at the surrounding.
- *Storage*. The shelves for drugs that are neither cooled nor have restricted access. In the example, the storage consist of two shelves with different drugs (IDs are given as pseudo EPCs).
- *Cooled Storage*. The shelves contain drugs that have to be stored at lower temperatures (e.g. vaccines).
- *Protected Storage*. The shelves contain the drugs that only authorized personnel can access. The storage is protected by a lock (i.e., the actuator lock).

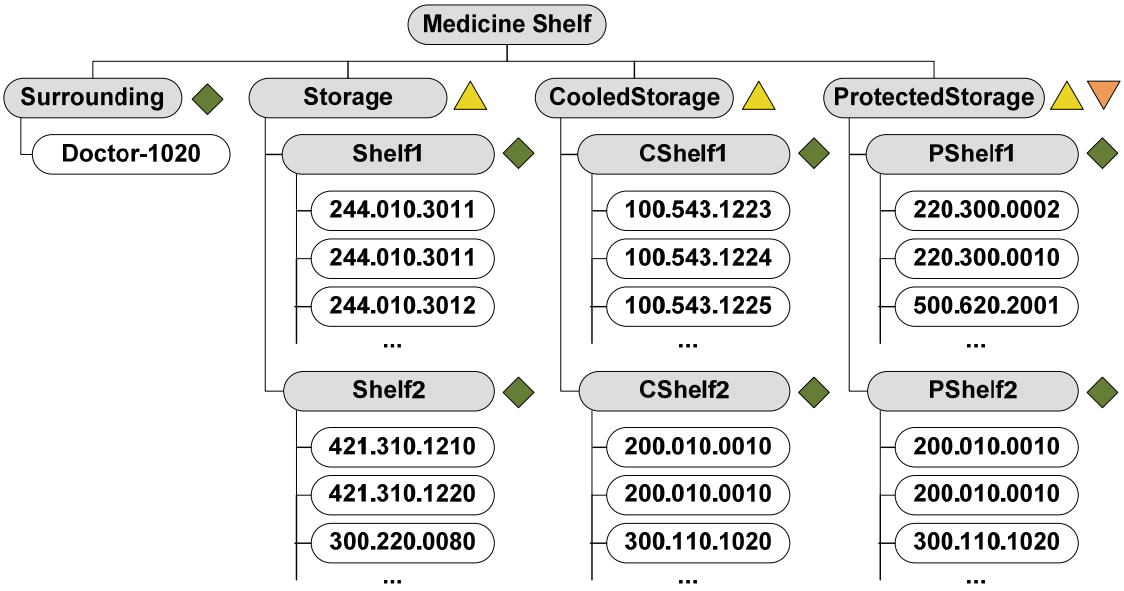


Figure 6-2 Auto-ID Object Model instance for Smart Medicine Shelf

Following the Auto-ID Application Development Process (see section 5.2), the model instance (see Figure 6-2) was created with the VIT (see Figure 6-3). In this case, the metaphor of compartments or boxes was used as a visual representation instead of a floor plan.

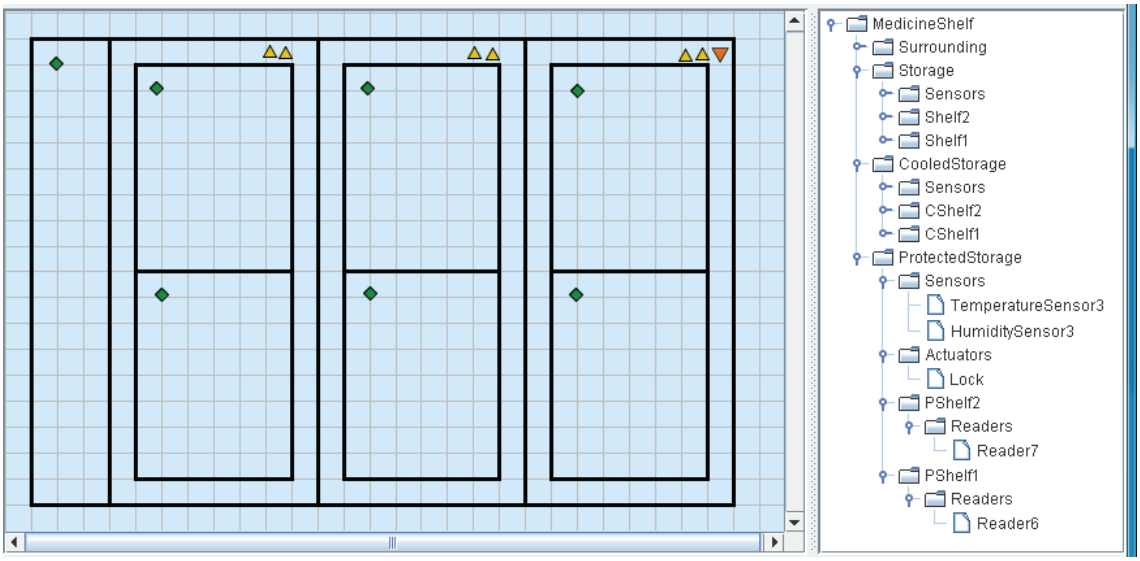


Figure 6-3 Instantiating the Smart Medicine Shelf with the VIT

The Auto-ID triggered business processes that are part of the model instance to monitor the drugs and their state and report exceptional states to the ERP system and applications are listed in Table 6-1, together with the object in the model instance to which they are connected.

Table 6-1 Business processes of the Smart Medical Shelf

Object	Connected Business Processes
Surrounding	Restrict Access
Storage	Check Expiry Date Monitor Temperature and Humidity (15º C) Check Recalled Drugs
CooledStorage	Check Expiry Date Monitor Temperature and Humidity (5º C) Check Recalled Drugs
ProtectedStorage	Check Expiry Date Monitor Temperature and Humidity (15º C) Check Recalled Drugs

The business process definitions of three business processes are shown in Figure 6-4. The business process “Restricted Access” monitors the child objects of the object Surrounding. If at least one object belonging to the set “AuthorizedPersonnel” is in the set CHILDREN then the lock securing the ProtectedStorage is opened (i.e., if a doctor is located at the

Surrounding). Likewise if no doctor is located at the Surrounding any more, the Shelf is about to lock after 1 minute. The set AuthorizedPersonnel is defined as a pattern set, where all authorized personnel has an ID that starts with "Doctor". In a real-world deployment, the personnel ID might contain a different pattern that identifies doctors.

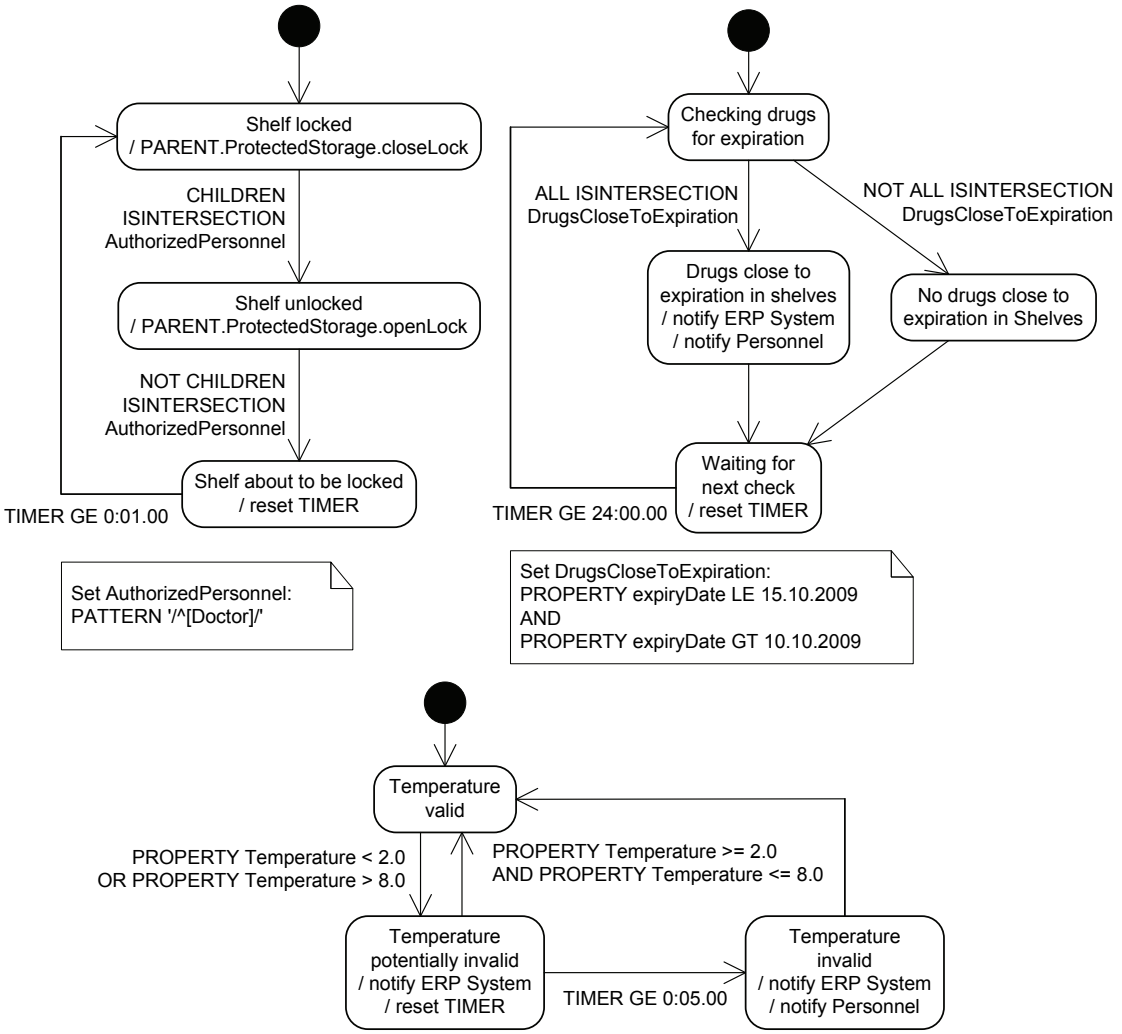


Figure 6-4 Business processes of the Smart Medical Shelf

The business process "Check Expiry Date" starts with a state where all objects in the shelves are checked if they contain at least one object that belongs to the set "DrugsCloseToExpiration". This set is updated by the ERP system each day and contains objects whose property "expiryDate" is in the range of five days from the day of checking (i.e., the expiry date is about to be reached). If there are drugs detected that are close to expiration then the ERP system is notified with a report containing all objects in

question. In addition, an application that generates a message to the responsible personnel is notified. The following state resets a timer and waits for one day until a next check is performed which means that expiration is only checked once a day.

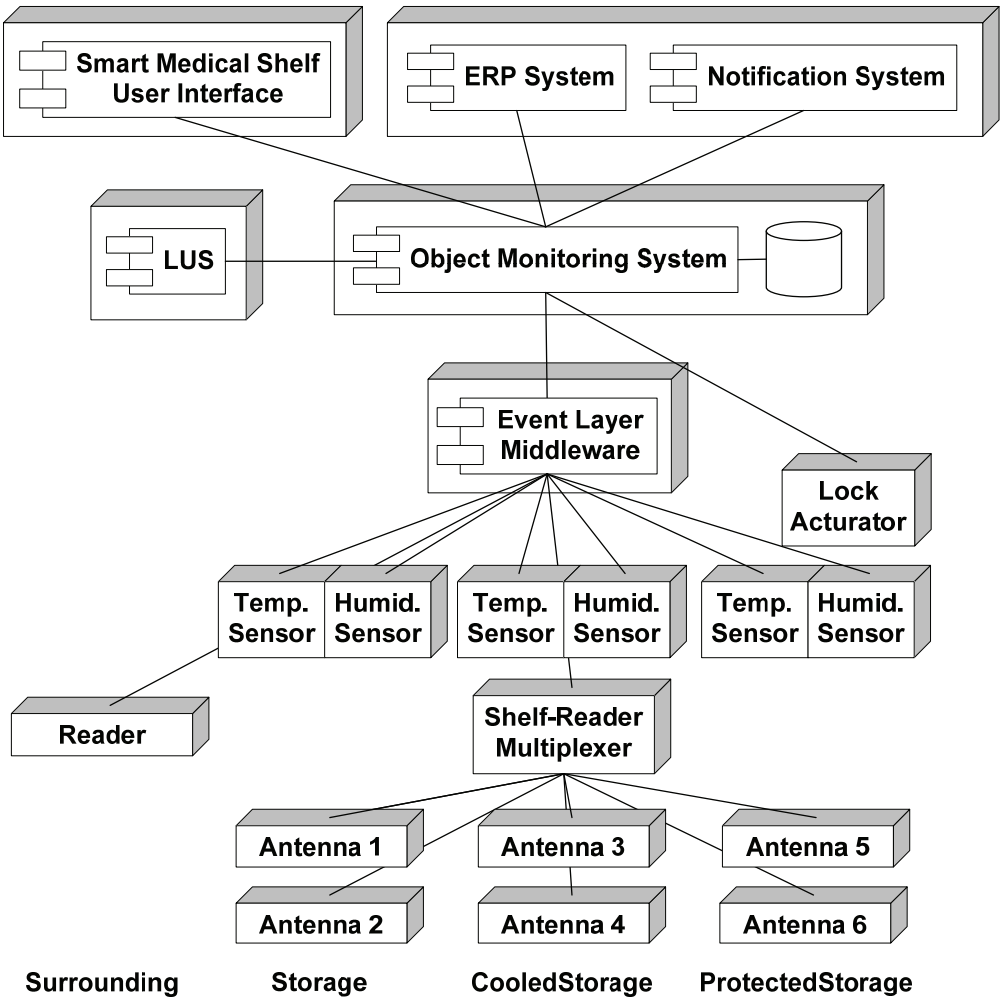


Figure 6-5 Smart Medical Shelf deployment

The correct temperature range in the CooledStorage is checked by the business case “Monitor Temperature”. If the property “Temperature” of the object CooledStorage leaves a defined temperature range of 2º to 8º Celsius a state is set that indicates a potential temperature violation which also notifies the ERP system for audit purposes. If after 5 minutes the temperature is still outside the required range, the temperature is set as invalid. The ERP system is notified and also responsible personnel to check the cause of the temperature violation. In a similar manner, the humidity can be monitored.

The business process “Check Recalled Drugs” checks if objects in all shelves contain at least one object belonging to the set “RecalledDrugs” which is frequently update by the ERP system. This check is performed once an hour modeled in a similar way as states and transitions as the check once a day in the business process DrugsCloseToExpiration.

The deployment of the Smart Medical Shelf components is performed using the Deployment Definition Tool and the Deployment Tool. Figure 6-5 illustrates the final deployment. In this case the Event Layer component and the OMS component are deployed on different machines since the filtering and aggregation of the Event Layer could also be used for other existing Smart Medical Shelf instances in the same application scenario. Only one reader is used to monitor the shelves of the different compartments using a multiplexer that scans through the different antennas. In this scenario one antenna monitors one shelf. The Surrounding is monitored by another reader since reading the badges of the personnel involves another RFID technology than detecting the drugs.

6.2. Tool Management in Aircraft Maintenance

The objective of the tool management in aircraft maintenance application is to improve the overall maintenance, repair, and overhaul (MRO) process for aircrafts by providing a better management of movable assets, that is the tools used by the mechanics. Since strict regulations define requirements for quality, safety, and documentation of the MRO process, improving the tool management can lead to a more secure and cost efficient process.

