

REMOTE DATA ACQUISITION OF EMBEDDED SYSTEMS USING INTERNET TECHNOLOGIES: A ROLE-BASED GENERIC SYSTEM SPECIFICATION

THÈSE N° 2388 (2001)

PRÉSENTÉE AU DÉPARTEMENT D'INFORMATIQUE

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES TECHNIQUES

PAR

Txomin NIEVA

Ingénieur en systèmes électroniques, Ecole Universitaire de Mondragon, Espagne
de nationalité espagnole

acceptée sur proposition du jury:

Prof. A. Wegmann, directeur de thèse
Prof. K. Aberer, rapporteur
Dr A. Fabri, rapporteur
Prof. H. Kirmann, rapporteur
M. A. Moertlseder, rapporteur

Lausanne, EPFL
2001

À Nathalie

Acknowledgments

First, I express my deepest thanks to my thesis supervisor Prof. Alain Wegmann for his support during this work. He offered me the unique opportunity to learn and experiment and he guided me from the deep black hole up to the light. I extend my deepest thanks to the rest of the members of the jury Prof. Karl Aberer (EPFL-LSIR), Dr Andreas Fabri (INRIA, Sophia Antipolis, France), Prof. Hubert Kirmann (ABB Corporate Research, Baden, Switzerland), and Alfred Mörtlseher (BBV Software Services, Lucern, Switzerland) for accepting the evaluation of my thesis and their invaluable contributions to this work. I also thank Prof. Roger D. Hersch (EPFL-LSP) for his assistance as president of the jury.

During more than three years I had the rare chance of working in both a prestigious university and in a research center of one of the largest industrial companies in the world. I am thankful for the EPFL-ICA and for ABB Corporate Research for this great experience. I am also sincerely thankful for the University of Mondragon to give me the opportunity to come to the EPFL under the frame of the GOIER program. I am grateful toward the people at EPFL-ICA and at ABB Corporate Research, particularly toward Andrey Naumenko, Gil Regev, Pavel Balabko, Guy Genilloud, and Pierre Castori, from EPFL-ICA, and toward Andreas Fabri, Hubert Kirmann, and Otto Preiss, from ABB Corporate Research. I express my eternal gratitude also toward Holly Cogliati for spending such a long time improving my writing English and making my technical articles more readable. I thank also Danielle Alvarez, Angela Devenoge and Jean-Pierre Dupertuis for their administrative and technical support.

As part of my work at ABB Corporate Research I was involved in two major projects. I am deeply grateful to all the members of the WP4 of the ROSIN European project; they brought a real framework to the discussions and the implementation of our hypotheses. I am thankful for the members of the DaVinci project of the railway manufacturer Adtranz, particularly for the members of the System Architecture sub-project.

The work of this thesis could not have been done without the invaluable contributions of my diploma students Monica Perez (the railway equipment modeling expert), Jan Ellerbrock (the XML/XSL expert), Abdendi Benammour (the Jini expert), Jose Carpio (the “roles” expert), Ramzi Bouzerda (the SOAP expert), Alessandro Specchia (the DAS expert) and my semester students Fabrice Wohnrau (the LDAP and WAP expert) and Felix Jaeger (another WAP expert). We made a great team together! I extend my most sincere and deepest thanks to all of them.

Besides the technical and administrative support and the friendship and the collaboration framework of my colleagues at EPFL-ICA and ABB Corporate Research, I had the chance of having many good friends in Lausanne and in Rentería (my original village in Spain). They supported me psychologically during the work of my thesis, giving to me their friendship and company. It is sure that I could have not achieved the work of this thesis without this invaluable support. I will be eternally grateful toward all of them.

A special thanks goes to my family, to my father, mother, brother and sister. I have missed them a lot of times but I always had the warm feeling that they were there all the time, ready to listen to me, to help me and looking forward seeing me again. I feel really lucky to have such a wonderful family.

Finally, my greatest and eternal gratitude goes to my beloved Nathalie. She was my inspiration, my motivation, my expectation, my patience, my desire, ...my everything! Actually, the work of this thesis is not my own work but the work of the great team that we form together and that, I firmly believe, we will form forever.

Abstract

Data Acquisition Systems (DAS) are the basis for building monitoring tools that enable the supervision of local and remote systems. DASs are complex systems. It is difficult for developers to compare proprietary generic DAS products and/or standards, and the design of a specific DAS is costly. In this thesis we propose an implementation independent specification, based on conceptual and role-based use case modeling, of a generic architecture for DASs. This generic DAS specification gives DAS developers an abstraction of DASs; it enables them to compare existing DAS products and standards; and it provides the DAS developers that aim to develop a specific DAS with a starting point for the design of a specific DAS. A generic system specification has many advantages. We propose patterns and techniques that are useful for the development of specifications of generic systems. Additionally, the generic DAS specification provides a case study on the development, based on conceptual and role-based use case modeling, of implementation independent specifications of generic systems that demonstrates, by means of an industrial example, the advantages of these techniques for the development of specifications of generic systems.

The work of this thesis has been sponsored by the FNRS (Swiss National Science Foundation)¹, ABB Corporate Research Ltd. (Switzerland), EPFL, and the University of Mondragon.

Keywords: Information System Engineering; Conceptual Modeling; Role-based Use Case Modeling; Data Acquisition Systems; Remote Monitoring Systems; Embedded Systems

¹ In the frame of the NePESM (New Paradigms for Embedded Systems Management) project of the SPP-ICS (Swiss Priority Programme for Information and Communications Structures, 1996-1999) programme.

Version Abrégée

Les systèmes d'acquisition de données (DAS) sont à la base des outils informatiques qui permettent la surveillance locale et à distance des systèmes. Les DASs sont des systèmes complexes. Il est difficile, pour les constructeurs de DASs, de comparer les différents produits propriétaires et génériques et/ou les différentes normes de DASs. En plus, le design d'un DAS spécifique est coûteux. Dans cette thèse nous proposons une spécification d'une architecture générique pour des DASs. Cette spécification générique est indépendante des choix d'implémentation et elle est basée sur la modélisation conceptuelle et la modélisation des cas d'utilisation basée sur des rôles. Cette spécification générique donne aux constructeurs de DAS une abstraction de DASs; elle leur permet de comparer les produits et normes existants; et elle donne aux constructeurs qui veulent concevoir des DAS spécifiques un point de départ pour leur design. Une spécification d'un système générique a plusieurs avantages. Nous proposons quelques patrons et techniques utiles pour la conception des spécifications de systèmes génériques. En outre, notre spécification d'un DAS générique fournit un cas d'étude sur la conception, basée sur la modélisation conceptuelle et la modélisation des cas d'utilisation basée sur des rôles, des spécifications de systèmes génériques. Ce cas d'étude démontre, en utilisant un exemple industriel, les avantages de ces techniques pour la conception des spécifications de systèmes génériques.

Le travail de cette thèse a été financé par le FNRS (Fonds National Suisse de la Recherche Scientifique)², ABB Corporate Research Ltd. (Suisse), l'EPFL, et l'Université de Mondragon.

Mots-clé: Conception de Systèmes d'Information; Modélisation Conceptuelle; Modélisation des Cas d'Utilisation basée sur des Rôles; Systèmes d'Acquisition des Données; Systèmes de Surveillance à Distance; Systèmes Embarqués.

² Dans le cadre du projet NePESM (Nouveaux Paradigmes pour la Gestion des Systèmes Embarqués) financé par le programme SPP-ICS (Programme Prioritaire de Recherche pour les Structures d'Information et de Communication, 1996-1999)

Contents

Acknowledgments	v
Abstract	vii
Version Abrégée	ix
Contents	xi
List of Figures	xv
List of Tables	xvii
Glossary	xix
1. Introduction	1
1.1 Research Context.....	1
1.2 Problem Statement, Goals, and Major Contributions.....	1
1.3 Organization of this Thesis.....	3
2. Context	5
2.1 Introduction.....	5
2.2 Embedded Systems.....	5
2.3 Maintenance, Asset Management, and Condition Monitoring.....	6
2.4 Monitoring and Data Acquisition Systems.....	7
2.5 Measurement.....	8
2.6 Summary.....	8
3. Method	9
3.1 Introduction.....	9
3.2 Conceptual Modeling.....	9
3.3 Role-based Use Case Modeling.....	9
3.4 Patterns, Frameworks and Architectures	10
3.4.1 Patterns	10
3.4.2 Frameworks	11
3.4.3 Architectures.....	11
3.4.4 Patterns vs. Frameworks vs. Architectures.....	12
3.5 External Specification.....	13
3.6 UML	14