The tool management in aircraft maintenance application [68, 104, 103, 147] consists of several movable toolboxes of the mechanics which include a great variety of tools (called the Smart Tool Box) and one tool inventory where mechanics can check-out special tools they infrequently need (called the Smart Tool Inventory). Both the tool boxes and the tool inventory monitor the tools they contain and provide services to improve the MRO process. The tool boxes report exceptional states (e.g., missing tools) to the ERP system. The tool inventory is closely coupled with the inventory management system of the ERP system. The benefits of the tool management in aircraft maintenance application are among others:

- The Smart Tool Box (see Figure 6-6, left) eliminates the cumbersome process of performing the required routine and base complete-

ness checks since the tools are constantly monitored in the tool box. The status of the routine checks is displayed to the mechanic using the traffic light metaphor. A green light signals that all the correct tools are contained, a yellow light indicating that tools which do not belong to the box, in addition to all correct tools are contained, and a red light signaling that some tools are missing. Base checks after an MRO tasks is finished have to be initiated by the mechanic using a button on the tool box. The status of base checks is reported to the ERP system for audit purposes.

Figure 6-6 Setup of the Smart Toolbox prototype (left) and the Smart Tool Inventory applications (right)

- The Smart Tool Inventory (see Figure 6-6, right) automates the check-out and return process of specialized tools. Tools are checked-out by a mechanic who identifies himself under the supervision of the person responsible for the tool inventory. During unattended shifts the mechanic can perform a self check-out. The information about the mechanic and the tools being checked-out or returned are submitted to the tool inventory system which offers a web application to retrieve all tool related information. Time consuming searches for tools in the inventory or manual keeping track of tool information is thereby eliminated.
- MRO actions, tool usage, and completeness checks are documented automatically. This ensures accuracy and completeness while reducing cumbersome manual tasks caused by paper-based documentation. As a result, legal requirements are enforced and accurate documentation improves planning of the following MRO tasks.

- Tool usage both for the tools in the tool boxes and in the tool inventory can be estimated based on the number and time of the tools being out of the toolbox resp. checked-out of the tool inventory. This allows replacing or maintaining a tool in time and therefore avoids delays due to broken tools. Tools that require maintenance or replacement are communicated to the mechanic when placing the tool into the toolbox resp. to the person responsible for the tool inventory when a tool is returned.

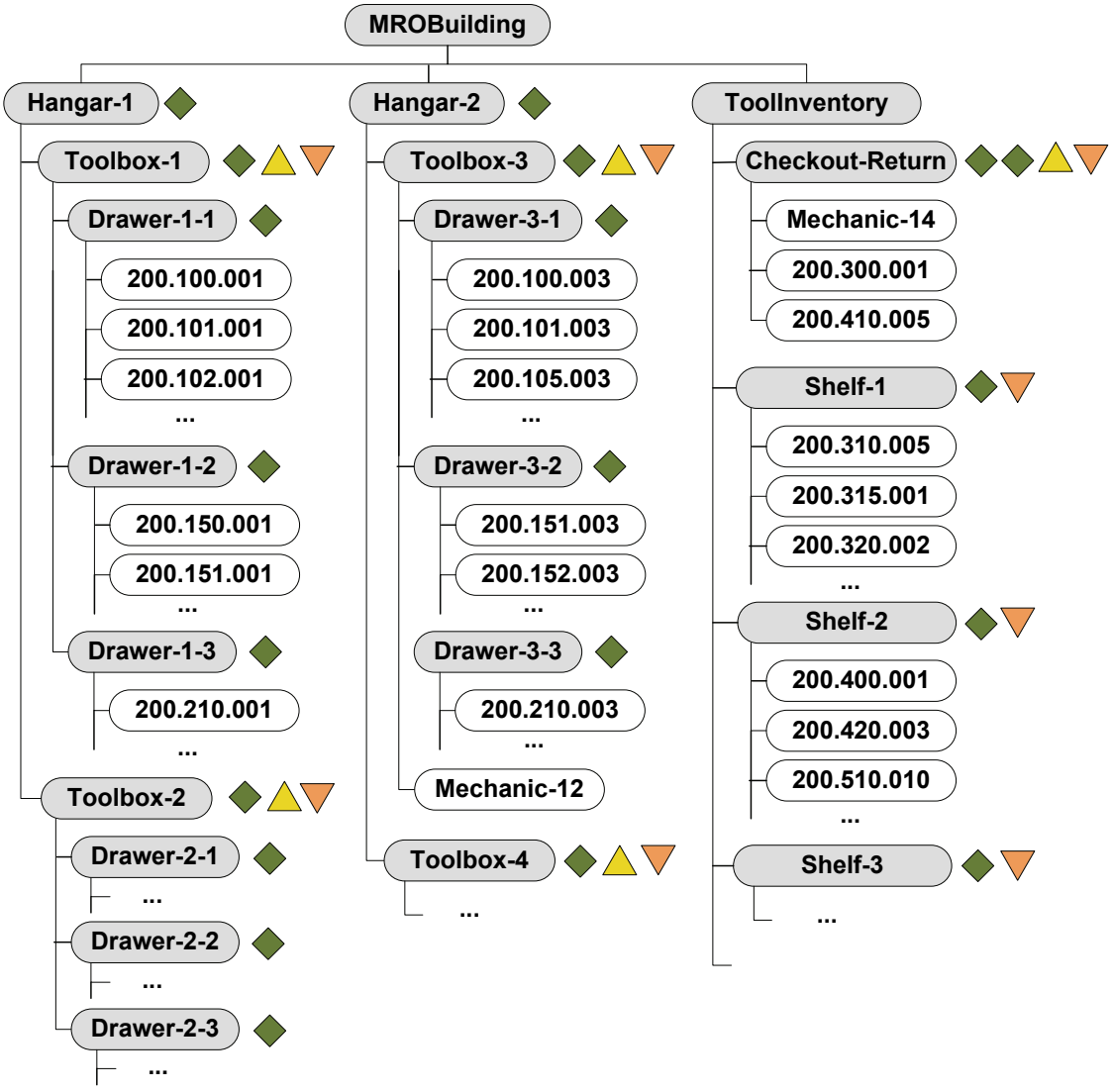


Figure 6-7 Auto-ID Object Model instance for the tool management in aircraft maintenance application

The tool management in aircraft maintenance application consists of four main locations and several mobile objects which are structured in child objects as shown in the instance of the Auto-ID Object Model in Figure 6-7 (readers are marked by green diamonds, sensors by yellow triangles and actuators by orange triangles on the head):

- *Hangar*. The main location where MRO tasks are performed at the aircrafts. The MRO Building has two hangars.
- *Toolbox*. The toolbox of a mechanic contains the tools in several drawers. The traffic light indicating the state of the toolbox is modeled as an actuator and the button to initiate the base checks as a sensor. The toolboxes are mobile objects, that is, they are child objects of the different hangars that can move from one to the other and, in addition, monitor their own content.
- *ToolInventory*. The tool inventory contains the tools in different shelves and compartments. A special child object defines the location where the check-out and return takes place. A check-out or return process is triggered as soon as a tool or mechanic is identified. A traffic light indicates the state of the check-out resp. return process.

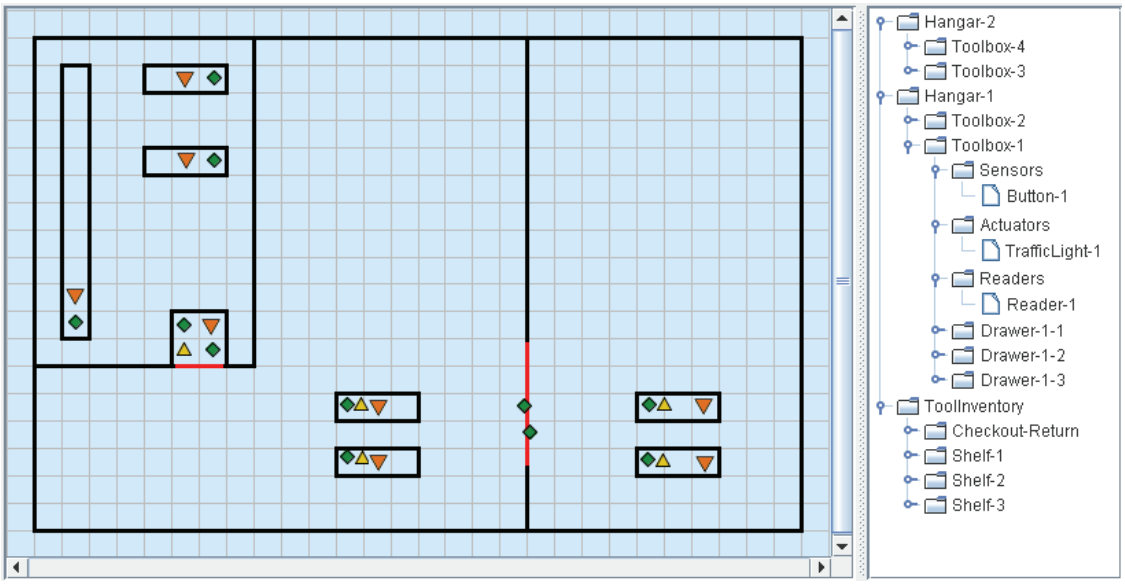


Figure 6-8 Instantiating the tool management in aircraft maintenance application with the VIT

The model instance was created with the VIT (see Figure 6-8). The floor plan shows the two hangars, the tool boxes and the tool inventory.

The different shelves of the toolboxes and the tool inventory are drawn using the special feature of the VIT to define levels. For example, a toolbox that is shown as a rectangle defines the object Toolbox-1. The levels which have been added can be seen in the top right toolbar of the VIT. Levels then define the Drawer child objects of the object Toolbox-1.

Table 6-2 Business processes of the tool management in aircraft maintenance application

Object	Connected Business Processes
Toolbox-n	Base Check, Routine Check
Shelf-n	Required Tool Maintenance
Checkout-Return	Checkout Return

The Auto-ID triggered business processes that are part of the model instance to monitor the tools and report exceptional states to the ERP system and applications are listed in Table 6-2, together with the object in the model instance to which they are connected. The business process definitions of business processes are shown in Figure 6-9 and Figure 6-10.

The business process “Routine Check” (see Figure 6-9) starts with a state where all tools are contained within the drawers of the toolbox (completeness) and no other tools (e.g., of other toolboxes) are contained in the toolbox (correctness). The toolbox is modeled in such a way that its direct child objects are the three drawers and a mechanic who identifies himself by placing his ID badge on the reader on top of the toolbox. The tools are therefore contained one level below the direct child objects. If the completeness and correctness criterion is invalidated, the set that comprises of Level 2 (i.e., the tools in the drawers) is not equal to the set of all the tools of the toolbox. The routine check simply switches between the two states complete and incomplete depending on the condition of the tools in the toolbox.

In the “Base Check” business process, a mechanic has to identify himself and intentionally press the button on the toolbox, which initiates the “Base check in progress” state. The button is modeled as a property “Button” that can have the values 0 (i.e., not pressed) or 1 (i.e., pressed). A blinking yellow light acts as a feedback for the mechanic. The check for completeness resp. correctness is performed equivalently as in the routine check. The result (i.e., complete or incomplete) is sent to the ERP system together with data of the object tree of the toolbox. A green, blinking light

indicates that the base check was successful, that is, completeness and correctness could be validated. If the check was unsuccessful, a red, blinking light is shown. The base check is concluded by switching of the lights and the transition to the first state, waiting for another base check.

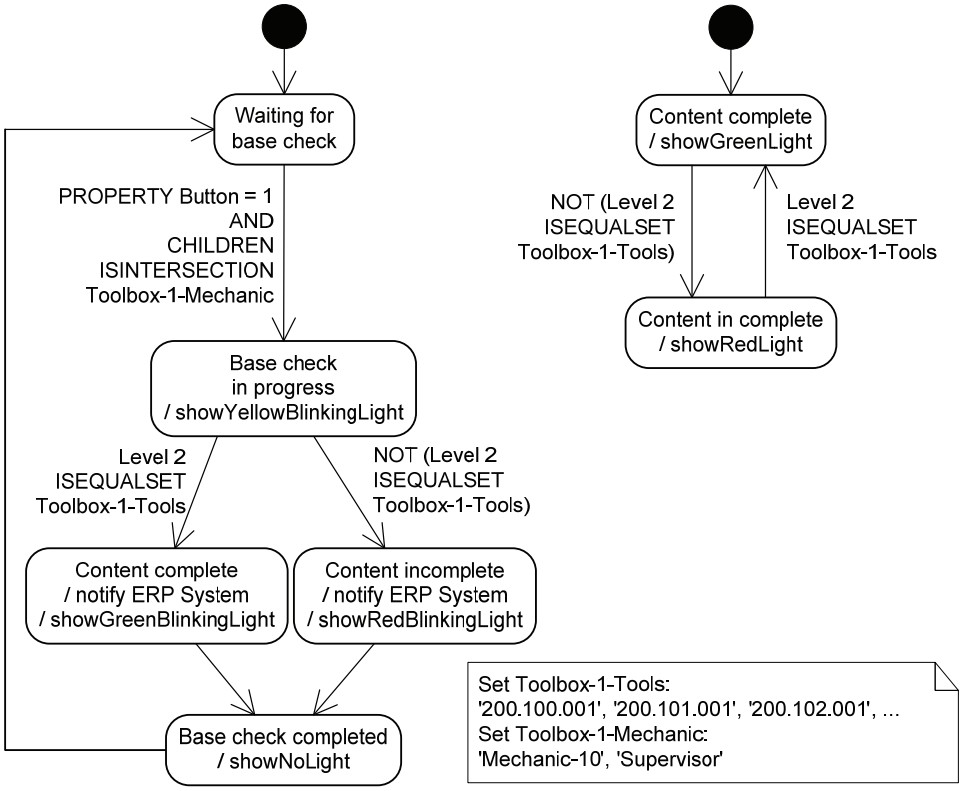


Figure 6-9 Toolbox business processes Base Check (left) and Routine Check (right) of the tool management in aircraft maintenance application

For the tool inventory, the "Checkout Return" business process monitors the check-out and return of tools (Figure 6-10). The process starts in a waiting state until at least one tool is placed onto the check-out and return counter monitored by a reader or a mechanic is identified. The mechanic has a different reader since the personnel ID system is a different RFID system than the tool identification. The yellow light indicates that the system detected tools and the mechanic. If both checked-out and available tools are placed on the counter, the system cannot decide which action the mechanic wishes to perform and the business process transits into the "Illegal action" state indicated by a red light. At least one tool that is checked out is modeled by the state comparison expression `ALL ISIN-`

`TERSECTION CheckedoutTools`. The second part of the condition is the check if any available tools are detected: From the set of detected tools, all checked-out tools are removed. If any tool is left, it must be a tool that is still available. As soon as the illegal state is cleared, the check-out or return process continues. If the mechanic or clerk presses the button and at least one tool and a mechanic have been identified, the action to perform is decided and indicated by a yellow, blinking light. If the detected tools are checked-out, they are returned by the mechanic; otherwise, they are checked-out. The action with the data about the tools and the mechanic is sent to the ERP system and the successful action is indicated by a green light. The business process concludes by switching off the lights and transitioning to the first state.

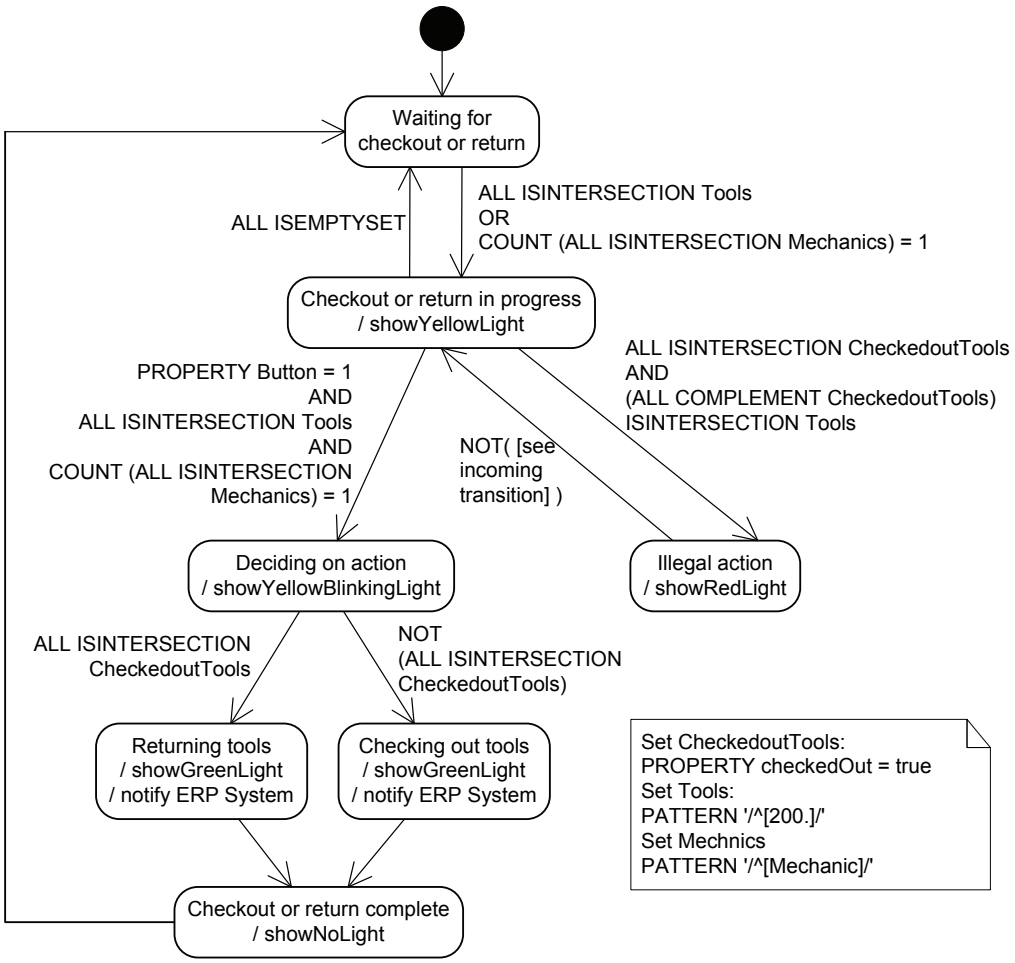


Figure 6-10 Tool inventory business processes of the tool management in aircraft maintenance application

The deployment of the components of the tool management in aircraft maintenance application is performed using the Deployment Definition Tool and the Deployment Tool. Figure 6-11 illustrates the final deployment. A simple Event Layer component is performing filtering and aggregation already on the toolbox. The data is then sent via wireless communication to the OMS component. The OMS is deployed together with the Event Layer that filters and aggregates the incoming observations from the hangar and tool inventory readers. The shelves of the tool inventory are monitored using readers with multiplexer that scan through the different antennas.

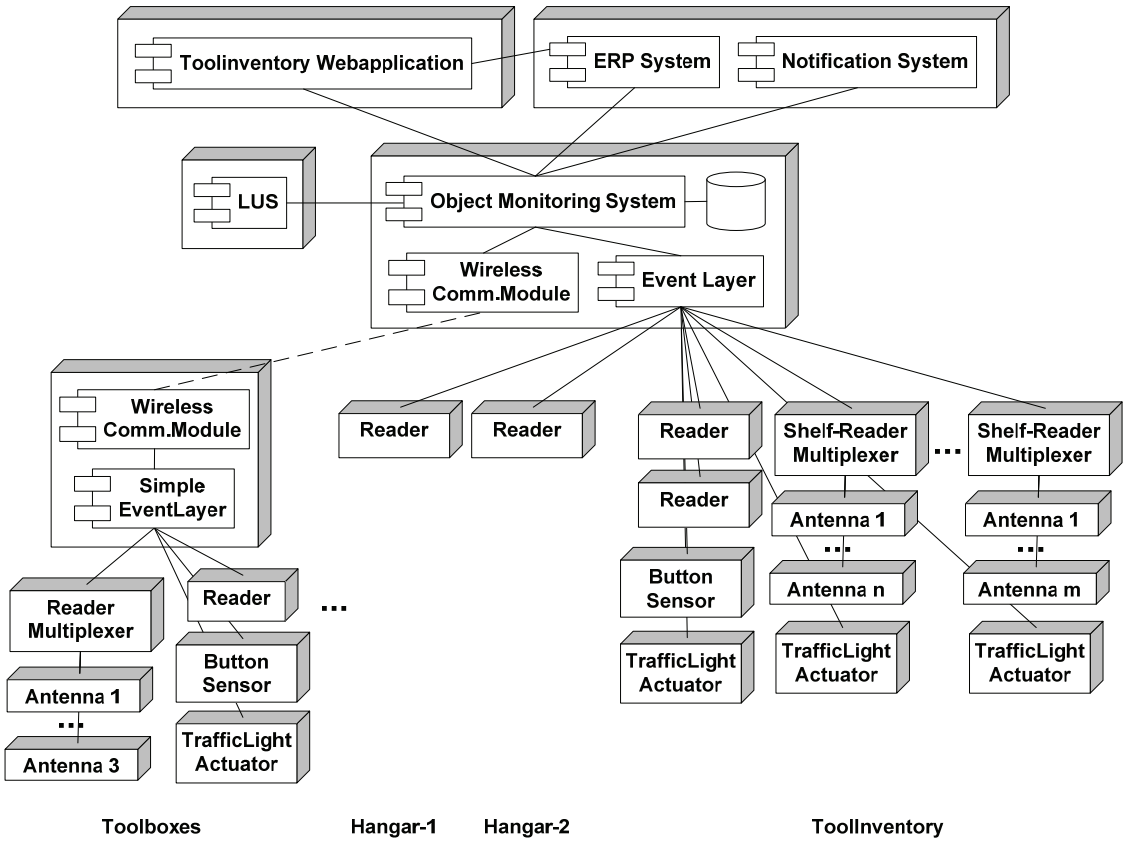


Figure 6-11 tool management in aircraft maintenance application deployment

6.3. Augmented Knight's Castle

Playing with toys is an essential part of the childhood. Besides being a recreational amusement and pure fun, playing also serves as an important function for the psychological, physiological and social development of a

child. To further support creativity and inspire the fantasy of children, traditional toys can be enriched by adding multimedia content and special effects to them. The ideal entertainment and learning experience then comes from the combination of physical experience, virtual content, storytelling and the imagination of the child.

Figure 6-12 Overall setup of the Augmented Knight's Castle

The objective of the Augmented Knight's Castle as a pervasive computing playset (see Figure 6-12) is to foster the children's pretend play and offer ideal possibilities of integrating interactive learning experiences into the children's play. The development of the Augmented Knight's Castle including a user study with children is described in [102, 85, 101, 100].

The main features of the Augmented Knight's Castle are that based on the current game situations or learning scenario, multimedia content or special effects are played. This happens either as a response to an action (e.g., the fanfare is played when the king comes out of his quarters) or randomly (e.g., a dog barks or birds chirp). The multimedia and special effects are:

- Short verbal commentaries of figures in the play (e.g., the knight that gets bored since he is not moved for a while)

- Verbal stories of figures (e.g., the king that explains the life in a castle)
- Sounds (e.g., a fanfare, background sounds of animals, the canon firing, or the dragon roaring)
- Background music that adapts to the play
- Special effects such as lightning the water at the fairy spring or emitting smoke out of the dragon's dungeon

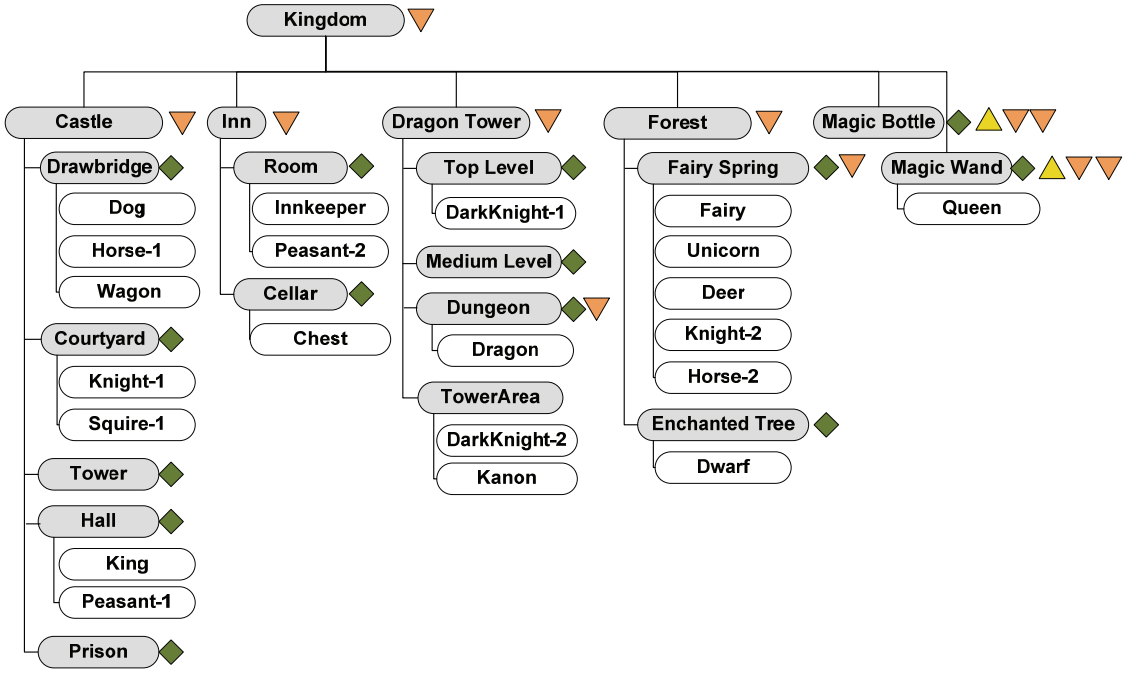


Figure 6-13 Auto-ID Object Model instance of the Augmented Knight's Castle

The Augmented Knight's Castle consists of four main locations and two special objects as shown in the instance of the Auto-ID Object Model in Figure 6-13 (readers are marked by green diamonds, sensors by yellow triangles and actuators by orange triangles on the head). The four main locations have each a loudspeaker (i.e., actuators) where sounds and music can be played:

- *Castle*. The castle has several sub locations where toy figures are monitored. The bridge that can be lowered is an object itself that if detected indicates that it is lowered.
- *Inn*. The inn is a house with a main room and a cellar which is close to the castle.

- *Dragon Tower*. The tower is the head quarter of the dark knights with a dungeon where the red dragon lives. A special actuator in the dungeon can release smoke as a special effect.
- *Forest*. The fairy spring and the enchanted tree are part of the forest. More magical toy figures are associated with the forest. The fairy spring has a light actuator to create a special light effect of the water.
- *Magic Bottle* and *Magic Wand*. These two objects are modeled as locations below the root object. They are special toys that children can use to point to other toy figures and locations (e.g., the Magic Wand can touch the queen to create during play). They both have an acceleration sensor and light and vibration actuators.

Following the Auto-ID Application Development Process, the model instance (see Figure 6-13) was created with the VIT (see Figure 6-14). The metaphor of a building plan represents the different toy buildings.

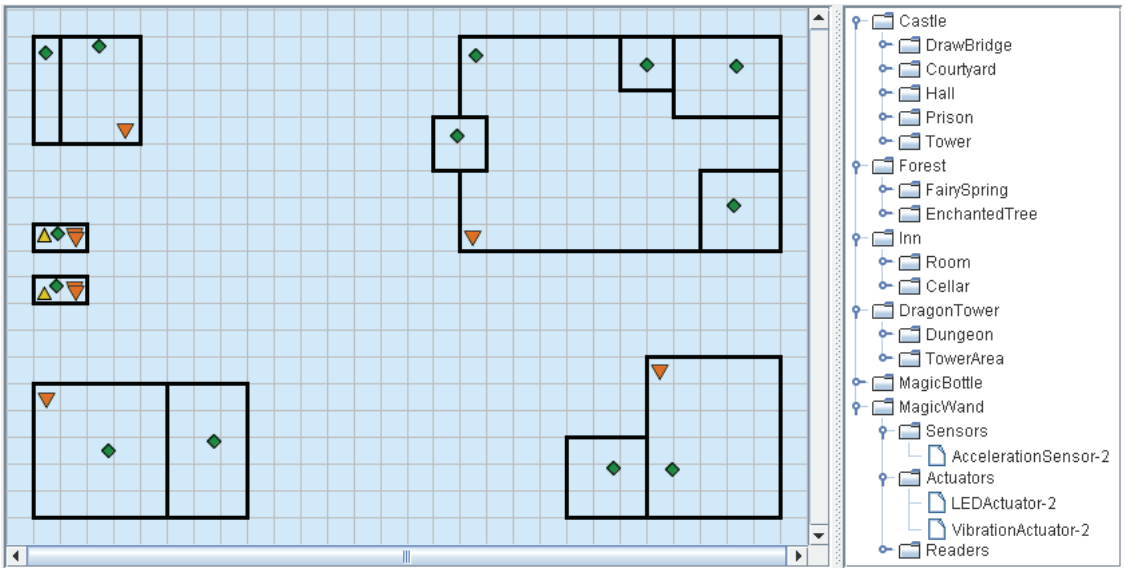


Figure 6-14 Instantiating the Augmented Knight's Castle with the VIT

The Auto-ID triggered business processes that are part of the model instance to monitor the toy figures and enrich the play of the children are listed in Table 6-3, together with the object in the model instance to which they are connected. Since there are many options in the rich play environment, many business processes at different locations exist. They can be classified according to the multimedia content they are playing into business processes that:

- generate general background sounds depending on the toy figures at a location. For example, *CraftsmenWorking* has two states: One where craftsmen figures are detected, the other where they are absent. In the first state a custom action plays random background sounds of craftsmen at work. In a similar manner *CastleSounds*, *KnightsTraining*, *KnightsFighting* or *PeopleTalking* are modeled.
- create figure specific sounds and special effects at a location. For example, *DragonInDungeon* plays roaring sounds and special smoke effects if the dragon is in the dungeon. Other examples are *GhostInCastle*, *CanonFiring* or *FanfareForKing*.
- let certain play figures utter small phrases or sentences. These verbal commentaries act as impulses to the child's play and depend on the figure residing at a location and their activities. For example, *WelcomeKnights* invokes a welcome phrase for the knights, if a king's knight is detected at the drawing bridge. Other examples are *PeopleLooking*, *WelcomeKnights* or *DragonInactive*.
- give certain play figures a voice to tell stories. These stories often depend on interaction of the child with the figure. Examples are *KingsCastleTour*, *InnkeeperStory* or *FairyStory*. As shown below, these business processes are more complex depending on the level of interaction and options the story offers.