3.7	Catalysis.....	15
3.8	Method Overview	16
3.9	Summary.....	17
4.	State of the art of Data Acquisition Systems	19
4.1	Introduction.....	19
4.2	Software Patterns for Data Acquisition Systems.....	19
4.3	OMG's DAIS RFP.....	20
4.4	Data Acquisition Standards	21
4.4.1	OPC	22
4.4.2	IVI.....	23
4.4.3	ODAS	24
4.5	Summary.....	26
5.	Case Study – The RoMain System: A Remote Data Acquisition System Applied to Railway Equipment	27
5.1	Introduction.....	27
5.2	The GLASS System.....	27
5.3	The RoMain System	28
5.4	RoMain Java: Monitoring of all Devices on a Single Train.....	30
5.5	RoMain XML: Monitoring of all Devices on a Fleet of Trains	31
5.6	Plug&Play.....	33
5.6.1	System Plug&Play.....	33
5.6.2	Network Plug&Play.....	36
5.7	Summary.....	38
6.	External Specification of a Generic Architecture for Data Acquisition Systems.....	39
6.1	Introduction.....	39
6.2	Data Acquisition Conceptual Model	39
6.2.1	Device Models	40
6.2.2	Device Items	41
6.2.3	Device Model Monitoring Criteria	41
6.2.4	Device Item Monitoring Criteria	42
6.2.5	Observations & Monitoring Reports	43
6.2.6	Detailed Concepts.....	44
6.2.7	Complete DAS Conceptual Model	47
6.3	Data Acquisition Role-based Use Case Model.....	49
6.3.1	Discover	52
6.3.2	Define Data Access.....	55
6.3.3	Access Data	59
6.3.4	Notify Data Availability	60
6.3.5	Upload Data	61
6.4	Summary.....	62
7.	Discussion	63
7.1	Introduction.....	63
7.2	Conceptual Model.....	63
7.2.1	Device Models vs. Device Items	63
7.2.2	Naming Management.....	64
7.2.3	Composition Management.....	64

7.2.4	Plug&Play.....	66
7.2.5	Physical Values vs. Sampled Values.....	66
7.3	Role-based Use Case Model.....	67
7.3.1	Elementary Roles vs. Actors.....	67
7.3.2	Representation of the System.....	68
7.3.3	System Behavior Modeling.....	69
7.3.4	Broker Pattern.....	71
7.3.5	Administrator-Manager Pattern.....	72
7.3.6	Specification of Role-based use cases.....	73
7.3.7	Specification of Roles.....	73
7.4	Development Process.....	74
7.5	Summary.....	75
8.	Application and Validation.....	77
8.1	Introduction.....	77
8.2	Issuing/Replying a RFP.....	77
8.3	Evaluation of Existing Systems or Proposals.....	78
8.3.1	OMG's DAIS RFP vs. DAS Standards vs. RoMain.....	78
8.4	Design of a New System.....	87
8.4.1	Development of a DAS for Railway Equipment.....	87
8.5	Summary.....	91
9.	Conclusion.....	93
9.1	Introduction.....	93
9.2	Major Contributions.....	93
9.3	Major Findings.....	93
9.3.1	Conceptual Model.....	93
9.3.2	Role-based use case Model.....	94
9.3.3	Development Process.....	94
9.4	Future Work.....	95
Appendix A	RoMain Java vs. RoMain XML.....	97
Appendix B	DAS Role-based Use Cases.....	105
Appendix C	DAS Elementary Roles.....	143
Bibliography.....		169
Curriculum Vitae.....		175

List of Figures

Figure 1 The DAS Nightmare.....	1
Figure 2 And there was Light ... in the DAS World.....	2
Figure 3 Specific DAS Development.....	2
Figure 4 Maintenance and Asset Management.....	7
Figure 5 Monitoring and Data Acquisition System.....	8
Figure 6 Pattern vs. Framework vs. Architecture.....	12
Figure 7 External Specification.....	14
Figure 8 The Catalysis Approach.....	15
Figure 9 Method Overview.....	16
Figure 10 OPC Data Access Model.....	22
Figure 11 IVI-MSS Model.....	24
Figure 12 ODAS Model.....	26
Figure 13 The GLASS System.....	28
Figure 14 The RoMain System.....	29
Figure 15 Monitoring of all Devices on a Single Train.....	30
Figure 16 Monitoring of all Devices on a Fleet of Trains.....	32
Figure 17 Maintenance Information Metadata Structure.....	34
Figure 18 System Initialization.....	35
Figure 19 Jini-enabled System Architecture.....	36
Figure 20 Jini-enabled System.....	37
Figure 21 Data Acquisition Main Packages.....	39
Figure 22 Device Models.....	41
Figure 23 Device Items.....	41
Figure 24 Device Model Monitoring Criteria.....	42
Figure 25 Device Item Monitoring Criteria.....	43
Figure 26 Observations & Monitoring Reports.....	44
Figure 27 Measurement Type.....	44
Figure 28 Mapping Policy.....	45
Figure 29 Time Trigger Condition.....	45
Figure 30 Device Model Event Trigger Condition.....	46
Figure 31 Device Item Event Trigger Condition.....	47
Figure 32 Data Qualifiers.....	47
Figure 33 Complete DAS Conceptual Model.....	48
Figure 34 Device Lifecycle.....	49
Figure 35 Data Acquisition Use Cases.....	50
Figure 36 Data Acquisition Activity Diagram.....	51
Figure 37 Discover Use Case.....	53
Figure 38 Discover Activity Diagram.....	54

Figure 39 Define Data Access Use Case	56
Figure 40 Define Data Access Activity Diagram	58
Figure 41 Access Data Use Case	59
Figure 42 Access Data Activity Diagram	60
Figure 43 Notify Data Availability Use Case.....	61
Figure 44 Upload Data Use Case.....	61
Figure 45 Device Models vs. Device Items.....	63
Figure 46 Example of Model Composition without Functional Model	65
Figure 47 Model Composite Pattern.....	65
Figure 48 Example of Model Composition with Functional Model.....	66
Figure 49 Physical Values vs. Sampled Values.....	67
Figure 50 Elementary Roles vs. Actors	68
Figure 51 Representation of the System in the Use Case.....	69
Figure 52 Example of Modeling of the Interactions across Use Cases	70
Figure 53 Example of Modeling of the Scenarios of Use Cases	71
Figure 54 Example of Representation of the System Broker in the Use Cases.....	71
Figure 55 Example of (Un)Registration of Services in the System Broker.....	71
Figure 56 Example of Comm. between Roles using the System Broker.....	72
Figure 57 Example of Simplified Comm. between Roles using the System Broker.....	72
Figure 58 Administrator-Manager Pattern.....	73
Figure 59 Traditional Development Process	74
Figure 60 Followed Development Process	75
Figure 61 Issuing/Replying a RFP.....	77
Figure 62 RFP vs. Standard vs. Application.....	78
Figure 63 OPC Conceptual Model.....	81
Figure 64 IVI Conceptual Model.....	82
Figure 65 ODAS Conceptual Model	83
Figure 66 RoMain Conceptual Model	84
Figure 67 Railway Equipment DAS Conceptual Model	88
Figure 68 Railway Equipment DAS System Use Case Model.....	90
Figure 69 Communication Performance Comparison	99
Figure 70 One Update vs. Ten Updates Comparison	100

List of Tables

Table 1 Software Patterns for Data Acquisition Systems.....	20
Table 2 Generic DAS Concept Comparison.....	86
Table 3 Generic DAS Functionality Comparison.....	87

Glossary

AMS	Asset Management System
API	Application Programming Interface
COTS	Commercial Off-The-Self
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
DAIS	Data Acquisition from Industrial Systems
DAS	Data Acquisition System
DCOM	Distributed Component Object Model
DSC	Communication Systems Department
DTD	Document Type Definition
EPFL	Swiss Federal Institute of Technology Lausanne
FNRS	Swiss National Science Foundation
GLASS	Global Access for Service and Support
GSM	Global System for Mobile communications
GUID	Global Unique IDentifier
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HVAC	Heating, Ventilation and Air-Conditioning system
ICA	Institute for computer Communications and Applications
IEC	International Electrotechnical Commission
ISO	International Standards Organization
ITU	International Telecommunications Union
IVI	Interchangeable Virtual Instrument
JLS	Jini Lookup Service
JVM	Java Virtual Machine
LAN	Local Area Network
MTTF	Mean Time To Failure
NePESM	New Paradigms for Embedded Systems Management)
ODAS	Open Data Acquisition Standard
ODP	Open Distributed Processing
OLE	Object Linking and Embedding
OMG	Object Management Group
OPC	OLE for Process and Control
PC	Personal Computer
PLC	Programmable Logic Controller
PnP	Microsoft's Plug&Play
PPM	Preventive/Predictive Maintenance
QoS	Quality of Service