Table 6-3 Business processes of the Augmented Knight's Castle

Object	Connected Business Processes
Castle	KingsCastleTour, CanonFiring, GhostInCastle, CastleSounds
DrawBridge	DrawBridgeUpDown, FriendOrFoeEntering, WelcomeKnights
Courtyard	FanfareForKing, CraftsmenWorking, KnightsTraining, KnightsFighting, DragonRoaring
Hall	PeopleTalking, KingGivingAudience
Tower	PeopleLooking
Prison	PrisonerStory, GhostInPrison
Inn	InnkeeperStory, PeasantStory
Room	PeopleCelebrating, InkeeperWaiting
Cellar	WhereIsTheChest, FindTheTreasureStory
DragonTower	DragonTowerSounds

Object	Connected Business Processes
TowerArea	DarkKnightsTalking, KnightsFighting
TopLevel	DarkKnightsLooking
MediumLevel	DarkKnightsLooking
Dungeon	DragonInDungeon, DragonInactive
Forest	ForestAnimalSounds, DragonRoaring
FairySpring	FairyAwaitingKing, FairyStory, FairySpringSounds
EnchantedTree	EnchantedTreeStory, FindTheTreasureStory
MagicBottle	TouchObject, ShakingBottle
MagicWand	TouchObject, MovingWand

The business process definitions of three representative business processes are shown in Figure 6-15. The business process "DragonIn-Dungeon" checks if the object Dragon is in the Dungeon. If the Dragon is there, a roaring dragon sound is played and smoke as special effects is set off. After 5 seconds the dragon sound is played and smoke is set off one more time. If the dragon leaves the Dungeon, the dragon sound is played again. The business process "CraftsmenWorking" plays background sounds of one craftsman working if exactly one object of the set Craftsmen is present at the Courtyard. If more than one craftsman is present a different sound is played. To avoid playing sounds too often and thereby annoying the children, the process waits a minute and transits into the first state.

In the more complex business process "FairyStory", the Fairy if present at the Fairy Spring without the Unicorn, offers the child to tell a story about unicorns if the child brings the unicorn to the Fairy Spring. In addition, to playing the verbal suggestion, the water of the Fairy Spring is light up and a timer is started. If after 1 minute the unicorn is not at the Fairy Spring, the Fairy closes the suggestion and the lights are switched off. If the Unicorn is detected at the Fairy Spring, a light music is played and the Fairy starts telling a story. If during the story telling the Fairy is moved away from the Fairy Spring, the story is interrupted and a suggestion to continue the story by bringing the Fairy back to the Fairy Spring is played. If nothing happens, the suggestion is played after one minute again. Another minute later, the Fairy closes the suggestion. If the Fairy is brought back, the story continues. After 5 minutes, the story finishes and the Fairy asks the child to tell her a story.

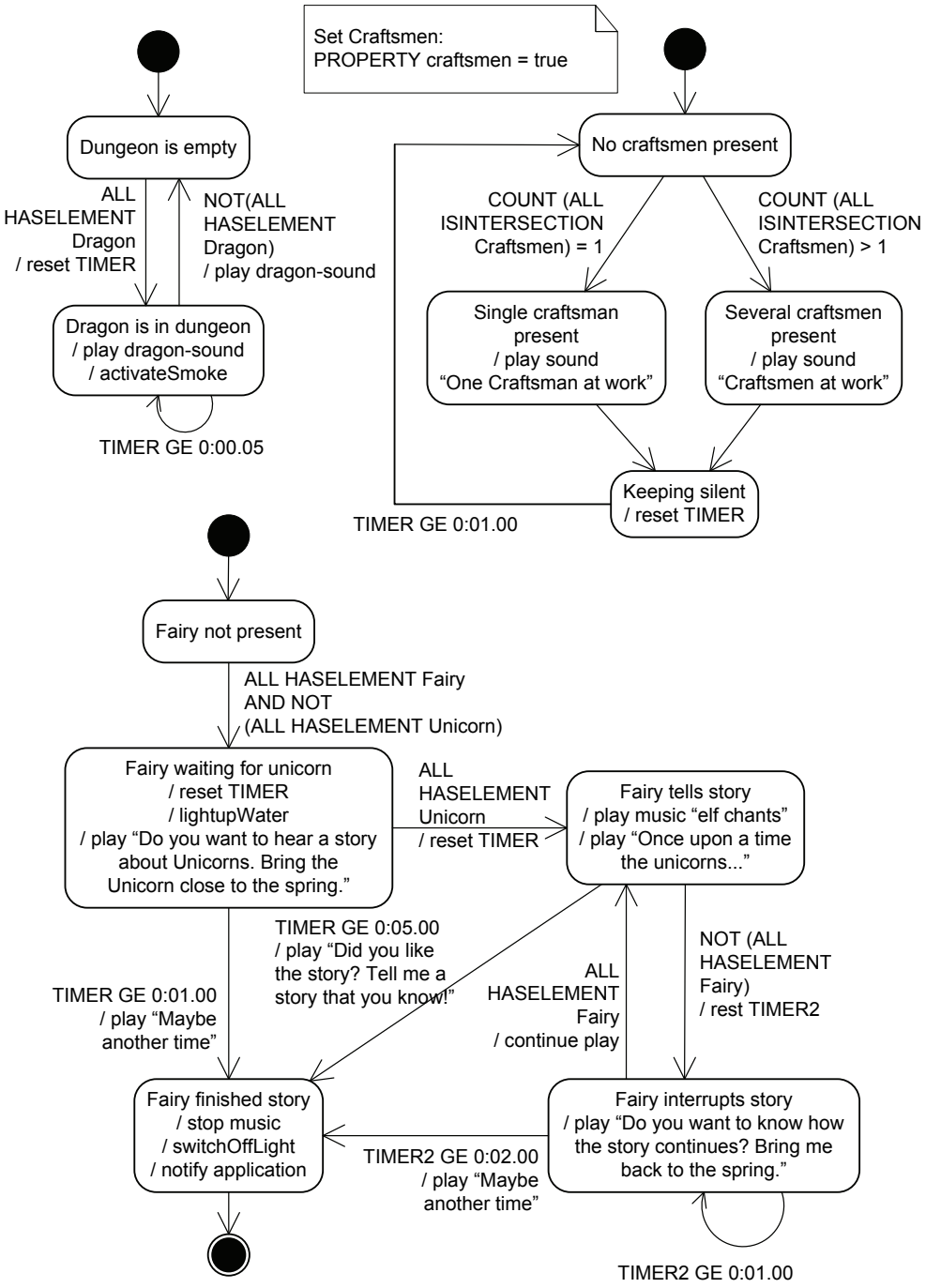


Figure 6-15 Business processes of the Augmented Knight's Castle

Designing a truly pervasive computing game and learning experience, we required that the augmentation does not interfere with, block or compromise the traditional play in any way (i.e., toys are handled in the way

children are used to). RFID readers hidden in the playset detect the position of objects (in our case 13.56 MHz RFID, see Figure 6-16, left). RFID tags of different sizes are attached to or incorporated into the pieces of the playset to uniquely identify them (see Figure 6-16, right).

The two point-and-touch toys, the Magic Bottle and the Magic Wand, rely on the custom-built mobile RFID reader module (Skyetek M1-mini RFID reader controlled by a BTnode that is used as a Bluetooth device server). In addition, the BTnote has several sensors and actuators attached to it.

Figure 6-16 Antennas embedded in the playset (left) and RFID transponders to tag toy pieces (right)

The deployment of the components of the Knight's Castle is performed using the Deployment Definition Tool and the Deployment Tool. Figure 6-17 illustrates the final deployment. The Event Layer, the Object Monitoring System, and the application are all deployed on one machine, called the Base Station, which is typically hidden in the playset. The different main locations are monitored by readers with multiplexer that scans through the different antennas which monitor the sub locations. Using only one reader with a multiplexer and antennas for all locations is not acceptable since it would result in up to 2 seconds worst-case until an object is detected on an antenna. The loudspeakers are part of a Dolby Surround system which is activated using a special sound library by the application. The two BTnode reader and sensor components communicate with a Bluetooth module on the Base Station which routes the communication to the Event Layer.

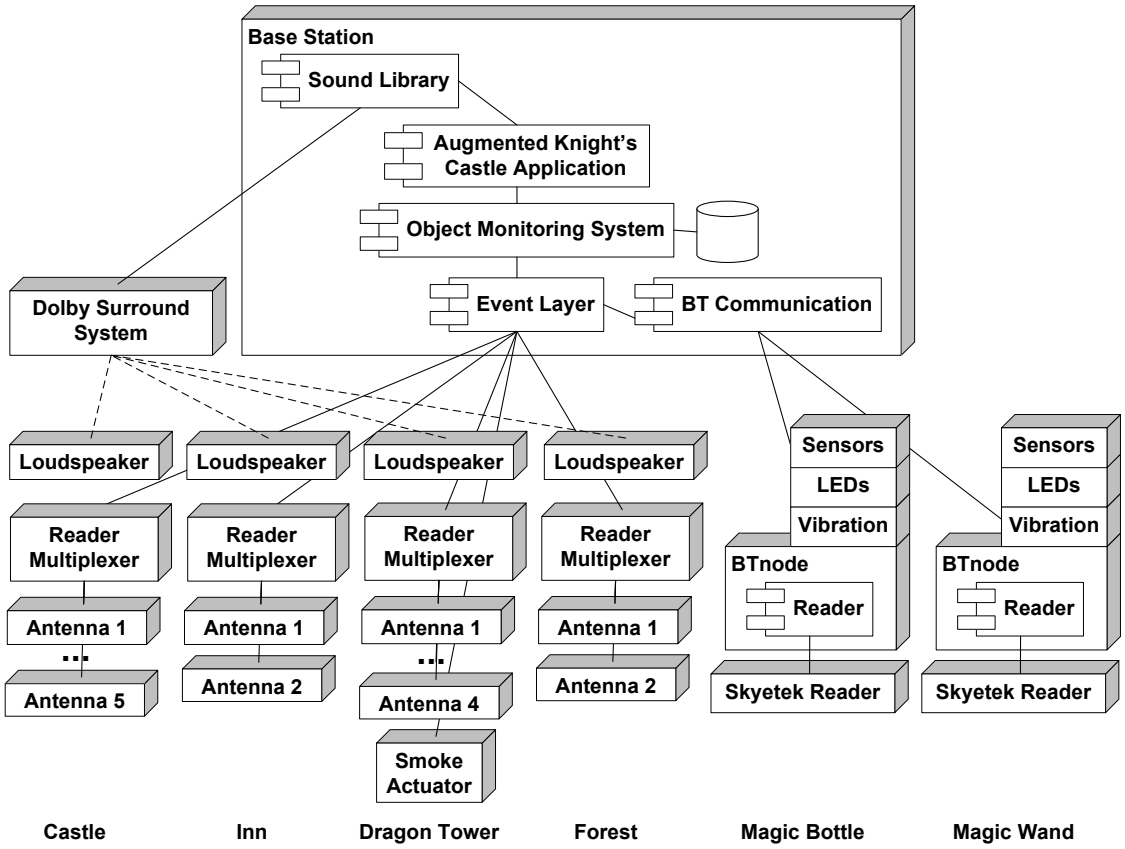


Figure 6-17 Augmented Knight's Castle deployment

6.4. Discussion

The Auto-ID Object Model provides for an adequate representation of the application domains and provides the right level of abstractions. The model instances could be clearly defined and the locations modeled as an object hierarchy. Merging the concepts of location and object greatly simplifies the model instances. Properties of objects hide the specific sources of object related information. The most common property sources are provided and specific data sources can be easily implemented. The state machine based programming model is powerful enough to define a great variety of Auto-ID related business processes in the application domains, yet simple enough that application developers can focus on the application logic they want to achieve. From a broader view, the micro business processes can be seen as the parts of higher workflow processes in an application domain that have been pushed closer to the physical ob-

jects on which they depend. Reports of business processes can then trigger the higher workflow.

The Auto-ID Application Development Process and tools greatly facilitates the instantiation, configuration and deployment of an Auto-ID infrastructure. No specific knowledge of the software components is needed and the application engineer can concentrate on the modeling of the object tree and the definition of the business processes. Since the process separates instantiation and deployment, a different user, typically an Auto-ID system administrator with specific knowledge about the server hardware manages the deployment of the components.

The high-level Auto-ID system requirements that are based on an analysis of different Auto-ID applications and that an Auto-ID infrastructure should meet are presented in section 3. Table 6-4 lists these system requirements and explains which concept of our presented approach implements the required feature. Our contributions (i.e., the Auto-ID Object Model, the business process support, the OMS and the Auto-ID Application Development Process) focus on the requirements R1-R10, R19 and R20. Requirements R11-18 are implemented by standard Auto-ID readers such as LLRP Readers or the filtering and aggregation middlewares such as the Event Layer or the ALE Middleware. The filtering and aggregation concepts can be seen as a minor contribution of our approach onto which the other concepts build.

Table 6-4 Auto-ID system requirements implemented by the presented approach

System Requirement	Implemented by Auto-ID Object Mode or Auto-ID Infrastructure feature	Sections
(R1) Object Representation	The Auto-ID Object Model provides a representation of physical objects as its core concept. The OMS maps tag observations to object movements in the object tree.	4.3.1 4.4.1
(R2) Object Persistency	The object tree is constantly stored to a RDBMS where it can be queried by the applications.	4.3.4 04.4.1
(R3) Object Relationships	The Auto-ID Object Model directly supports the containment relationship between objects. The child objects of an object define the neighbor relationship among themselves.	4.3.1 4.4.1

System Requirement	Implemented by Auto-ID Object Mode or Auto-ID Infrastructure feature	Sections
(R4) Location Information	All objects implicitly define a symbolic location identified by their ObjectID. If geospatial coordinates of objects are needed they can be represented as an object property.	4.2.2 4.3.1 4.4.1
(R5) Object Identification	Objects are identified by the OMS using either the TagID of the observation or by the ObjectID stored on the tag. Via the Event Layer reps. ALE Middleware the OMS abstracts from different Auto-ID or sensor systems.	4.3.1 4.4.1
(R6) Object Identifier	The Auto-ID Object Model requires a unique ObjectID for each object. The OMS can either map TagIDs to ObjectIDs or preferably use ObjectIDs such as the EPC.	4.3.1 4.4.1 2.2
(R7) Object Data Enrichment	The Auto-ID Object Model provides additional object data by offering Properties of objects. As an abstraction of different data sources, PropertySources are provided.	4.3.2 4.4.1
(R8) Physical World Interaction	The system reacts to the physical world by detecting physical objects via their tags and by reading from sensors. Interaction is provided by actuators which are abstracted in the Auto-ID Object Model by the concept of Functions.	4.3.2 4.3.3 4.4.1
(R9) Business Context Enrichment	Business processes based on the Auto-ID Object Model offer a rich mechanisms to formulate conditions and process flow to report exceptional states to applications.	4.3.5 4.4.1
(R10) Data Dissemination	Several reporting actions in the business process support of the Auto-ID Object Model/OMS allow notifying applications and information systems of exceptional states. The Event Layer reps. ALE Middleware also allow applications to directly register with them to receive filtered and aggregated Auto-	4.3.5 4.4.1

System Requirement	Implemented by Auto-ID Object Mode or Auto-ID Infrastructure feature	Sections
	ID observations.	
(R11) Auto-ID Data Aggregation	The presented aggregation mechanisms provided by the Event Layer reps. ALE Middleware allow aggregating observations into e.g. enter/exit events.	4.1 4.4.1
(R12) Auto-ID Data Filtering	The presented filtering mechanisms provided by the Event Layer reps. ALE Middleware allow filtering on ObjectIDs and ReaderIDs.	4.1 4.4.1
(R13) Fault and Configuration Management	LLRP Readers provide a Simple Network Management Protocol (SNMP) interface that allows integrating the reader into an existing hardware monitoring infrastructure.	2.2
(R14) Tag Identifier Management	Most RFID systems provide a unique TagID (e.g., Philips I-Code) or provide user memory to store a unique ID.	2.2
(R15) Tag User Memory	Most RFID systems provide user memory on tags, e.g. the EPC Gen2 Tags.	2.2
(R16) Sensor Support	The Auto-ID Object Model abstracts from sensors by using PropertySources. Arbitrary sensors can be added to the system by implementing plug-ins for the OMS.	4.3.2 4.3.3 4.4.1
(R17) Actuator Support	The Auto-ID Object Model abstracts from actuators that are supported by the OMS by using Functions. New actuators can be added by implementing plug-ins for the OMS.	4.3.3 4.4.1
(R18) External Reader Triggers	LLRP Readers can be triggered by external sources.	2.2
(R19) Loose Coupling of Components	Components are dynamically connected using a lookup service where components are specified by IDs.	4.4.1
(R20) Configurability of System	OMS, EventLayer resp. ALE Middleware are configurable by XML files. The Auto-ID Application Development Process provides visual configuration for end users.	4.4.1 5.2

7. Conclusion

In this thesis, we argue that despite existing Auto-ID systems, Auto-ID application developers are still facing several challenges: Instead of concentrating on the application or business logic they have to deal with Auto-ID specific details and have to re-create features when developing new Auto-ID applications, for example to represent and enrich data or to react to business events.