RFP	Request For Proposal
RMI	Remote Method Invocation
RM-ODP	Reference Model for Open Distributed Processing
RoMain	Railway Open Maintenance tool
ROSIN	Railway Open System Interconnection Network European project
RPC	Remote Procedure Call
SGML	Standard Generalized Markup Language
SI	International System of Units
SPP-ICS	Swiss Priority Programme for Information and Communications Structures, 1996-1999
SSL	Secure Socket Layer
TCN	Train Communication Network
TCP/IP	Transmission Control Protocol/Internet Protocol
TNM	Train Network Management
UIC	Union Internationale des Chemins de fer
UML	Unified Modeling Language
UPnP	Universal Plug&Play
UTC	Universal Coordinated Time
W3C	World Wide Web Consortium
WLAN	Wireless Local Area Network
XML	eXtensible Markup Language
XSL	eXtensible Style Language

1. Introduction

1.1 Research Context

In the last few years, companies from many business areas have become increasingly interested in maintenance and asset management. A management technique that can be applied for improving maintenance and asset management is condition monitoring. The access to utility data source is essential. Remote monitoring systems have been developed in many business areas such as building [1], power engineering [2] and transportation systems [3] to provide condition-monitoring systems with information about the state of equipment. The kernel of any remote monitoring system is a data acquisition system (DAS), which enables the collection of relevant data. There are many standards for DASs such as OLE for Process and Control (OPC) [4], Interchangeable Virtual Instrument (IVI) [5] and Open Data Acquisition Standard (ODAS) [6], among others. Additionally, the Object Management Group (OMG) has recently issued a Data Acquisition from Industrial Systems (DAIS) Request For Proposal (RFP) [7]. Based on DAS standards, there are many commercial generic DAS products that DAS developers can buy and customize for their specific DAS application. DAS developers have to choose between buying a commercial DAS product and customizing it for their specific requirements or designing from scratch a specific DAS.

1.2 Problem Statement, Goals, and Major Contributions

However, DASs are complex systems. It is difficult for DAS developers to understand DAS standards and/or generic DAS products. As each standard or product uses a different idiom it is also difficult for DAS developers to compare them. Additionally, the development of a specific DAS from scratch is a difficult task that requires high development costs.

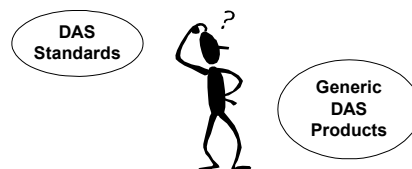


Figure 1 The DAS Nightmare

The main problems are:

- (i) Understanding DAS standards and/or generic DAS products is difficult.

- (ii) Comparing DAS standards and/or generic DAS products is difficult.
- (iii) Designing a specific DAS requires high development costs

We found that a high-level generic abstraction of DASs may help DAS developers to understand and compare the different standards and/or generic products. Therefore, we propose a conceptual model and a role-based use case model of a generic DAS. These models give DAS developers a high level abstraction of DASs. They also provide DAS developers with an implementation independent specification of a generic architecture for DASs. This generic DAS specification enables the comparison of existing standards and products.

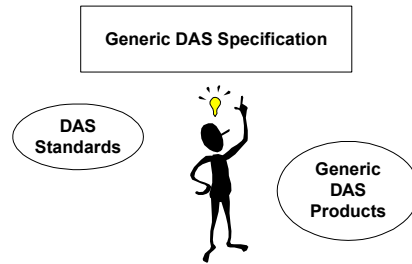


Figure 2 And there was Light ... in the DAS World

Additionally, this generic DAS specification provides the DAS developers that aim to develop a specific DAS with a starting point for the design of a specific DAS. The specification of a generic architecture for DASs reduces the development costs of a specific DAS.

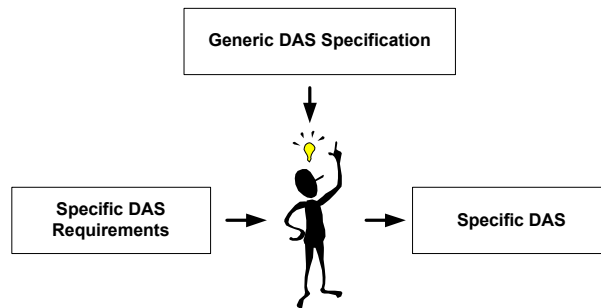


Figure 3 Specific DAS Development

Therefore, the main benefits of this generic DAS specification are:

- (i) Reduction of the time and the costs needed to understand a DAS standard and/or generic DAS product.
- (ii) Enabling of the comparison of the different products and standards
- (iii) Reduction of the time and the costs needed to implement a specific DAS.

We have found that an implementation independent specification of a generic system has many advantages. We propose patterns and techniques to facilitate the development of specifications of generic systems. Additionally, our generic DAS specification provides a case study on the development, based on conceptual and role-based use case modeling, of the specifications of generic systems that demonstrates, by means of an industrial example, the advantages of these techniques for the development of specifications of generic systems.

To summarize, the major contributions of this thesis are:

- (i) We provide an implementation independent specification of a generic architecture for DASs.
- (ii) We propose patterns and techniques to develop, based on conceptual and role-based use case modeling, implementation independent specifications of generic systems.
- (iii) We provide a case study on the development, based on conceptual and role-based use case modeling, of specifications of generic systems.

1.3 Organization of this Thesis

This thesis is organized as follows:

Chapter 2. Context

In this chapter we define relevant concepts related to the problem domain of DASs. We define embedded systems. We define maintenance, condition monitoring and asset management, which are techniques that managers have found give substantial benefits to companies. We define remote monitoring and data acquisition systems, which are the core pieces to enable the application of such techniques. Finally, we define measurement, which is one of the principal processes performed by a DAS.

Chapter 3. Method

In this chapter we explain concepts and techniques regarding the methodology that we followed to obtain an implementation independent specification of a generic architecture for DASs. We describe conceptual and role-based use case modeling as useful techniques for the development of specifications of generic systems. We define the concepts of pattern, framework, and architecture in the software domain, establishing a clear distinction between these terms to avoid any confusion. We introduce the concept of *external specification*, which is the term used from now on to refer to an implementation independent specification. We present the Unified Modeling Language (UML) as the modeling language used in our research work. We give an overview of the Catalysis development process, which is an object-oriented methodology based on UML. Finally, we give an overview of the methodology that we followed to obtain an external specification of a generic architecture for DASs.

Chapter 4. State of the art of Data Acquisition Systems

In this chapter we review the state of the art of DASs. First, we give a list of analysis and design patterns related to the domain of DASs. Second, we give an overview of the OMG's Data Acquisition from Industrial Systems (DAIS) Request for Proposal (RFP), which solicited proposals for standard interfaces to access data within industrial systems by other applications. Third, we describe the most important data acquisition standards: OLE for Process and Control (OPC), Interchangeable Virtual Instrument (IVI) and Open Data Acquisition Standard (ODAS).

Chapter 5. Case Study – The RoMain System: A Remote Data Acquisition System Applied to Railway Equipment

In this chapter we describe the RoMain system, which is a web-based monitoring tool for trains to support maintenance work that we developed. This system is based on the GLASS system, which is a generic system to provide remote monitoring capabilities to a large range of industrial facilities based on Internet technologies. We used the development of the RoMain system as a case study to acquire the required knowledge about DASs. In one sense, the external specification of a generic architecture for DASs is a generalization of the model build for the RoMain system with some extensions and improvements. Part of the work described in this chapter appeared in [3, 8, 9].

Chapter 6. External Specification of a Generic Architecture for Data Acquisition System

In this chapter we describe an external specification of a generic architecture for DASs. This generic DAS specification enables the comparison of existing DAS products and standards. Additionally, it provides the DAS developers that aim to develop a specific DAS with a starting point for the design of a specific DAS. The generic DAS specification is composed of a conceptual model and a role-based use case model of a generic DAS. These models give DAS developers a high-level abstraction of DASs. Parts of the work described in this chapter appeared in [10].

Chapter 7. Discussion

In this chapter we discuss key issues about the conceptual and role-based use case models. We also discuss the development process that we followed in this thesis and we explain the benefits of this development process versus traditional development processes. Parts of the work described in this chapter appeared in [10].

Chapter 8. Application and Validation

In this chapter we explain the applications of an external specification of a generic system. The most direct application of a generic system specification is for the writing of a RFP for a new standard. Another potential application of a generic system specification is for the evaluation of existing systems, standards or RFP responses. We illustrate this by using our generic DAS specification to compare the OMG's DAIS RFP, the different DAS standards and the RoMain system. Finally, a generic system specification can be applied in the development of a particular system. This will significantly reduce the development costs of a specific system. We illustrate this by means of an example of development of a DAS for railway equipment based on our generic DAS specification.

Chapter 9. Conclusion

In this chapter we summarize the major findings and contributions from the actual work. We also point out some future work in this field.

2. Context

2.1 Introduction

In this chapter we define relevant concepts related to the domain of DASs. We define embedded systems. We define maintenance, condition monitoring and asset management, which are techniques that managers have found give substantial benefits to companies. We define remote monitoring and data acquisition systems, which are the core pieces to enable the application of such techniques. Finally, we define measurement, which is one of the principal processes performed by a DAS.

2.2 Embedded Systems

The U.K. Institute of Electrical Engineers defined embedded systems as:

“A general-purpose definition of embedded systems is that they are devices used to control, monitor or assist the operation of equipment, machinery or plant. Embedded reflects the fact that they are an integral part of the system. In many cases their embeddedness may be such that their presence is far from obvious to the casual observer and even the more technically skilled might need to examine the operation of a piece of equipment for some time before being able to conclude that an embedded control system was involved in its functioning.” [11]

In fact, an embedded system is nothing but a specialized computer system. Embedded systems differ from desktop PCs in the following aspects:

- (i) Embedded systems often do not have user displays or keyboards.
- (ii) Embedded systems are usually embedded within larger systems or machines.
- (iii) Embedded systems may not include any operating system.
- (iv) Embedded systems usually have hard constraints (small memory, slow CPU, real-time response and so on).

There are many more – to the order of several magnitudes - embedded systems than desktop PCs. We can find embedded systems almost anywhere: home (microwaves, TVs, etc.), power substations (switches, control systems, etc.), buildings (HVAC, alarm systems, etc.), transportation systems (HVAC, door controllers, brakes, etc.), etc. just to mention a few examples.

This thesis is about remote data acquisition from embedded systems. However, there are no constraints that impede the application of the results of this

thesis to non-embedded systems. This is possible because embedded systems have tighter constraints than non-embedded systems. Additionally, non-embedded systems may provide advanced features that could be used to optimize the acquisition and transmission of data from these systems to consumers of this data such as maintenance management systems, condition monitoring systems and asset management systems.

2.3 Maintenance, Asset Management, and Condition Monitoring

Maintenance improves the reliability and availability of equipment and therefore the quality of service (QoS), which managers have found provides substantial benefits. Maintenance management however makes up anywhere from 15 to 40% of total product cost [12]. Consequently, improving maintenance management can also represent a substantial benefit to companies. Traditionally, there are two major maintenance approaches: Corrective Maintenance and Preventive/Predictive Maintenance (PPM). Corrective Maintenance focuses on efficiently repairing or replacing equipment after the occurrence of a failure. Corrective Maintenance aims to increase the maintainability of equipment by improving the speed of repair, or return to service, after a failure. PPM focuses on keeping equipment in good condition in order to minimize failures; repairing components before they fail. PPM aims to increase the reliability of equipment by reducing the frequency of failures.

Substantial benefits can also be obtained by the intensive use of Asset Management Systems (AMS). Asset Management is defined in [14] as:

“The process of guiding the acquisition, use and disposal of assets to make the most of their service delivery potential (i.e., future economic benefit) and manage the related risks and costs over their entire life.”

Asset management is a task complementary to maintenance. It provides support for the planning and operation phases. Similar to maintenance tasks, in AMSs access to utility data source is essential.

A management technique that can be applied for improving maintenance and asset management is the on-line supervision of the health of the equipment, which is usually known as condition monitoring. Condition Monitoring is defined in [13] as:

“Condition monitoring is a management technique that uses the regular evaluation of the actual operating condition of plant equipment, production systems and plant management functions, to optimize total plant operation.”

Condition monitoring, applied to maintenance tasks, provides necessary data in order to schedule preventive maintenance and to predict failures before they happen. Condition monitoring is based on direct monitoring of the state of equipment to estimate its Mean Time To Failure (MTTF). AMSs will propose or update PPM plans based on the information provided by the condition monitoring systems.

DASs and remote monitoring systems build the infrastructure, shown in Figure 4, to provide condition-monitoring systems with information about the state of equipment.

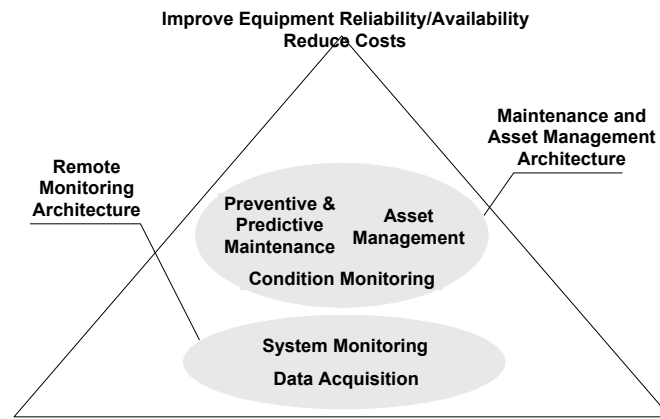


Figure 4 Maintenance and Asset Management

2.4 Monitoring and Data Acquisition Systems

Data Acquisition System (DAS) is defined in [15] as:

“A DAS is a set of hardware and software resources designed to compute the internal representation and then, to deliver to the user the external representation.”

Although this definition is appropriate, it does not reflect certain important aspects of a DAS. We postulate that a DAS is a system that provides:

- Means to discover *knowledge-level* data and access *operational-level* system data
- Means to interpret and process *operational-level* system data, in order to generate system information
- Means to publish system information

In order to clarify this definition we adopted the following definitions according to [16]:

- Discovering: “to obtain sight or knowledge of for the first time”
- Access: “to get at”
- Interpret: “to explain or tell the meaning of”
- Data processing: “the converting of raw data to machine-readable form and its subsequent processing”
- Publish: “to produce or release for distribution”

Therefore, our definition of a DAS is:

“A DAS is a set of hardware and software resources that provides the means to obtain *knowledge-level* data of a system, provides the means to access *operational-level* system data, converts *operational-level* system data to more useful system information and distributes this information to the user.”

Additionally, we define a monitoring system as a system that gives a (client specific) view of the data obtained from a DAS.

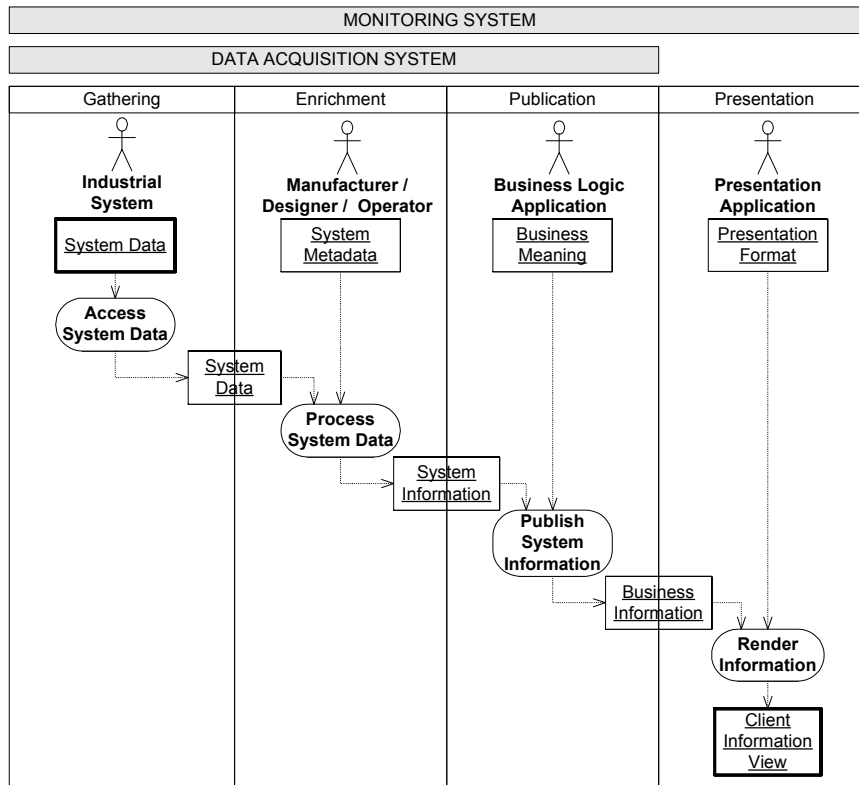


Figure 5 Monitoring and Data Acquisition System

2.5 Measurement

One of the most important processes performed by a DAS is the process of measurement. Measurement is a fundamental method in science that enables one to obtain knowledge of a system. Measurement is informally defined in [17] as:

“Measurement is the process of empirical, objective assignment of numbers to the properties of objects and events of the real world in such a way as to describe them.”