To bridge this gap between the software applications or information systems such as ERP systems on one side and the Auto-ID systems on the other, this thesis provides concepts, programming models, building blocks and tools that abstract from Auto-ID specific details and provide the necessary services and the appropriate level of reuse to facilitate the development of Auto-ID applications. The major contributions are:

- The *Auto-ID Object Model* abstracts from low-level Auto-ID and sensor concepts and provides the means for an application to model and represent its domain as the base for the application logic. Moreover, a state machine-based programming model allows defining Auto-ID related micro business processes to report exceptional states of the monitored objects to the applications.
- The *Auto-ID Application Development Process* is a visual tool-based approach to instantiate, configure and manage an Auto-ID infrastructure. The approach provides a concrete representation of the application domain and supports non-software developers in the different tasks over the lifecycle of an Auto-ID infrastructure.

The Object Monitoring System (OMS) is a prototypical implementation of an Auto-ID infrastructure that includes the proposed concepts. Using the OMS, both the Auto-ID Object Model and the Auto-ID Application Development Process were evaluated, based on several different application scenarios to demonstrate that our approach is applicable for a variety of different applications domains. The applications and application domains we analyzed are: A retail store supply chain application that provides support for the logistical management; the Smart Medicine Shelf, an

automated shelf in hospitals that keeps track of different kinds of medications that require different conditions; the tool management in aircraft maintenance application which keeps track of tools and parts in an aircraft maintenance environment; and the Augmented Knight's Castle, a pervasive computing playset which enriches a child's pretend play by using background music, sound effects, and verbal commentary of toys that react to the child's play.

In this final section, we revisit the proposed concepts and benefits of the contributions. We also discuss limitations of the approaches and outline potential future work.

7.1. Auto-ID Object Model and Business Process Support

7.1.1. Contribution

The Auto-ID Object Model provides an abstraction from the low-level Auto-ID and sensor concepts (e.g., tag observations or tag memory) and provides the means for an application to model its domain as the base for the application logic. However, the Auto-ID Object Model does not intend to provide a world model that allows describing any potential application domain. The model focuses on the domain of Auto-ID applications, that is, applications whose application logic is based on an implicit or explicit model of the physical world and is triggered by (near) real time observations of the physical world through Auto-ID readers and sensors.

The Auto-ID Object Model is based on a symbolic location model, in which physical objects also define locations. The term *object* stands for the entity in the model that represents physical objects in the real world. Objects can contain other objects and thereby form a hierarchical object tree. Auto-ID readers that are linked to objects indentify physical objects by their Auto-ID tags. Identified objects then become the child objects of the object to which the reader is linked. *Properties* represent additional dynamic or static business information about objects. Such information is received and related to an object in the model using property sources (e.g., product information databases, sensors or data on Auto-ID tags read by Auto-ID readers). *Functions* are the abstraction for the interaction of the system with the physical world. The interaction is then performed by actuators such as locking a door or setting a signal light to red.

For a specific Auto-ID application, the model is instantiated. Such an instance contains static information, i.e. objects that represent locations of the application domain and dynamic information that changes during the life-time of the application such as objects and object state information observed by Auto-ID readers and sensors.

The dynamic changes in the Auto-ID Object Model correspond to the changes in the physical world. Many applications are only interested in exceptional states related to objects and their properties such as the completion of an incoming shipment, reached expiry dates of products in a shelf, or the temperature inside a cool box that exceeds a threshold.

In most cases a business process that also needs business knowledge about the application domain can be defined to determine such exceptional states. In traditional approaches, applications would need to implement such business processes that would involve querying the Auto-ID Object Model instance quite frequently leading to high traffic for the OMS. Another disadvantage would be that first, system engineers have to define the business processes, and then software engineers have to implement them, thereby often re-implementing similar concepts over and over again.

Our approach offers a business process support combined with the Auto-ID Object Model. A business process can be defined using a state machine-based model with transition conditions and actions based on the state of objects in the Auto-ID Object Model instance. The application is only notified in cases of exceptional states.

The rationale for a state based approach is that the analysis of Auto-ID applications has shown that an event mechanism based simply on conditions of object and property configurations is not sufficient to describe many exceptional states. The decision if an exceptional state occurred often depends on a related state that has happened a certain time before the state in question. This dependency of states over time can very well be expressed and modeled using state machines. In addition, state machines express a kind of business process flow that is well known in industry and state machines can be modeled using available design and modeling tools such as UML.

In the Auto-ID Object Model, a business process is always linked to an Object representing a location or physical object in which the business process is interested. A business process comprises of several *states* which have *transitions* between them. State transitions are coupled to certain configurations of objects and their properties. A transition happens if

the *condition* describing the transition is true. Conditions can involve presence or absence of a set or number of objects or object properties and time constraints. It is also possible to logically combine different conditions to describe a transition. The conditions are defined using an Auto-ID Object Model specific condition language that allows access to the model and provides several operators to formulate the logical conditions. *Actions* can be defined to take place when a transition is performed, when a state is entered and when a state is exited. Actions can be model-actions such as executing a function or setting a property, reporting-actions, that is sending reports to applications and information systems, or custom-actions provided by the application.

Our evaluation, based on several representative case-studies of different application domains, shows that the Auto-ID Object Model provides for an adequate representation of the application domains and provides the right level of abstractions. The model instances could be clearly defined and the locations modeled as an object hierarchy. Merging the concepts of location and object greatly simplifies the model instances. Properties of objects hide the specific sources of object related information. The most common property sources are provided and specific data sources can be easily implemented. The state machine based programming model is powerful enough to define a great variety of Auto-ID related business processes in the application domains, yet simple enough that application developers can focus on the application logic they want to achieve. From a broader view, the micro business processes can be seen as the parts of higher workflow processes in an application domain that have been pushed closer to the physical objects on which they depend. Reports of business processes can then trigger the workflow.

7.1.2. Limitations and Future Work

The hierarchical symbolic location model as base for the Auto-ID Object Model has been chosen since it fits naturally with Auto-ID readers monitoring a not well defined space around them. The model therefore provides no direct support for geospatial information such as coordinates of objects and calculations such as distance between objects. Geospatial information could be stored as a property of objects extending the model to a hybrid location model. However, special calculations have to be provided by the application. The closeness of objects can only be derived by the neighboring relationship, that is, all the child objects of one parent.

In the location tree, containment relationships are optimal represented, for example, a shelf with its compartments is modeled as a shelf object with compartments as child objects. In this case, the real physical containment is mapped to our model. However, since an object can only have one parent object, certain cases cannot be naturally modeled. For example, a building with two floors and two wings where rooms should be contained in floors and wings. A lattice or graph structure with multiple parents would allow modeling this relationship. In our model, objects that define a combination of floors and wings would have to be introduced.

Since objects only exist in a model instance if they are defined statically or if they are monitored by a reader, an object that leaves the read range of a reader also leaves the model instance, that is, no information is available of that particular object any more. Mobile objects (i.e., objects that move and also monitor their surrounding such as a fork lifter or shopping cart) have therefore to be defined in the instantiation phase of the model. The OMS will then keep these objects in the model under the special location “unknown” in case they are not monitored any more. If they are detected again, the object with its sub tree will become available.

Many business processes can successfully be modeled with the state machine based programming model. However, since one design objective for the model is simplicity, the model has its limits if transition conditions and dependencies among states get too complex. In this case, the state machine definitions would get rather complicated and its value for application developers would decrease. If the application logic gets too complex, only elementary parts of the process would be modeled as a business process of the Auto-ID Object Model. The overall process flow is then handled by the application which is notified by the elementary business processes.

Currently the OMS is only a prototypical implementation of the Auto-ID Object Model, the business process support and object persistency and query capabilities. For a software component that can be used in productive environments, performance optimizations have to be implemented such as a more performant database approach as discussed in this thesis. In addition, the management mechanism of the state machines can be improved and the reporting and messaging can be extended. For application domains with large number of locations and objects such as large supply chains, the OMS could be implemented as a distributed system. Since the Auto-ID Object Model is a hierarchical model it can be naturally distri-

buted to allow processing of state machines in parallel. The distribution of the model does not need to be transparent to the application. Connecting to a certain OMS node, the application would perceive the sub tree as its model. Connecting to the root OMS node, the application would be provided with the complete model.

In the future, the Auto-ID Object model and the business process support have to be applied to more application domains to further refine and extend the model. By instantiating the model for different application scenarios, the right balance between a simple and a powerful model will increase and the model will stabilize.

7.2. Auto-ID Application Development Process

7.2.1. Contribution

Typically an Auto-ID infrastructure consists of many different hardware and software components that have to be set up, configured and customized to fit the need of a certain application domain. Typically the configurations of these components such as OMS, filtering and aggregation middlewares, readers and sensors have to be defined in component specific formats (e.g. XML or text based configurations) and often certain parts of the components have to be implemented and integrated into these components. Such an instantiation and configuration is a tedious task and requires programming skills, specific knowledge of all software and hardware components of an Auto-ID application deployment, in addition to the application domain knowledge.

In this thesis, we present the Auto-ID Application Development Process to bridge this gap between Auto-ID application developers on one side and the components of an Auto-ID infrastructure on the other. The process is based on generative and visual programming concepts and provides a concrete representation of the application domain and supports non-software developers in the different tasks over the lifecycle of an Auto-ID infrastructure.

In the *instantiation* phase, a visual tool, the Visual Instantiation Toolkit, allows an application engineer to intuitively construct a hierarchical object model using the metaphor of building and floor plans as a concrete representation. Objects such as buildings, rooms, shelves, or boxes can be drawn on a plan. Readers and sensors can be placed and configured, and

are automatically linked to objects. In addition, business processes linked to objects can be defined. A *generation* tool automatically creates instantiated software components that contain the specific configuration, and a description of the hardware configuration in the correct format. These components are then deployed using a visual *deployment* tool for a system administrator to define the hardware machines for the components and their communication relationships.

Our evaluation, based on several representative case-studies of different application domains, shows that the process and tools greatly facilitates the instantiation, configuration and deployment of an Auto-ID infrastructure. No specific knowledge of the software components is needed and the application engineer can concentrate on the modeling of the object tree and the definition of the business processes. Since the process separates instantiation and deployment, a different user, typically an Auto-ID system administrator with specific knowledge about the server hardware manages the deployment of the components.

7.2.2. Limitations and Future Work

The visual metaphor of building and floor plans to represent the Auto-ID Object Model needs to be reinterpreted for certain application domains. For example, for the Smart Medicine Shelf, a box drawn on the plan represents not a vertical view onto a building but a horizontal view on the compartments of the shelf. The concreteness of the visual metaphor could therefore be improved further. Different metaphors could be offered from which the application engineer could choose. For certain application domains such as supply chain management, a specific metaphor could be developed to increase the concreteness even further.

To have a better overview over more complex state machines used in the business process definitions, an intelligent zoom mechanism that reduces the information according to the zoom level would be beneficial. Standard zooming simply reduces all information proportionally.

The definition of state transition using the Auto-ID Object Model set and state transition definition language is currently provided as a textual-based construction kit. For more complex state transition definitions this feature is not intuitive any more. A more visual approach should be taken based on metaphors for set theory operations. For example, the set intersection operation could be visualized through two overlapping ellipses.

Currently the Visual Instantiation Tool and the deployment tool are only prototypical implementations. For the usage in productive environments, the stability and performance have to be further improved. To verify the metaphors used for the visual representations, user studies with application engineers of different application domains should be conducted. Results from these user studies could improve the usability of the tools and the overall development process. The process and tools could also be extended to provide for a more seamless integration of ERP systems and other information systems into the Auto-ID application development.

8. Appendices

8.1. Formal Definitions of the Auto-ID Object Model Set Definition Language in EBNF

```
Sets = SetDecl {Sets}.
SetDecl = “SET” SetName “:” SetType.
SetType = ObjectSet | PatternSet | PropertySet.
ObjectSet = “OBJECTS” ObjectsDecl.
ObjectsDecl = ObjectDecl “,” ObjectsDecl      |
ObjectDecl.
ObjectDecl = “'” ObjectId “'”.
PatternSet = “PATTERN '” PatternString “'”
     | “PATTERN
PatternString = “/” (Alpha | Digit | Char) { Alpha
| Digit | Char } “/”.
PropertySet = BoolExpr.
BoolExpr = BoolTerm | (BoolExpr “OR” BoolTerm).
BoolTerm = BoolFactor | (BoolFactor “AND”
BoolTerm).
BoolFactor = PropertyExpr | ( “(”BoolExpr “)” )
     | (“NOT” BoolFactor).
PropertyExpr = “PROPERTY” PropertyName
PropertyCond.
PropertyCond = “EXISTS” | NumOp FloatNumber |
DateOp Date | “=” String.
NumOp = “=” | “>” | “<” | “>=” | “<=”.
DateOp = “EQ” | “GT” | “LT” | “GE” | “LE”.
SetName = Ident.
ObjectId = Ident.
Ident = Alpha {Alpha | Digit | IdentChar}.
String = {Alpha | Digit | IdentChar}.
Alpha = “a” | “b” | “c” | …| “z” | “A” | “B”.| “C”
|…| “Z”.
Digit = “0” | “1” | “2” | “3” | “4” | “5” | “6” |
“7” | “8” | “9”.
```

```
Char = "_" | "-" | "+" | "& " | "*" | "?" | "(" |
")" | "{" | "}" | "[" | "]" | "^" | "." | "!" |
":" | "=" | "|" | "$" | "\" | "#" | "'" | "`".
IdentChar = "_" | "-" | "+" | "& " | "*" | "?" |
"{" | "}" | "[" | "]" | "^" | "." | "!" | "=" |
"|" | "$" | "\" | "#" | "`".
Date = Digit Digit "." Digit Digit "." Digit Digit
Digit Digit
```