A property is a quality, aspect, or attribute common to all members of a class of objects that may be subject to measurement. From the previous definition it is convenient to highlight that measurement is an empirical process and that this process must be objective.

2.6 Summary

In this chapter we defined relevant concepts related to the domain of DASs. We defined embedded systems. We defined maintenance, condition monitoring and asset management, which are techniques that managers have found give substantial benefits to companies. We defined remote monitoring and data acquisition systems, which are the core pieces to enable the application of such techniques. Finally, we defined the concept of measurement, which is one of the principal processes performed by a DAS.

3. Method

3.1 Introduction

In this chapter we explain concepts and techniques regarding the methodology that we followed to obtain an implementation independent specification of a generic architecture for DASs. We describe conceptual and role-based use case modeling as useful techniques for the development of specifications of generic systems. We define the concepts of pattern, framework, and architecture in the software domain, establishing a clear distinction between these terms to avoid any confusion. We introduce the concept of *external specification*, which is the term used from now on to refer to an implementation independent specification. We present the Unified Modeling Language (UML) as the modeling language used in our research work. We give an overview of the Catalysis development process, which is an object-oriented methodology based on UML. Finally, we give an overview of the methodology that we followed to obtain an external specification of a generic architecture for DASs.

3.2 Conceptual Modeling

A conceptual model is a formal description of a system, from the object perspective, that shows the relevant concepts and relationships that make up this system. Using a conceptual model of a system makes it easier to understand the system, because the model only focuses on the main aspects of the system by hiding low-level details that render it difficult to understand. Boman et al. noted in [28] that:

“An effective approach to analyzing and understanding a complex phenomenon is to create a model of it. By a model is meant a simple and familiar structure or mechanism that can be used to interpret some part of reality. A model is always easier to study than the phenomenon it models, because it captures just a few of the aspects of the phenomenon.”

3.3 Role-based Use Case Modeling

But a conceptual model only specifies the static concepts of a system. We used use case modeling to specify the expected behavior of a system. The artifacts of use case modeling are actors and use cases. The terms *actor* and *use case* are defined in [29]:

“An actor is an idealization of an external person, process, or thing interacting with a system, subsystem, or class. An actor characterizes the interactions that outside users may have with the system.”

“A use case is a coherent unit of externally visible functionality provided by a system unit and expressed by sequences of messages exchanged by the system unit and one or more actors of the system unit. The purpose of a use case is to define a piece of coherent behavior without revealing the internal structure of the system.”

Fowler gives a simpler definition of these terms in [30]:

“An actor is a role that a user plays with respect to the system.”

“A use case is a typical interaction between a user and a computer system.”

Thus, a use case represents an interaction between actors, typically external users or parts of the system, to carry out a functionality of the system as seen from the external point of view.

We used elementary roles rather than actors in the use cases, because this allows us to specify the system independently of architectural choices, requirements, QoS, and/or available technologies specific for a particular system.

3.4 Patterns, Frameworks and Architectures

In this thesis we propose an external specification of a generic architecture for DASs. This generic DAS specification is inspired by several software patterns. Additionally, we extend some of the existing patterns to the domain of DASs and we propose new domain-specific patterns for DASs. We found that the terms *framework* and *architecture* are very often used interchangeably in the industry. In this section we define the concepts of pattern, framework, and architecture in the software domain. We also establish a clear distinction between these terms to avoid any confusion.

3.4.1 Patterns

Patterns, in this context, have their origin in the architectural domain. The first person that used the term of *pattern* was the architect *Christopher Alexander* [18, 19] who defined a pattern as:

“A recurring solution to a common problem in a given context and system of forces.”

Software architects and designers found patterns in the software domain very useful, creating the concept of *software pattern*. In the patterns definition section of the *Patterns Home Page* [20], *Richard Gabriel* provides a clear and concise definition of software pattern:

“Each pattern is a three-part rule, which expresses a relation between a certain context, a certain system of forces which occurs repeatedly in that context, and a certain software configuration which allows these forces to resolve themselves.”

Software patterns can be classified according to many criteria. *Brad Appleton* gives a non-exhaustive major classification of patterns, and a good overview of the essential concepts and terminology of software patterns, in [21]. According to this

classification, software patterns can be classified as:

- *Software Design Patterns*. Software design patterns describe, usually object-oriented, patterns in software designs. *Buschmann et al.* [22] classified software design patterns, according to the level of abstraction they are intended for, into: *Architectural Patterns*, which are high-level patterns that express fundamental structural organizations or schemas for software systems; *Design Patterns*, which are middle-level patterns that provide schemas for refining the subsystems or components of a software system, or the relationships between them, describing commonly recurring structures of communicating components that solve a general design problem within a particular context; and *Idioms*, also called *coding* or *programming* patterns, which are low-level patterns, specific to a programming language, that describe how to implement particular aspects of components or the relationships between them using the features of a given language.
- *Software Analysis Patterns*. Software analysis patterns describe a common construction in business modeling.
- *Organization Patterns*. Organization patterns describe patterns for the structuring of organization or projects.
- *Process Patterns*. Process patterns describe patterns in the process of software design.
- *Domain-Specific Patterns*. Domain-specific patterns describe patterns in a specific domain.

In this thesis we are concerned with software analysis patterns, software design patterns and DAS domain specific patterns that are independent from a specific programming language.

3.4.2 Frameworks

A framework defines a basic generic structure of things and their relationships within a particular domain that can be instantiated to the creation of many similar systems. The concept of frameworks in software is closely related to patterns and object-oriented technology. *Ralph Johnson* [23] defined software framework as:

“A framework is a reusable design expressed as a set of abstract classes and the way their instances collaborate. It is a reusable design for all or part of a software system.”

A software framework provides a reusable mini-architecture, within an application domain, of a generic system that may be applied to the development of many specific systems.

3.4.3 Architectures

Architecture is a concept that originally comes from the creation of building structures. Today, this term is also employed to refer to the creation of the structure of any kind of system, including software systems being then called *software architecture*.

Shawn and Garland [24] defined software architecture as:

“The architecture of a software system defines that system in terms of computational components and interactions among those components.”

In this context a component is anything that can be used as a part of software systems such as databases, communication middlewares, client-servers, application servers and so on. In some communities, people use interchangeably the terms of framework and architecture. However, in the object-oriented community these concepts have different meanings, as pointed out by *Jean-Philippe Martin-Flatin* in [25]. Software architecture refers to the collection of models devised at the analysis and high-level design phases of an application. A software architecture is abstract and independent of a specific programming language. A software framework can be seen as an implementation of a software architecture that provides a set of programming language specific template classes. Software developers can use a software framework by instantiating its template classes into specific classes as part of the development of a specific system.

3.4.4 Patterns vs. Frameworks vs. Architectures

From now on we will refer to the terms *software patterns*, *software frameworks* and *software architectures* as simply *patterns*, *frameworks* and *architectures* respectively. The major differences between patterns, frameworks and architectures are:

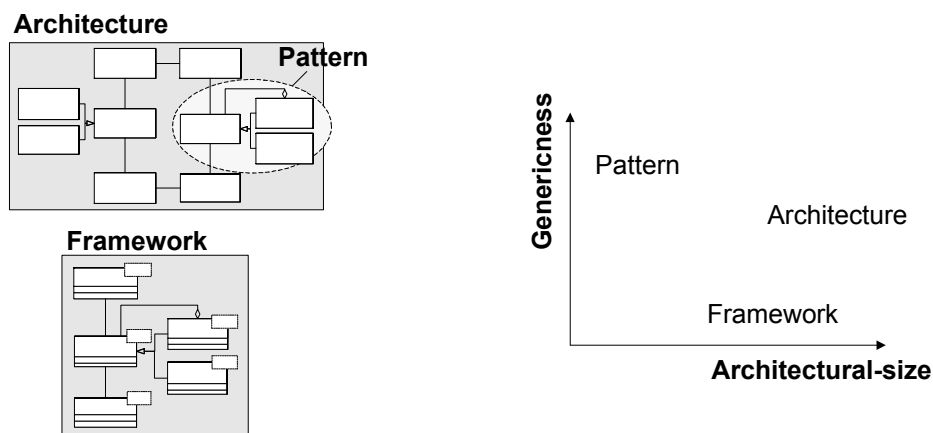


Figure 6 Pattern vs. Framework vs. Architecture

- *Patterns are more abstract than Architectures and Frameworks.* A design pattern describes a solution that may be applied in many application domains, whereas architectures and frameworks are domain specific.
- *Architectures are more abstract than Frameworks.* Frameworks can be seen as an implementation of architectures. Frameworks are programming language specific whereas architectures are independent of a specific programming language.

- *Architectures and Frameworks are bigger structures than Patterns.* Patterns describe solutions to small parts of a system. Architectures and frameworks describe solutions to entire (or significant parts of) systems. Actually, architectures and frameworks may be composed of many software patterns.

3.5 External Specification

In this thesis we propose an implementation independent specification of a generic architecture for DASs. In this section we introduce the concept of *external specification*, which is the term that we will use from now on to refer to an implementation independent specification.

The term *specification* is defined in [26] as:

“An essential technical requirement for items, materials, or services, including the procedures to be used to determine whether the requirement has been met.”

A more software-oriented definition of this term is given by *Alain Wegmann* in [27]:

“The set of constraints satisfied or to be satisfied by the system of interest. The system of interest can be a whole business, an IT application, a software component, etc... A specification is a set of models used for a detailed and precise presentation of a specific context.”

An external specification is the specification of a system from the external point of view. It specifies in detail what the system will do, but not how the system will implement such a functionality. An external specification is therefore independent of a certain technology of a specific implementation. Our external specification of a generic architecture for DASs includes (see Figure 7):

- *A conceptual model of the system.* This model specifies the high-level concepts the system deals with. It specifies a business/domain context for the system.
- *A role-based use case model of the system.* This model specifies the functions the system will perform. It specifies the expected behavior of the system. It specifies a contract between the system and outside users.
- *A collaboration model.* This model includes activity diagrams that describe the interactions across use cases, and sequence diagrams that illustrate example scenarios of use cases.
- *A specification of the roles of the system.* This specification presents the interfaces of the roles of the system and the set of concepts and relationships of the system that the role has to now to carry out its expected behavior.

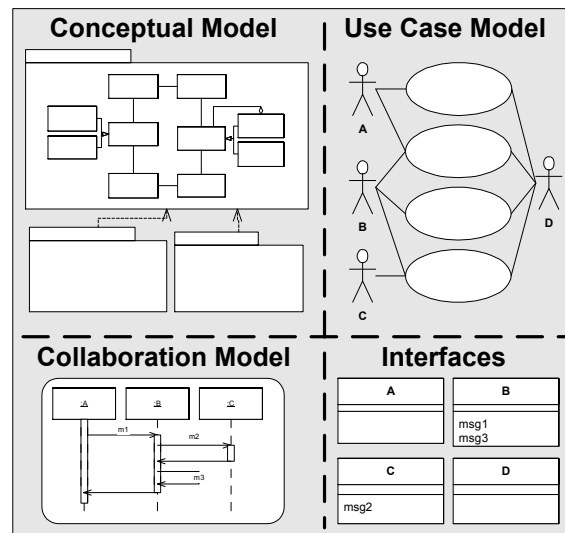


Figure 7 External Specification

An external specification neither specifies the low-level structure of the system, nor how the system internally performs the functions described in the use cases. An external specification of a system remains invariant and independent of specific implementations issues.

An external specification of a system is similar to the information viewpoint of the Reference Model for Open Distributed Processing (RM-ODP). RM-ODP [35] is an ISO/IEC standard and an ITU-T recommendation for the modeling of large distributed systems. RM-ODP provides a rich set of modeling concepts, and five viewpoint languages (enterprise, information, computational, engineering, and technology). To avoid misunderstanding, RM-ODP provides a rigorous definition of the concepts (object, class, interface, template, type, action, behavior, role and so on) commonly encountered in object-oriented models. The Information Viewpoint of RM-ODP focuses on the semantics of the information of a system and the processing of this information. RM-ODP defines information as any kind of knowledge (things, facts, concepts and so on) that is exchangeable among users in a universe of discourse. The result of the information viewpoint is a specification of the system from the point of view of “what the system does”, rather than “how the system implements it”. Therefore, this specification is independent of how a system is built [36].

3.6 UML

In the last few years, many efforts have been made toward developing a single, unified language for the modeling of concepts in software engineering and business domains. The Unified Modeling Language (UML) [29] is mainly the result of the fusion of the concepts from the Booch [31], OMT [32] and OOSE [33] methods.

“The Unified Modeling Language (UML) is a language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling

of large and complex systems.” [34]

A preliminary version of UML appeared in middle 1996. After that, continuous improvements have been made with feedback from the general community and many industrial partners. In the late 1997, the OMG adopted UML 1.1 as standard modeling language for object-oriented analysis and design.

3.7 Catalysis

Catalysis [38] is a standards-based methodology for the systematic development of object and component-based systems. Catalysis unifies the concepts of objects, frameworks and components. It provides methods and techniques for component-based development, high-integrity design, object-oriented design and reengineering. Catalysis uses the UML as notation.

The basic concepts in Catalysis are the *object*, which represents a cluster of information, and the *action*, which represents anything that happens. Catalysis places both concepts on an equal footing.

The Catalysis approach supports three levels of description, as shown in Figure 8:

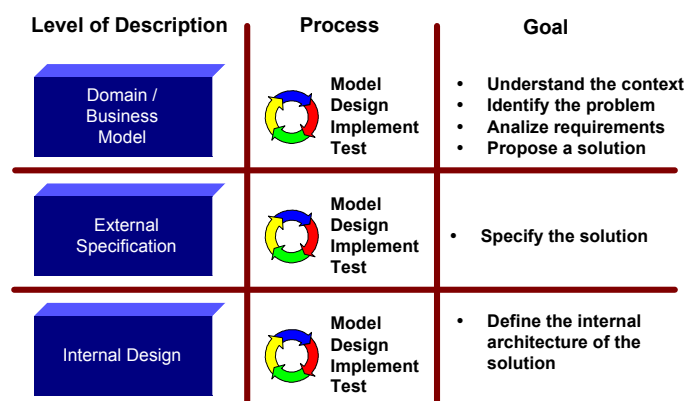


Figure 8 The Catalysis Approach

- *Domain/Business Model*. The main goals are to understand the context (domain terminology, business processes, roles and collaborations), identify the problem, analyze the business requirements, and propose a solution. A business model is a model that describes the business through real-world objects and their interactions. An “*as-is*” business model describes the context of the business as it is currently (this model makes sense only if the business currently exists). A “*to-be*” business model describes the context of the business as it has to be in the future.
- *External Specification*. The main goal is to specify the solution, specify the scope of the different components, define their responsibilities, define the component/system interfaces and specify the desired component operations.

- *Internal design.* The main goal is to define the internal architecture of the solution, define internal components and collaborations and design the insides of the system.

For each level of description Catalysis proposes a recursive, non-linear, iterative and parallel process consisting of the modeling, designing, implementation and testing. Depending on the sequence of deliverables best suited for a specific project, different routes though the method can be taken.

3.8 Method Overview

To obtain the external specification of a generic architecture for DASs we carried out the following steps, shown in Figure 9:

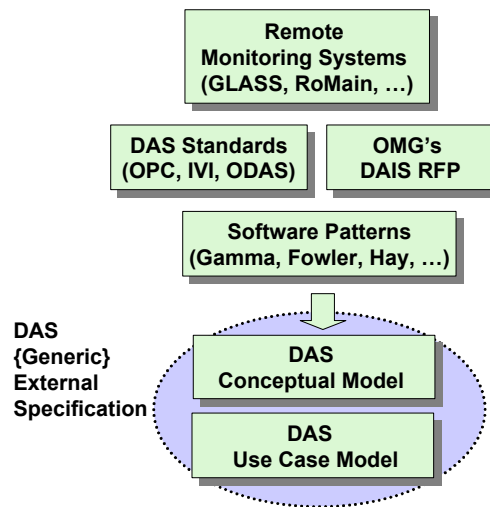


Figure 9 Method Overview

- (i) *We analyzed some remote monitoring systems.* We analyzed several remote monitoring systems from different business domains such as building, power engineering, and transportation systems. We also analyzed a DAS for railway equipment that we developed. During the development of such a system we used our own variation, which puts emphasis on role-based use case modeling, of the Catalysis development process.
- (ii) *We analyzed some DAS standards.* We analyzed different DAS standards such as such as OLE for Process and Control (OPC), Interchangeable Virtual Instrument (IVI) and Open Data Acquisition Standard (ODAS).
- (iii) *We analyzed the OMG's Data Acquisition from Industrial System (DAIS) RFP.* This RFP solicited proposals for standard interfaces to access data within industrial systems by other applications.
- (iv) *We enhanced this specification with several software patterns.* We found that some existing software patterns (e.g. *Composite* and *Broker*) give an effective proven solution to some problems that appeared in the specification of DASs. Additionally, we proposed new patterns (e.g. *Model Composite* and *Administer-Manager*) for DASs.