8.2. Formal Definitions of the Auto-ID Object Model Business Process Condition Definition Language in EBNF

```
StateTransitionCondition = ConditionExpr.
ConditionExpr = ConditionTerm
     | (ConditionExpr  "OR" ConditionTerm).
ConditionTerm = ConditionFactor
     | (ConditionFactor "AND" ConditionTerm).
ConditionFactor = SimpleConditionExpr
     | (" (" ConditionExpr ")" )
     | ("NOT" ConditionFactor).
SimpleConditionExpr = SetExpr | CountExpr
     | PropertyExpr.
SetExpr = EqualSetDecl | IntersectSetDecl
     | SubSetDecl | SuperSetDecl.
EqualSetDecl = Level "ISEQUALSET" SetName.
IntersectSetDecl = Level "ISINTERSECTION" SetName.
SubSetDecl = Level "ISSUBSET" SetName.
SuperSetDecl = Level "ISSUPERSET" SetName.
CountExpr = "COUNT (" Level CountSetOp SetName ")"
     NumOp Number
     | "COUNT (" Level ")" NumOp Number.
CountSetOp = "INTERSECTION" | "SUBSET".
PropertyExpr = "PROPERTY" PropertyName NumOp
     FloatNumber
     | "PROPERTY" PropertyName "=" String.
Level = "ALL" | "CHILDREN" | "PARENT"
     | ("LEVEL" Number).
NumOp = "=" | ">" | "<" | ">=" | "<=".
SetName = Ident.
PropertyName = Ident.
Ident = Alpha {Alpha | Digit | IdentChar}.
```

```
Alpha = “a” | “b” | “c” | …| “z” | “A” | “B”.| “C”
     |…| “Z”.
Digit = = “0” | “1” | “2” | “3” | “4” | “5” | “6”
     | “7” | “8” | “9”.
IdentChar = “_” | “-“ | “+” | “& “ | “*” | “?” |
     “{“ | “}” | “[“ | “]” | “^” | “.” | “!” | “=”
     | “|” | “$” | “\” | “#” | “`”.
```

8.3. Example Business Process Definition in XML

```
<business-process-definition>
  <states>
    <state name="START"/>
    <state name="Awaiting Shipment"/>
    <state name="Shipment Arriving">
      <action type="start-time"
              timerName="shipmentTimer"/>
    </state>
    <state name="Shipment Potentially Incomplete">
      <action type="enter"
              kind="notify-staff"
              className="oms.action.NotifyStaff"
              args="mobile,0441234567"/>
    </state>
    <state name="Shipment Incomplete">
      <action type="enter"
              kind="report"
              client="wms-app"
              objects="ALL INTERSECTION
                           ObjectsShipment1020"/>
    </state>
    <state name="Shipment Complete">
      <action type="enter"
              kind="report"
              client="wms-app"
              objects="ObjectsShipment1020"/>
    </state>
    <state name="END"/>
    </state>
  </states>
  <transitions>
    <transition start="START"
                end="Awaiting Shipment">
```

```
        </transition>
        <transition start="Awaiting Shipment"
                    end="Shipment Arriving">
        </transition>
        <transition start=""
                    end="">
        </transition>
        <transition start=""
                    end="">
        <transition start=""
                    end="">
        <transition start=""
                    end="">
        <transition start=""
                    end="">
        <transition start=""
                    end="">
      </transitions>
    </business-process-definition>
```

8.4. OMS Database Implementation Details

```
CREATE TABLE `property_history` (
`ObjectID` varchar(400) NOT NULL,
`PropertyName` varchar(400) NOT NULL,
`PropertyValue` varchar(400) NOT NULL,
`Timestamp` bigint(64) NOT NULL
);
CREATE TABLE `object_history` (
`ObjectID` varchar(400) NOT NULL,
`ParentID` varchar(400) NOT NULL,
`EnterTime` bigint(64) NOT NULL
`ExitTime` bigint(64) NOT NULL
);
```

Figure 8-1 SQL statements to create the object and property history

8.5. XML Schemes of the VIT / Generator

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!--W3C Schema erstellt mit XMLSpy v2005 rel. 3 U
(http://www.altova.com)-->
```

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
     <xs:element name="actionstring" type="xs:string"/>
     <xs:element name="actiontype" type="xs:string"/>
     <xs:element name="boundedArea">
          <xs:complexType>
               <xs:sequence>
                    <xs:element ref="segment" minOc-
curs="0" maxOccurs="unbounded"/>
               </xs:sequence>
               <xs:attribute name="id"
type="xs:string"/>
               <xs:attribute name="fid"
type="xs:string"/>
          </xs:complexType>
     </xs:element>
     <xs:element name="boundedAreas">
          <xs:complexType>
               <xs:sequence>
                    <xs:element ref="boundedArea" maxOc-
curs="unbounded"/>
               </xs:sequence>
          </xs:complexType>
     </xs:element>
     <xs:element name="canvas">
          <xs:complexType>
               <xs:attribute name="height"
type="xs:string" use="required"/>
               <xs:attribute name="width"
type="xs:string" use="required"/>
          </xs:complexType>
     </xs:element>
     <xs:element name="children">
          <xs:complexType>
               <xs:sequence>
                    <xs:element ref="object" minOc-
curs="0" maxOccurs="unbounded"/>
               </xs:sequence>
          </xs:complexType>
     </xs:element>
     <xs:element name="coords">
          <xs:complexType>
               <xs:attribute name="x" type="xs:string"
use="required"/>
```

```
              <xs:attribute name="y" type="xs:string"
use="required"/>
         </xs:complexType>
     </xs:element>
     <xs:element name="definition">
          <xs:complexType mixed="true">
               <xs:choice minOccurs="0" maxOc-
curs="unbounded">
                    <xs:element ref="name"/>
                    <xs:element ref="offset"/>
                    <xs:element ref="length"/>
               </xs:choice>
               <xs:attribute name="id"
type="xs:string"/>
          </xs:complexType>
     </xs:element>
     <xs:element name="definitions">
          <xs:complexType>
               <xs:sequence>
                    <xs:element ref="definition" maxOc-
curs="unbounded"/>
               </xs:sequence>
          </xs:complexType>
     </xs:element>
     <xs:element name="fieldmap">
          <xs:complexType>
               <xs:sequence>
                    <xs:element ref="name"/>
                    <xs:element ref="pattern"/>
                    <xs:element ref="definitions"/>
               </xs:sequence>
          </xs:complexType>
     </xs:element>
     <xs:element name="fieldmaps">
          <xs:complexType>
               <xs:sequence>
                    <xs:element ref="fieldmap" maxOc-
curs="unbounded"/>
               </xs:sequence>
          </xs:complexType>
     </xs:element>
     <xs:element name="gfx-objects">
          <xs:complexType>
               <xs:sequence>
```

```
                    <xs:element ref="canvas" minOc-
curs="0"/>
                    <xs:element ref="segments" minOc-
curs="0"/>
                    <xs:element ref="boundedAreas" mi-
nOccurs="0"/>
                    <xs:element ref="coords" minOc-
curs="0"/>
               </xs:sequence>
          </xs:complexType>
     </xs:element>
     <xs:element name="length" type="xs:string"/>
     <xs:element name="mastermodel">
          <xs:complexType>
               <xs:sequence>
                    <xs:element ref="gfx-objects"/>
                    <xs:element ref="tree"/>
                    <xs:element ref="passages"/>
               </xs:sequence>
          </xs:complexType>
     </xs:element>
     <xs:element name="name" type="xs:string"/>
     <xs:element name="object">
          <xs:complexType>
               <xs:sequence>
                    <xs:element ref="gfx-objects"/>
                    <xs:element ref="properties"/>
                    <xs:element ref="states"/>
                    <xs:element ref="readers"/>
                    <xs:element ref="sensors"/>
                    <xs:element ref="passages"/>
                    <xs:element ref="children"/>
               </xs:sequence>
               <xs:attribute name="id" type="xs:string"
use="required"/>
               <xs:attribute name="current_level"
type="xs:string"/>
               <xs:attribute name="inside"
type="xs:string"/>
               <xs:attribute name="next_level_id"
type="xs:string"/>
               <xs:attribute name="vit_type"
type="xs:string"/>
```

```
            <xs:attribute name="level_number"
type="xs:string"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="offset" type="xs:string"/>
    <xs:element name="passage">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="properties" minOc-
curs="0"/>
                <xs:element ref="states" minOc-
curs="0"/>
                <xs:element ref="readers" minOc-
curs="0"/>
                <xs:element ref="sensors" minOc-
curs="0"/>
            </xs:sequence>
            <xs:attribute name="fid"
type="xs:string"/>
            <xs:attribute
name="containing_segment_fid" type="xs:string"/>
            <xs:attribute name="id"
type="xs:string"/>
            <xs:attribute name="segment_fid"
type="xs:string"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="passages">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="passage" minOc-
curs="0" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="pattern" type="xs:string"/>
    <xs:element name="properties">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="property" minOc-
curs="0" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
```

```
    <xs:element name="property">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="value"/>
                <xs:element ref="type"/>
                <xs:element ref="sourcetype" minOc-
curs="0"/>
                <xs:element ref="source" minOc-
curs="0"/>
            </xs:sequence>
            <xs:attribute name="name"
type="xs:string" use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="reader">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="gfx-objects"/>
                <xs:element ref="properties"/>
                <xs:element ref="states"/>
            </xs:sequence>
            <xs:attribute name="id" type="xs:string"
use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="readers">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="reader" minOc-
curs="0" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="rootobject">
        <xs:complexType>
            <xs:choice>
                <xs:element ref="children"/>
                <xs:element ref="object"/>
            </xs:choice>
            <xs:attribute name="id" type="xs:string"
use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="segment">
```

```
          <xs:complexType>
               <xs:attribute name="id"
type="xs:string"/>
               <xs:attribute name="passage"
type="xs:string"/>
               <xs:attribute name="x1"
type="xs:string"/>
               <xs:attribute name="x2"
type="xs:string"/>
               <xs:attribute name="y1"
type="xs:string"/>
               <xs:attribute name="y2"
type="xs:string"/>
               <xs:attribute name="fid"
type="xs:string"/>
          </xs:complexType>
     </xs:element>
     <xs:element name="segments">
          <xs:complexType>
               <xs:sequence>
                    <xs:element ref="segment" maxOc-
curs="unbounded"/>
               </xs:sequence>
          </xs:complexType>
     </xs:element>
     <xs:element name="sensor">
          <xs:complexType>
               <xs:sequence>
                    <xs:element ref="gfx-objects"/>
                    <xs:element ref="properties"/>
                    <xs:element ref="states"/>
               </xs:sequence>
               <xs:attribute name="id" type="xs:string"
use="required"/>
          </xs:complexType>
     </xs:element>
     <xs:element name="sensors">
          <xs:complexType>
               <xs:sequence>
                    <xs:element ref="sensor" minOc-
curs="0" maxOccurs="unbounded"/>
               </xs:sequence>
          </xs:complexType>
     </xs:element>
```

```
    <xs:element name="set">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="name"/>
                <xs:element ref="type"/>
                <xs:element ref="definition"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="sets">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="set" maxOc-
curs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="source" type="xs:string"/>
    <xs:element name="sourcetype" type="xs:string"/>
    <xs:element name="state">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="definition"/>
                <xs:element ref="actiontype"/>
                <xs:element ref="actionstring"/>
            </xs:sequence>
            <xs:attribute name="id" type="xs:string"
use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="states">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="state" minOc-
curs="0"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="tree">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="rootobject"/>
            </xs:sequence>
        </xs:complexType>
```

```
    </xs:element>
    <xs:element name="type" type="xs:string"/>
    <xs:element name="value" type="xs:string"/>
    <xs:element name="view">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="gfx-objects"/>
                <xs:element ref="tree"/>
                <xs:element ref="passages"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="vit-instance">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="fieldmaps"/>
                <xs:element ref="sets"/>
                <xs:element ref="view"/>
                <xs:element ref="mastermodel"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!--W3C Schema erstellt mit XMLSpy v2005 rel. 3 U
(http://www.altova.com)-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
    <xs:element name="definition" type="xs:string"/>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="set">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="name"/>
                <xs:element ref="type"/>
                <xs:element ref="definition"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="sets">
        <xs:complexType>
            <xs:sequence>
```

```
                    <xs:element ref="set" maxOc-
curs="unbounded"/>
               </xs:sequence>
          </xs:complexType>
     </xs:element>
     <xs:element name="type" type="xs:string"/>
</xs:schema>

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!--W3C Schema erstellt mit XMLSpy v2005 rel. 3 U
(http://www.altova.com)-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
     <xs:element name="definition">
          <xs:complexType>
               <xs:sequence>
                    <xs:element ref="name"/>
                    <xs:element ref="offset"/>
                    <xs:element ref="length"/>
               </xs:sequence>
               <xs:attribute name="id" type="xs:string"
use="required"/>
          </xs:complexType>
     </xs:element>
     <xs:element name="definitions">
          <xs:complexType>
               <xs:sequence>
                    <xs:element ref="definition" maxOc-
curs="unbounded"/>
               </xs:sequence>
          </xs:complexType>
     </xs:element>
     <xs:element name="fieldmap">
          <xs:complexType>
               <xs:sequence>
                    <xs:element ref="name"/>
                    <xs:element ref="pattern"/>
                    <xs:element ref="definitions"/>
               </xs:sequence>
          </xs:complexType>
     </xs:element>
     <xs:element name="fieldmaps">
          <xs:complexType>
               <xs:sequence>
```

```
                    <xs:element ref="fieldmap" maxOc-
curs="unbounded"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="length" type="xs:string"/>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="offset" type="xs:string"/>
        <xs:element name="pattern" type="xs:string"/>
</xs:schema>
```

Bibliography

[1] BizTalk RFID Server, Microsoft Corporation, 2009, www.microsoft.com/biztalk/en/us/rfid.aspx.

[2] Bluetooth Special Interest Group, www.bluetooth.org.

[3] BTnodes - A Distributed Environment for Prototyping Ad Hoc Networks, 2007, http://btnode.ethz.ch.

[4] Enterprise Information Architecture for RFID and Sensor-Based Services, Oracle White Paper, 2006, available from: www.oracle.com/technology/products/sensor_edge_server/collateral/Oracle_SES_Technical_White_Paper.pdf.

[5] Epsilon Wi-Fi Module Family Product Brief, G2 Microsystems, 2009, available from: www.g2microsystems.com/downloads/PB_Epsilon_Modules_Final.pdf.

[6] GS1 - The global language of business, www.gs1.org.

[7] Infrared Data Association, www.irda.org.

[8] Java Web Start Technology, http://java.sun.com/javase/technologies/desktop/javawebstart/index.jsp.

[9] Lego Mindstorms, http://mindstorms.lego.com.

[10] Oracle Sensor Edge Server, www.oracle.com/technology/products/iaswe/edge_server.

[11] PostgreSQL - a powerful, open source object-relational database system, www.postgresql.org.

[12] RFID at WINMEC, 2005, http://winmec.ucla.edu/rfid/.

[13] The Savant Version 0.1. MIT-AUTOID-TM-003, Oat Systems and MIT Auto-ID Center, 2002.

[14] Simulink, The MathWorks, Inc., www.mathworks.com/products/simulink/.

[15] Tmote Sky Datasheet, Moteiv Corporation, 2006, available from: www.sentilla.com/pdf/eol/tmote-sky-datasheet.pdf.

[16] WebSphere RFID Premises Server, IBM, 2004, http://www-306.ibm.com/software/pervasive/ws_rfid_premises_server/.

[17] Wi-Fi Alliance, www.wi-fi.org.

[18] XSL Transformations (XSLT) Version 2.0, W3C Recommendation, 2007, available from: www.w3.org/TR/2007/REC-xslt20-20070123/.

[19] ZigBee Alliance, www.zigbee.org.

[20] 1SYNC, Supply chain data synchronization, www.transora.com/home.html.

[21] P. Ackermann, D. Eichelberg and B. Wagner, Visual Programming in an Object-Oriented Framework, Swiss Computer Science Conference, Zurich, Switzerland, 1996.

[22] K. Alexander, T. Gilliam, K. Gramling, M. Kindy, D. Moogimane, M. Schultz and M. Woods, Focus on the Supply Chain: Applying Auto-ID within the Distribution Center, Auto-ID Center, 2002.

[23] Y. Bai, F. Wang and P. Liu, Efficiently Filtering RFID Data Streams, in D. Lee and C. Li, eds., First International VLDB Workshop on Clean Databases, Seoul, Korea, 2007, pp. 54-56.

[24] M. Bauer, C. Becker and K. Rothermel, Location Models from the Perspective of Context-Aware Applications and Mobile Ad Hoc Networks, Personal and Ubiquitous Computing, Springer-Verlag, London, UK, 2002, pp. 322 - 328.

[25] T. Beck, JTemporal temporal framework for Java, 2002, http://jtemporal.sourceforge.net/.

[26] M. Beigl, Special Issue on Location Modeling in Ubiquitous Computing, Personal and Ubiquitous Computing (2002), pp. 311 - 312.

[27] J. Beutel, O. Kasten, F. Mattern, K. Römer, F. Siegemund and L. Thiele, Prototyping Wireless Sensor Network Applications with BTnodes, in H. Karl, A. Willig and A. Wolisz, eds., Wireless Sensor Networks, First European Workshop, EWSN 2004, Springer-Verlag, Berlin, Germany, 2004, pp. 323-338.

[28] C. Bornhoevd, T. Lin, S. Haller and J. Schaper, Integrating Automatic Data Acquisition with Business Processes - Experiences with SAP's Auto-ID Infrastructure, 30st international conference on very

large data bases (VLDB), VLDB Endowment, Toronto, Canada, 2004, pp. 1182-1188.

[29] D. L. Brock, The Electronic Product Code (EPC) - A Naming Scheme for Physical Objects, Auto-ID Labs Whitepaper, MIT-AUTOID-WH-002, Auto-ID Center, 2001, available from: www.autoidlabs.org/uploads/media/MIT-AUTOID-WH-002.pdf.

[30] C. Browne, NonRelational Database Systems - Object Oriented Databases, http://linuxfinances.info/info/oodbms.html.

[31] M. M. Burnett, Visual Programming, in J. G. Webster, ed., Encyclopedia of Electrical and Electronics Engineering, John Wiley & Sons Inc., New York, NY, USA, 1999.

[32] M. M. Burnett, M. J. Baker, C. Bohus, P. Carlson, S. Yang and P. vanZee, Scaling Up Visual Programming Languages, Computer, 1995, pp. 45-54.

[33] R. G. G. Cattell, D. K. Barry, D. Bartels, M. Berler, J. Eastman, S. Gamerman, D. Jordan, A. Springer, H. Strickland and D. Wade, eds., The Object Database Standard: ODMG 2.0, in The Morgan Kaufmann Series in Data Management Systems, Morgan Kaufmann Publishers, 1997.

[34] V. Cechticky, P. Chevalley, A. Pasetti and W. Schaufelberger, A Generative Approach to Framework Instantiation, (2003).

[35] V. Cechticky and A. Pasetti, Generative Programming for Space Applications, Data System in Aerospace (DASIA) Conference, Prague, Czech Republic, 2003.

[36] S. Chakravarthy, ed., Bulletin of the Technical Committee on Data Engineering. Special Issue on Active Databases, IEEE Computer Society, 1992.

[37] D. Chang, L. Dooley and J. E. Tuovinen, Gestalt Theory in Visual Screen Design: A New Look at an Old Subject, Computers in Education: Australian Topics, Australian Computer Society, Inc., Copenhagen, Denmark, 2002, pp. 5-12.

[38] S. Chawathe, V. Krishnamurthy, S. Ramachandran and S. Sarma, Managing RFID Data, 30st international conference on very large data bases (VLDB), VLDB Endowment, Toronto, Canada, 2004, pp. 1189-1195.

[39] A. Chella, M. Cossentino and L. Sabatucci, Tools and patterns in designing multi-agent systems with passi, WSEAS Transactions on Communications, 3 (2004), pp. 352-358.

[40] K. Czarnecki and U. W. Eisenecker, Analysis and design methods and techniques, in K. Czarnecki and U. W. Eisenecker, eds., Generative Programming: Methods, Tools, and Applications, Addison Wesley, 2000, pp. 19-59.

[41] K. Czarnecki and U. W. Eisenecker, What is this book about?, in K. Czarnecki and U. W. Eisenecker, eds., Generative Programming: Methods, Tools, and Applications, Addison Wesley, 2000, pp. 1-16.

[42] F. DeRemer and H. Kron, Programming-in-the large versus programming-in-the-small, International Conference on Reliable Software, Los Angeles, California, 1975, pp. 114 - 121.

[43] Deutsches Institut für Normung e.V. (DIN), DIN 16557-3. Elektronischer Datenaustausch für Verwaltung, Wirtschaft und Transport (EDIFACT); Allgemeine Einführung für Einheitliche Nachrichtentypen (UNSMs), 1994.

[44] S. Domnitcheva, Location Modeling: State of the Art and Challenges, Ubicomp 2001, Atlanta, USA, 2001.

[45] C. Dyreson, F. Grandi, W. Käfer, N. Kline, N. Lorentzos, Y. Mitsopoulos, A. Montanari, D. Nonen, E. Peressi, B. Pernici, J. F. Roddick, N. L. Sarda, M. R. Scalas, A. Segev, R. T. Snodgrass, M. D. Soo, A. Tansel, P. Tiberio and G. Wiederhold, A Consensus Glossary of Temporal Database Concepts, ACM SIGMOD Record, 23 (1994), pp. 52-64.

[46] A. Eisma, Building RFID solutions with the IBM WebSphere RFID Device Infrastructure, Software for Sensor Networks Workshop at the International Conference on Object Oriented Programming, Systems, Languages and Applications (OOPSLA), Portland, OR, 2006.

[47] EPCglobal, Application Level Events 1.1.1 Standard - Part 1, 2009, available from: www.epcglobalinc.org/standards/ale/ale_1_1_1-standard-core-20090313.pdf.

[48] EPCglobal, EPC Information Services Standard v. 1.0.1, 2007, available from: www.epcglobalinc.org/standards/epcis/epcis_1_0_1-standard-20070921.pdf.

[49] EPCglobal, EPC Tag Class Definitions, 2007, available from: www.epcglobalinc.org/standards/TagClassDefinitions_1_0-whitepaper-20071101.pdf.

[50] EPCglobal, EPC Tag Data Standard v. 1.4, 2003, available from: www.epcglobalinc.org/standards/tds/tds_1_4-standard-20080611.pdf.

[51] EPCglobal, EPC Tag Data Translation Standard v. 1.4, 2009, available from: www.epcglobalinc.org/standards/tdt/tdt_1_4-standard-20090610.pdf.

[52] EPCglobal, EPCglobal Architecture Framework v. 1.3, 2009, available from: www.epcglobalinc.org/standards/architecture/architecture_1_3-framework-20090319.pdf.

[53] EPCglobal, Low Level Reader Protocol Standard v. 1.0.1, 2007, available from: www.epcglobalinc.org/standards/llrp/llrp_1_0_1-standard-20070813.pdf.

[54] EPCglobal, Object Naming Service Standard v. 1.0.1, 2008, available from: www.epcglobalinc.org/standards/ons/ons_1_0_1-standard-20080529.pdf

[55] EPCglobal, Reader Management Standard v. 1.0.1, 2007, available from: www.epcglobalinc.org/standards/rm/rm_1_0_1-standard-20070531.pdf.

[56] EPCglobal, Reader Protocol Standard v. 1.1, 2006, available from: www.epcglobalinc.org/standards/rp/rp_1_1-standard-20060621.pdf.

[57] EPCglobal, Review of Hardware Action Group's UHF Generation 2 Protocol Working Group Activities, 2004, available from: www.epcglobalinc.org/standards_technology/EPCTagDataSpecification11rev124.pdf

[58] M. Erwig and B. Meyer, Heterogeneous Visual Languages - Integrating Visual and Textual Programming, 11th International IEEE Symposium on Visual Languages, Darmstadt, Germany, 1995.

[59] D. Estrin, D. Culler, K. Pister and G. Sukhatme, Connecting the Physical World with Pervasive Networks, IEEE Pervasive Computing, 1 (2002), pp. 59-69.

[60] O. Etzion, S. Jajodia and S. Sripada, eds., Temporal Databases: Research and Practice, in Lecture Notes in Computer Science, Vol. 1399, Springer, 1998.

[61] K. Finkenzeller, RFID-Handbook - Fundamentals and Applications in Contactless Smart Cards and Identification, 2nd edition, Wiley & Sons LTD, 2003.

[62] E. Fleisch, Business Perspectives on Ubiquitous Computing, M-Lab working paper (2001).

[63] E. Fleisch and M. Dierkes, Ubiquitous computing: Why Auto-ID is the logical next step in enterprise automation, Auto-ID Center Technical Reports, Auto-ID Center, 2003, available from: www.m-lab.ch.

[64] C. Floerkemeier, EPC-Technologie - vom Auto-ID Center zu EPCglobal, in E. Fleisch and F. Mattern, eds., Das Internet der Dinge - Ubiquitous Computing und RFID in der Praxis: Visionen, Technologien, Anwendungen, Handlungsanleitungen, Springer Verlag, Berlin, 2005, pp. 87-100.

[65] C. Floerkemeier, D. Anarkat, T. Osinski and M. Harrison, PML Core Specification 1.0, Technical Report STG-AUTOID-WH005, Auto-ID Center, 2003, available from: www.autoidlabs.org/uploads/media/STG-AUTOID-WH005.pdf.

[66] C. Floerkemeier and M. Lampe, RFID middleware design - addressing application requirements and RFID constraints, Smart Objects Conference (SOC), Grenoble, France, 2005, pp. 219-224.

[67] C. Floerkemeier, M. Lampe and C. Roduner, Fosstrak: Open Source RFID Software Platform, 2006, www.fosstrak.org.

[68] C. Floerkemeier, M. Lampe and T. Schoch, The Smart Box Concept for Ubiquitous Computing Environments, Smart Objects Conference (sOc), Grenoble, 2003, pp. 118-121.

[69] C. Floerkemeier, C. Roduner and M. Lampe, RFID Application Development with the Accada Middleware Platform, IEEE Systems Journal. Special Issue on RFID Technology, 1 (2007), pp. 82-94.

[70] C. Floerkemeier, R. Schneider and M. Langheinrich, Scanning with a Purpose – Supporting the Fair Information Principles in RFID protocols, in H. Murakami, H. Nakashima, H. Tokuda and M. Yasumura, eds., Ubiquitious Computing Systems. Revised Selected Papers from the 2nd International Symposium on Ubiquitous Com-

puting Systems (UCS 2004). Lecture Notes in Computer Science (LNCS). Lecture Notes in Computer Science (LNCS), Tokyo, Japan, 2004, pp. 214–231.

[71] D. Fritsch, D. Klinec and S. Volz, NEXUS - Positioning and Data Management Concepts for Location Aware Applications, International Symposium on Telegeoprocessing, Sophia-Antipolis, France, 2000, pp. 171-184.

[72] S. Garfinkel and B. Rosenberg, eds., RFID: Applications, Security, and Privacy, Addison-Wesley, 2005.

[73] D. Garlan, D. Siewiorek, A. Smailagic and P. Steenkiste, Project Aura: Toward Distraction-Free Pervasive Computing, IEEE Pervasive Computing, 4 (2002), pp. 22-31.

[74] P. Geach and M. B. Black, Translations from the Philosophical Writings of Gottlob Frege, 3rd edition, Rowman & Littlefield Pub Inc, 1980.

[75] R. Glassey and I. Ferguson, Location Modeling for Pervasive Environments, First UK-UbiNet Workshop, London, UK, 2003.

[76] M. Gorlick and A. Quilici, Visual Programming in the Large versus Visual Programming in the Small, IEEE Symposium on Visual Languages, St. Louis, MO, USA, 1994.

[77] T. R. G. Green and M. Petre, When Visual Programs are Harder to Read than Textual Programs, in G. C. v. d. Veer, M. J. Tauber, S. Bagnarola and M. Antavolits, eds., Human-Computer Interaction: Tasks and Organisation, 6th European Conference on Cognitive Ergonomics, Rome, Italy, 1992.

[78] GS1, Bar Code Types, BarCodes & Identification, GS1, www.gs1.org/barcodes/technical/bar_code_types.

[79] W. Guangming and J. Gonglian, Research and Design of RFID Data Processing Model Based on Complex Event Processing, International Conference on Computer Science and Software Engineering, 2008, pp. 1396-1399.

[80] H. Helson, The Fundamental Propositions of Gestalt Psychology, Psychological Review, 40 (1933), pp. 13-32.

[81] T. Helstrup and R. E. Anderson, Visual discovery in mind and on paper, Memory & Cognition, 21 (1993), pp. 283-293.

[82] C. K. Hess and R. H. Campbell, A Context-Aware Data Management System for Ubiquitous Computing Applications, International Conference of Distributed Computing Systems (ICDCS 2003), Providence, Rhode Island, 2003.

[83] J. Hightower and G. Borriello, Location Systems for Ubiquitous Computing, IEEE Computer, 34 (2001), pp. 57-66.

[84] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler and K. Pister, System architecture directions for networked sensors, 9th Int'l Conf. Architectural Support Programming Languages and Operating Systems (ASPLOS-IX), ACM Press, 2000.

[85] S. Hinske, M. Lampe, N. Yuill, S. Price and M. Langheinrich, Kingdom of the Knights: evaluation of a seamlessly augmented toy environment for playful learning, Proceedings of the 8th International Conference on Interaction Design and Children, ACM, Como, Italy, 2009.

[86] Y. Hu, S. Sundara, T. Chorma and J. Srinivasan, Supporting RFID-based item tracking applications in Oracle DBMS using a bitmap datatype, Proceedings of the 31st international conference on Very large data bases (VLDB), VLDB Endowment, Trondheim, Norway, 2005, pp. 1140 - 1151.

[87] S. Hudson, F. Flannery and C. S. Ananian, CUP - LALR Parser Generator in Java, Department of Computer Science. Technische Universität München., 2003, http://www2.cs.tum.edu/projects/cup/.

[88] Impinj Inc., Speedway Reader, 2008, available from: www.impinj.com/uploadedFiles/Documents/Reader/Speedway_Reader_Brochure_11_08.pdf.

[89] O. Kanoun and H.-R. Tränkler, Sensor Technology Advances and Future Trends, IEEE Transactions on Instrumentation and Measurement, 53 (2004), pp. 1497-1501.

[90] O. Kasten, A State-Based Programming Model for Wireless Sensor Networks, Ph.D. Thesis, No. 17397, Department of Computer Science, ETH Zurich, Zurich, Switzerland, 2007.

[91] O. Kasten and M. Langheinrich., First Experiences with Bluetooth in the Smart-Its Distributed Sensor Network, Workshop on Ubiqitous Computing and Communication at PACT 2001, 2001.

[92] S. H. Kim and J. W. Jeon, Programming LEGO mindstorms NXT with visual programming, International Conference on Control, Automation and Systems (ICCAS), Seoul, Korea, 2007.

[93] W. Kim, Introduction to Object-Oriented Databasesedition, The MIT Press, 1990.