- (v) *We generalized this specification to allow its use in different domains.* Any domain specific concept was generalized to generic concepts in order to allow its use in different domains. Domain specific details have been removed from the specification.

3.9 Summary

In this thesis we propose an external specification of a generic architecture for DASs. In this chapter we explained concepts and techniques regarding the methodology that we followed to obtain an external specification of a generic architecture for DASs. We described conceptual and role-based use case modeling as useful techniques for the development of specifications of generic systems. We defined the concepts of pattern, framework, and architecture in the software domain, establishing a clear distinction between these terms to avoid any confusion. We introduced the concept of *external specification*. We presented the Unified Modeling Language (UML) as the modeling language used in our research work. We gave an overview of the Catalysis development process, which is an object-oriented methodology based on UML. Finally, we gave an overview of the methodology that we followed to obtain an external specification of a generic architecture for DASs.

4. State of the art of Data Acquisition Systems

4.1 Introduction

In this chapter we review the state of the art of DASs. First, we give a list of analysis and design patterns related to the domain of DASs. Second, we give an overview of the OMG's Data Acquisition from Industrial Systems (DAIS) Request for Proposal (RFP), which solicited proposals for standard interfaces to access data within industrial systems by other applications. Third, we describe the most important data acquisition standards: OLE for Process and Control (OPC), Interchangeable Virtual Instrument (IVI) and Open Data Acquisition Standard (ODAS).

4.2 Software Patterns for Data Acquisition Systems

In this thesis we propose an external specification of a generic architecture for DASs that gives developers analysis and design patterns for the development of such systems. Analysis patterns are high-level patterns that describe a common construction in business modeling. Design patterns are intermediate level patterns that describe design solutions to build a specific system. In this section we compile some analysis and design patterns related to the domain of DASs. We classified these patterns according to the following classification:

- *Architectural Patterns*. Architectural patterns describe high-level partitions of a system into subsystems and their dependencies.
- *Behavioral Patterns*. Behavioral patterns describe how objects interact and distribute responsibility.
- *Diagnostic Patterns*. Diagnostic patterns describe how diagnostic messages are represented and processed.
- *Input and Output Patterns*. Input and Output patterns describe issues related to the gathering of data and its processing.
- *Knowledge Management Patterns*. Knowledge patterns describe how knowledge can be represented in a system.
- *Naming Patterns*. Naming patterns define how to identify objects.
- *Observations and Measurements Patterns*. Observations and measurement patterns describe how to represent observations and measurements of a system.
- *Structural Patterns*. Structural patterns define how to compose objects.

- *Temporal Patterns*. Temporal patterns define how to deal with objects that change over time.

We include the original source of the patterns; additionally, a brief description of them and their applicability can be found in [39].

Table 1 Software Patterns for Data Acquisition Systems

Category	Related Patterns
Architectural	Two-Tier Architecture, Three-Tier Architecture, Presentation and Application Logic [40]
Behavioral	Observer, Mediator, Iterator, Chain of Responsibility [41]; Broker, Client-Dispatcher-Server, Forwarder-Receiver, Publisher-Subscriber [22]; Reactor [42]
Diagnostic	Diagnostic Logger, Diagnostic Context, Typed Diagnostic [43]; Whole Value, Exceptional Value, Diagnostic Query [42]
Input and Output	MML, IO Gatekeeper, Mind Your Own Business, IO Triage, Timestamp, Who Asked?, George Washington Is Still Dead, Bottom Line, Five Minutes of No Escalation Messages, Shut Up and Listen, Pseudo-IO, Beltline Terminal, Audible Alarm, Alarm Grid, Office Alarms, Don't Let Them Forget, String a Wire, Raw IO [44]
Knowledge Management	Knowledge Level [45]
Naming	Name, Identification Scheme [40]
Observations & Measurements	Quantity, Conversion Ratio, Compound Units, Measurement, Observation, Subtyping Observation Concepts, Protocol, Dual Time Record, Rejected Observation, Active Observation / Hypothesis / Projection, Associated Observation, Measurement Protocol, Range, Phenomenon with Range [40]
Structural	Facade, Proxy, Bridge, Composite [41]; Whole-Part [22]
Temporal	Temporal Property, Temporal Association, Snapshot [44]

In our generic DAS specification we used some of these patterns (such as *Composite* and *Broker*). Most of the patterns listed in Table 1 may be used in the design phase of a specific system.

4.3 OMG's DAIS RFP

In this section we give an overview of the Data Acquisition from Industrial Systems (DAIS) Request For Proposal (RFP) [7], issued by the OMG in January 1999. This RFP solicited proposals for standard interfaces to access data within industrial systems by other applications. The goal of this RFP is to provide *operational-level* and *knowledge-level* data in a common format to applications running in a heterogeneous, distributed computing environment. This RFP solicited proposals of standard interfaces covering the following functionalities:

- *Discovery of Remote System and Device Schema.* “Mechanisms for discovering accessible remote devices, measurements, discrete/incremental information, permissible ranges and/or sets of values, alarms and industrial system sourced events. The requests for discovery could be triggered on-demand, or based on time, exception and/or event. A client system could register to receive notifications of changes of the composition from the device...A means shall be provided for a client system to determine the data types and quantities (i.e. cardinality) of data elements available from a particular entity within an industrial system, as well as the identifiers and some of the semantics associated with those data elements.”
- *Defining Data Access Request.* “Mechanisms for defining (and deleting) a set of data and how the set of data should be retrieved. Data sets are collections of data, defined by the client, by a third party, or pre-existing data on the device, that are transferred in response to an event or single read request. The request for data retrieval can be triggered on-demand, or based on time, exception and/or event. A client could register to receive event notifications for the availability of the data requested.”
- *Data Access/Retrieval.* “Mechanisms to define immediate data access retrieval upon request. The data elements transferred may be simple or structured types. A client could define a set of data to be retrieved at a time.”
- *Event Notification for Availability of Data.* “Mechanisms to allow the industrial system broadcasting events outside itself to which clients can subscribe in order to receive a notification that new data is available to be accessed.”
- *Event Driven Data Upload.* “Mechanisms to define event driven data retrieval sequence, by which data delivery can be done automatically upon the occurrence of a notification for availability of data.”

This RFP also requested proposals for standard data types to record information such as:

- *Measurements.* “A measurement is a specific value or set of values measured at a specific time and/or associated with a specific context within an activity.”
- *Discrete and incremental information.* “Discrete information can take one value of an enumerated set of values. Incremental information is used to indicate a variation from a reference value.”
- *Alarms.* “Alarms are indications of a certain state of the system.”
- *Timestamps.* “Timestamps record the time when, e.g., a measurement has been taken.”
- *Identifiers.* “Unique identifiers uniquely identify entities on the system.”

4.4 Data Acquisition Standards

In this section we describe the most important standards related to data acquisition systems.

4.4.1 OPC

The OLE for Process Control (OPC) standard provides clients with a common way to access heterogeneous data sources from the plant floor, and/or from databases in a control room, enabling the integration of data, in a transparent way, into their information systems.

The first draft version of the OPC specification was released in December 1995. Since 1996, the development of the OPC standard has been led by the OPC Foundation [4], a non-profit industry consortium of major players in the process control industry (Siemens, Fisher-Rosemount, Rockwell, and others) working in cooperation with Microsoft that currently has over 220 members around the world.

OPC draws a line between hardware providers and software developers. It provides a mechanism to provide data from a data source and communicate the data to any client application in a standard way. A vendor can develop a highly optimized proprietary server to communicate to the data source, and provide the server with an OPC interface to allow OPC clients to access their devices.

The OPC specification is based on Microsoft COM/DCOM [46, 47] technology. OPC defines a set of standard COM interfaces to provide the following functionalities:

- *Access to Online Data.* Mechanisms that allow OPC clients to efficiently read/write *data* from/to an OPC server.
- *Handling of Alarms and Events.* Mechanisms that allow OPC clients to subscribe to be notified of the occurrence of specified *events* and *alarm* conditions.
- *Access to Historical Data.* Mechanisms that allow OPC clients to read, process and edit *historical data*.