[94] T. Kindberg and J. Barton, A Web-Based Nomadic Computing System, Computer Networks, 35 (2001), pp. 443--456.

[95] G. Klein, S. Rowe and R. Décamps, JFlex - The Fast Lexical Analyser Generator, 2008, http://jflex.de/.

[96] D. Koelma, R. v. Balen and A. Smeulders, SCIL-VP: A multipurpose visual programming environment, ACM/SIGAPP Symposium on Applied Computing, ACM Press, Kansas City, Missouri, United States, 1992, pp. 1188 - 1198.

[97] U. Kubach, Integration von Smart Items in Enterprise-Software-Systeme, in H. Sauerburger, ed., Ubiquitous Computing, HMD - Praxis der Wirtschaftsinformatik, Vol. 229, dpunkt.verlag GmbH, Heidelberg, 2003, pp. 56-67.

[98] M. Lampe, C. Floerkemeier and S. Haller, Einführung in die RFID-Technologie, in E. Fleisch and F. Mattern, eds., Das Internet der Dinge - Ubiquitous Computing und RFID in der Praxis: Visionen, Technologien, Anwendungen, Handlungsanleitungen, Springer Verlag, Berlin, 2005, pp. 69-86.

[99] M. Lampe and C. Flörkemeier, The Smart Box Application Model, in G. Kotsis, ed., Advances in Pervasive Computing, OCG, Vienna, Austria, 2004.

[100] M. Lampe and S. Hinske, The Augmented Knight's Castle - Integrating Pervasive and Mobile Computing Technologies into Traditional Toy Environments, in C. Magerkurth and C. Röcker, eds., Concepts and technologies for Pervasive Games - A Reader for Pervasive Gaming Research, Shaker Verlag, 2007.

[101] M. Lampe and S. Hinske, Integrating Interactive Learning Experiences into Augmented Toy Environments, Pervasive Learning Workshop at Pervasive 2007, Toronto, Canada, 2007.

[102] M. Lampe, S. Hinske and S. Brockmann, Mobile Device based Interaction Patterns in Augmented Toy Environments, in T. Strang, V. Cahill and A. Quigley, eds., Third International Workshop on Per-

vasive Gaming Applications, PerGames 2006, Dublin, Ireland, May 2006, pp. 109-118.

[103] M. Lampe, M. Strassner and E. Fleisch, RFID in Movable Asset Management., in G. Roussos, ed., Ubiquitous and Pervasive Commerce - New Frontiers for Electronic Business, Computer Communications and Networks, Springer-Verlag, 2005, pp. 53-74.

[104] M. Lampe, M. Strassner and E. Fleisch, A Ubiquitous Computing Environment for Aircraft Maintenance, ACM Symposium on Applied Computing, Nicosia, Cyprus, 2004.

[105] M. Langheinrich, Personal Privacy in Ubiquitous Computing - Tools and System Support, ETH Zurich, Zurich, Switzerland, 2005.

[106] M. Langheinrich, RFID and Privacy, in M. Petkovic and W. Jonker, eds., Security, Privacy and Trust in Modern Data Management, Springer-Verlag, 2006.

[107] U. Leonhardt, Supporting Location-Awareness in Open Distributed Systems, PhD Thesis, Departement of Computing, Imperial College of Science, Technology and Medicine University of London, London, 1998.

[108] T. R. Licht, The IEEE 1451.4 proposed standard, IEEE Instrumentation & Measurement Magazine, 4 (2001), pp. 12-18.

[109] D. C. Luckham, The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systemsedition, Addison-Wesley Longman Publishing Co., Inc., 2001.

[110] J. Mark and P. Hufnagel, The IEEE 1451.4 Standard for Smart Transducers, IEEE 1451.4 Standard Working Group, 2009, available from: http://standards.ieee.org/regauth/1451/IEEE_1451d4_Standard_Genl_Tutorial_090104.pdf.

[111] L. V. Massawe, F. Aghdasi and J. Kinyua, The Development of a Multi-Agent Based Middleware for RFID Asset Management System Using the PASSI Methodology, Sixth International Conference on Information Technology: New Generations, 2009, pp. 1042-1048.

[112] F. Mattern, Vom Verschwinden des Computers – Die Vision des Ubiquitous Computing, in F. Mattern, ed., Total vernetzt, Springer-Verlag, 2003, pp. 1-41.

[113] D. L. Mills, RFC 956: Algorithms for Synchronizing Network Clocks, Request for Comments, Internet Engineering Task Force (IETF), 1985, available from: http://tools.ietf.org/html/rfc956.

[114] D. L. Mills, RFC 1305: Network Time Protocol (Version 3), Request for Comments, Internet Engineering Task Force (IETF), 1992, available from: http://tools.ietf.org/html/rfc956.

[115] B. A. Myers, Taxonomies of Visual Programming and Program Visualization, Journal of Visual Languages and Computing, 1 (1990), pp. 97-123.

[116] M. Nemecek, The Peter System - a visual programming tool for easy and quick application creation 1999, www.gemtree.com/peter.htm.

[117] J. V. Nickerson, Visual Programming: Limits of Graphic Representation, IEEE Symposium on Visual Languages, Los Alamitos, CA, USA, 1994, pp. 178-179.

[118] Object Management Group Inc., OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.2, 2009.

[119] P. Oehen, RFID Stack Event Layer, Term Project, Institute for Pervasive Computing, Department of Computer Science, ETH Zurich, Zurich, 2005.

[120] OpenLDAP, OpenLDAP - community developed LDAP software, 1998, www.openldap.org.

[121] M. Palmer, Seven Principles of Effective RFID Data Management, Progress Software, 2004, available from: www.dbis.ethz.ch/education/ws0708/infsyst_lab/rfid_resources/7principles.pdf.

[122] M. Petre and A. F. Blackwell, Mental Imagery in Program Design and Visual Programming, International Journal of Human-Computer Studies, 51 (1999), pp. 7-30.

[123] B. S. Prabhu, X. Su, H. Ramamurthy, C.-C. Chu and R. Gadh, WinRFID: A Middleware for the Enablement of Radiofrequency Identification (RFID)-Based Applications, in R. Shorey, A. L. Ananda, M. C. Chan and W. T. Ooi, eds., Mobile, Wireless, and Sensor Networks: Technology, Applications, and Future Directions, John Wiley & Sons, Inc., 2006, pp. 331-336.

[124] S. Pradhan, Semantic Location, Personal and Ubiquitous Computing, 4 (2000), pp. 213-216.

[125] B. R. Rao, Object-Oriented Databases: Technology, Applications, and Productsedition, McGraw-Hill Companies, 1994.

[126] C. Roduner and C. Floerkemeier, Towards an Enterprise Location Service, International Symposium on Applications and the Internet Workshops (SAINT 2006 Workshops), IEEE Computer Society, Phoenix, Arizona, USA, 2006, pp. 84-87.

[127] M. Roman, C. K. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell and K. Nahrstedt, Gaia: A Middleware Infrastructure to Enable Active Spaces, IEEE Pervasive Computing (2002), pp. 74-83.

[128] K. Römer, Time Synchronization and Localization in Sensor Networks, Ph.D. Thesis, No. 16106, Department of Computer Science, ETH Zurich, Zurich, Switzerland, 2005.

[129] K. Römer, T. Schoch, F. Mattern and T. Dübendorfer, Smart Identification Frameworks for Ubiquitous Computing Applications, Wireless Networks, 10 (2004), pp. 689-700.

[130] S. Sarma, A History of the EPC, in S. Garfinkel and B. Rosenberg, eds., RFID: Applications, Security and Privacy, Addison-Wesley, 2005, pp. 37-55.

[131] S. Sarma, Integrating RFID, ACM Queue, 2 (2004), pp. 50-57.

[132] S. Sarma, D. L. Brock and K. Ashton, The Networked Physical World - Proposals for Engineering The Next Generation of Computing, Commerce & Automatic Identification, 2000, www.autoidcenter.org/research/MIT-AUTOID-WH-001.pdf.

[133] S. Sarma, S. Weis and D. Engels, RFID Systems and Security and Privacy Implications, Workshop on Cryptographic Hardware and Embedded Systems, Springer, 2002, pp. 454-470.

[134] T. Schoch, Concepts and System Structures to Support Collaborating Everyday Items, Ph.D. Thesis, No. 15908, Department of Computer Science, ETH Zurich, Zurich, Switzerland.

[135] T. Schoch, Middleware für Ubiquitous-Computing-Anwendungen, in E. Fleisch and F. Mattern, eds., Das Internet der Dinge - Ubiquitous Computing und RFID in der Praxis: Visionen, Techno-

logien, Anwendungen, Handlungsanleitungen, Springer Verlag, Berlin, 2005, pp. 119-140.

[136] K. D. Schwartz, BizTalk RFID: Making RFID Deployments Easy, Simple and Economical, Microsoft Corporation, 2007, available from: www.microsoft.com/biztalk/en/us/wp-rfid.aspx.

[137] S.-J. Shin, The Logical Status of Diagramsedition, Cambridge University Press, Cambridge, England, 1994.

[138] B. Shneiderman, Direct Manipulation: A Step Beyond Programming Languages, IEEE Computer, 16 (1983), pp. 57-67.

[139] T. Sigaev and O. Bartunov, ltree - a PostgreSQL contrib module for tree-like structures, 2002, www.sai.msu.su/~megera/postgres/gist/ltree/.

[140] J. P. Singh, Development Trends in the Sensor Technology: A New BCG Matrix Analysis as a Potential Tool of Technology Selection for a Sensor Suite, IEEE Sensors Journal, 4 (2004), pp. 664-669.

[141] A. Sloman, Interactions between philosophy and AI: The role of intuition and non-logical reasoning in intelligence, 2nd IJCAI, London, 1971.

[142] D. C. Smith, Pygmalion: An executable electronic blackboard, (1977).

[143] R. T. Snodgrass, M. H. Bohlen, C. S. Jensen and A. Steiner, Transitioning Temporal Support in TSQL2 to SQL3, in O. Etzion, S. Jajodia and S. Sripada, eds., Temporal Databases: Research and Practice, Springer, 1998, pp. 150-194.

[144] J. P. Sousa and D. Garlan, Aura: an Architectural Framework for User Mobility in Ubiquitous Computing Environments, Software Architecture: System Design, Development, and Maintenance (Proceedings of the 3rd Working IEEE/IFIP Conference on Software Architecture), Kluwer Academic Publishers, 2002, pp. 29-43.

[145] M. Stonebraker, Object-Relational DBMS: The Next Wave, Informix Software, CA, available from: http://db.cs.berkeley.edu/papers/Informix/www.informix.com/informix/corpinfo/zines/whitpprs/illuswp/wave.htm.

[146] M. Stonebraker, P. Brown and D. Moore, Object-Relational DBMSs: Tracking the Next Great Waveedition, Morgan Kaufmann Publishers, San Francisco, 1999.

[147] M. Strassner, M. Lampe and U. Leutbecher, Werkzeugmanagement in der Flugzeugwartung - Entwicklung eines Demonstrators mit ERP-Anbindung, in E. Fleisch and F. Mattern, eds., Das Internet der Dinge - Ubiquitous Computing und RFID in der Praxis: Visionen, Technologien, Anwendungen, Handlungsanleitungen, Springer Verlag, Berlin, 2005, pp. 261-277.

[148] X. Su, C.-C. Chu, B. S. Prabhu and R. Gadh, On the Identification Device Management and Data Capture via WinRFID Edge-Server, IEEE Systems Journal. Special Issue on RFID Technology, 1 (2007), pp. 95-104.

[149] Sun Microsystems, Java EE at a Glance, Sun Developer Network (SDN), 2009, java.sun.com/javaee.

[150] SYNFOS, SYNFOS services, www.sinfosweb.de/serviceDE/Default.aspx?LANG=E.

[151] A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev and R. T. Snodgrass, eds., Temporal Databases: Theory, Design, and Implementation, in Database Systems and Applications Series, Benjamin/Cummings Pub. Co., Redwood City, CA, 1993.

[152] F. Thiesse, Architektur und Integration von RFID-Systemen, in E. Fleisch and F. Mattern, eds., Das Internet der Dinge - Ubiquitous Computing und RFID in der Praxis: Visionen, Technologien, Anwendungen, Handlungsanleitungen, Springer Verlag, Berlin, 2005, pp. 101-117.

[153] F. Thiesse, C. Floerkemeier, M. Harrison, F. Michahelles and C. Roduner, Technology, Standards, and Real-World Deployments of the EPC Network, IEEE Internet Computing, 13 (2009), pp. 36-43.

[154] J. R. Tuttle, Traditional and emerging technologies and applications in the radiofrequency identification (RFID) industry, IEEE Radio Frequency Integrated Circuits Symposium, Denver, CO, 1997.

[155] J. Venn, On the Diagrammatic and Mechanical Representation of Prepositions and Reasonings, Philosophical Magazine and Journal of Science (1880).

[156] F. Wang and P. Liu, Temporal Management of RFID Data, 31st VLDB Conference, VLDB Endowment, Trondheim, Norway, 2005.

[157] R. Want, RFID explained: a primer on radio frequency identification technologiesedition, Morgan & Claypool Publishers, 2006.

[158] M. Weiser, The Computer for the Twenty-First Century, Scientific American (1991), pp. 94-100.

[159] M. Wertheimer, W. Köhler and K. Koffka, Gestaltpsychologie, Berlin, 1910.

[160] K. N. Whitley and A. F. Blackwell, Visual Programming: The Outlook from Academia and Industry, in S. Wiedenbeck and J. Scholtz, eds., Proceedings of the 7th Workshop on Empirical Studies of Programmers, Alexandria, VA USA, 1997, pp. 180-208.

[161] J. Yi, M. Heiss, F. Qiuyun and N. A. Gay, A Prototype RFID Humidity Sensor for Built Environment Monitoring, International Workshop on Education Technology and Training and International Workshop on Geoscience and Remote Sensing, 2008, pp. 496-499.

[162] C. Zang and u. Fan, Complex event processing in enterprise information systems based on RFID, Enterprise Information Systems, 1 (2007), pp. 3-23.

Curriculum Vitae

Matthias Lampe

Personal Data

Date of Birth	June 10, 1971
Birthplace	Mannheim
Citizenship	Germany

Education

1982 –1986	*Gymnasium Markdorf*, Germany
1982 –1991	*Gymnasium Schramberg*, Germany
Jun 1991	Abitur
1991 –1992	Military Service, Memmingen, Germany
1992 –1997	*University of Constance*, Undergraduate Student at the Department of Physics and Computer Science
Apr 1995	Vordiplom Physik
1997 –1999	*Portland State University*, Oregon, USA, Graduate Student at the Department of Computer Science
Nov 1999	Master of Science in Computer Science
2002-2008	*ETH Zurich*, Switzerland, Ph.D. Student at the Department of Computer Science

Employment

1997	Web Developer/Assistant Network Administrator at *Fischer Computer Technik GmbH*, Radolfzell, Germany
1998 –1999	Graduate Teaching Assistant at the Department of Computer Science, *Portland State University*, Oregon, USA
1999	Internship in Project Management at *Freightliner Corporation*, Portland, Oregon, USA
1999	Post-Grad Internship at Narita Labs, *Waseda University*, Tokyo, Japan
2000 –2001	Research Assistant at the *Department* of Computer Science, *University of Constance*, Germany
2001 –2002	Lecturer in Software Engineering at the *University of Applied Science St. Gallen* (Fachhochschule St. Gallen), Switzerland
2002 –2008	Research Assistant at the *Department* of Computer Science, *ETH Zurch*, Switzerland