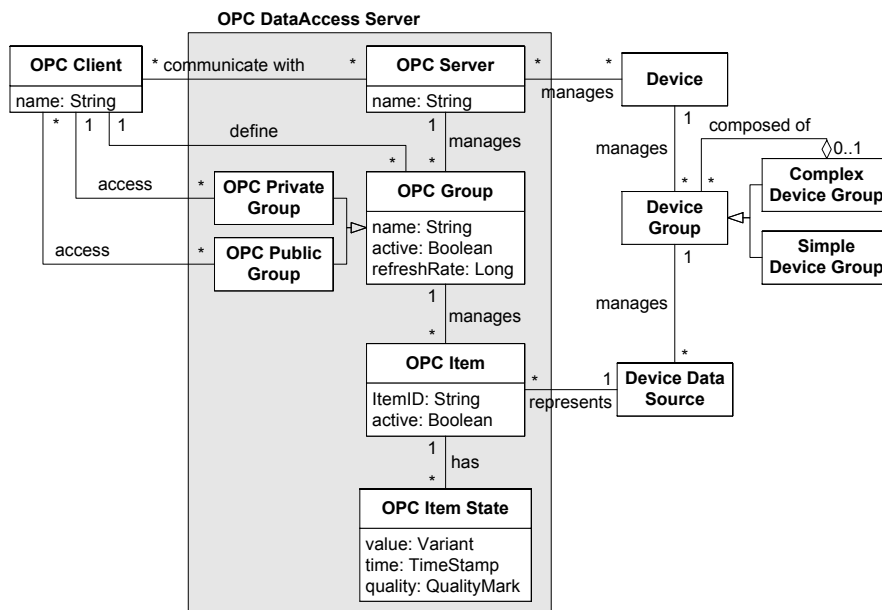


Figure 10 OPC Data Access Model

Other functionalities such as *security*, *batch* and *historical alarm and event data access* will be addressed in future releases of the OPC standard.

The OPC standard gives the following benefits:

- *Reduction of manufacturing costs.* Hardware manufacturers only have to develop a set of software components (the OPC server) to allow heterogeneous clients to access data in a common way. They do not have to provide customers with different drivers to access their devices from many platforms.
- *Simplification of the development of client applications.* Client applications developers may write applications easily regardless of proprietary drivers.
- *Interchangeability of devices.* We may exchange a device for another device if the OPC interface remains the same.

The *OPC Data Access model* is described in Figure 10. An *OPC Server* object manages a set of *OPC Groups*. Each *OPC Group* is a logical container that manages a set of *OPC Items*, each of one represents a single *Device Data Source* with a *value*, a *timestamp* and a *quality mark*. A *Device Data Source* belongs to a *Device* and is physically grouped with related data sources within a *Device Group*. An *OPC Server* can manage several *Devices*. An *OPC Client* communicates with one or several *OPC Servers*. The client is responsible for defining in the server side the *OPC Groups* and the *refreshing rate*. A client can define a *Public Group* that can be shared by several clients. If the group is only defined by and for a client, then it is called *Private Group*.

4.4.2 IVI

The Interchangeable Virtual Instrument (IVI) standard provides clients with standard drivers to access instruments such as oscilloscopes or digital multimeters, enabling the interchangeability of instruments from different vendors.

The development of the IVI standard is led by the IVI Foundation [5], a non-profit organization that was established, as an initiative of National Instrument, in the spring of 1997 by a consortium of large manufacturers of test instruments (The Boeing Company, GDE Systems, GEC Marconi, GenRad, Lockheed Martin Aeronautical Systems, Lucent Technologies, National Instruments, Northrop Grumman ESSD, Raytheon TI Systems, and Vektrex).

The IVI Foundation defines instrument classes, called IVI drivers, based on the most common functionalities and configurations available in instruments today. Currently available IVI classes include Oscilloscope (IviScope), Digital Multimeter (IviDmm), Function Generator (IviFGen), Switch (IviSwth), and Power Supply (IviPower) classes. In the future, additional classes will be identified and new IVI drivers will be defined.

The IVI standards are built upon the VXIplug&play standard. VXIplug&play is a standard for instrument drivers that defines a set of common functions that drivers should support, and also defines an I/O communication library called VISA. The IVI Foundation took some of these concepts and incorporated them into the IVI standard. The IVI standard defines both ANSI C and Microsoft COM interfaces for instrument

drivers.

IVI instrument interfaces give the following benefits:

- *Reduction of programming time and complexity.* IVI instrument interfaces provide a consistent programming approach for instruments.
- *Reduction of down-time and reduction of maintenance costs.* IVI instruments allow instruments to be swapped with minimal or no test code modifications.
- *Acceleration of the introduction of new products to market.* IVI instruments facilitate the rehost of test code from R&D to manufacturing regardless of instrumentation hardware used.

IVI Measurement and Stimulus System (IVI-MSS)

Besides IVI drivers, the IVI Foundation is also defining a standard to allow the definition of complex measurement systems composed of many instruments. This standard is called IVI Measurement and Stimulus System (IVI-MSS) [48]. The IVI-MSS model is described in Figure 11. An *IVI-MSS Server* may implement many *Role Interfaces*, each of them representing a functionality implemented by a *Conceptual Asset* (a real asset, also called synthetic instrument, or a set of instruments in the case of legacy instruments). A *Conceptual Asset* is controlled by a *Role Control Module*. Typically, an *IVI-MSS Servers* will group a set of functionalities. However, a single *Event Server* is used to record events, a single *IVI Class Factory* is used to create *IVI Classes*, and a single *IVI Configuration Store* is used to configure the system.

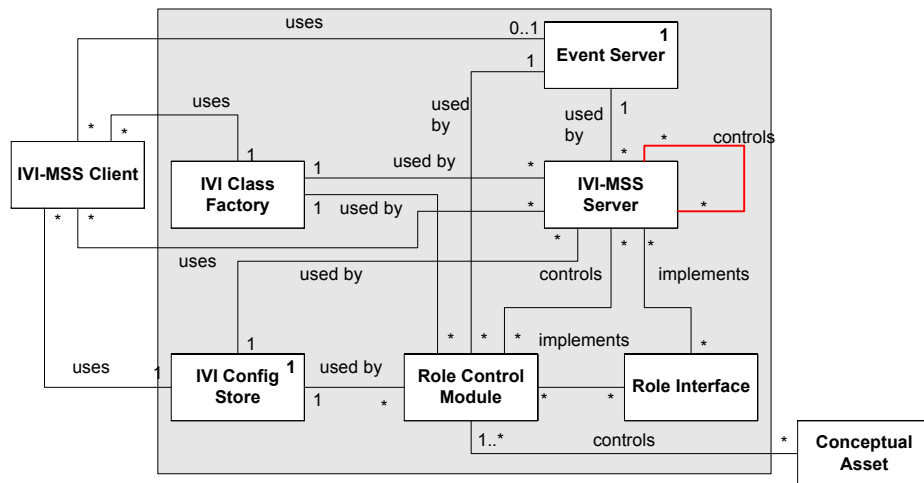


Figure 11 IVI-MSS Model³

4.4.3 ODAS

The Open Data Acquisition Standard (ODAS) standard provides a set of standard interfaces for PC plug-in (PCPI) data acquisition cards.

³ Thanks to Roger P. Oblad, from Agilent Technologies, Inc., for the discussions about the IVI-MSS conceptual model

The development of the ODAS standard is led by the ODAA (Open Data Acquisition Association) [6], a non-profit organization that was formerly established in the summer of 1998 by a consortium of companies (ComputerBoards, Data Translation, Hewlett-Packard, LABTECH, OMEGA Engineering, and Strawberry Tree).

The goal of ODAS is to provide users of data acquisition systems with a universal, open standard that allows interoperability between PC-based data acquisition hardware and software solutions from multiple vendors. The hardware products include, but are not limited to: plug-in boards, PC cards, parallel port systems, USB devices, serial devices and networked systems. The ODAS standard support simultaneous high speed streaming of multiple channels of data at very extremely high speed, sophisticated counter/timer functionality and the capability to perform very high speed control applications.

The ODAS standard is based in Microsoft COM technology. ODAS defines a set of standard COM interfaces to provide the following functionalities:

- *Analog In/Out*. It defines methods to read and write analog inputs and outputs, respectively. It supports multiple channels, synchronous/asynchronous operation, A/D conversions, internal/external clocking capabilities, notification of events, and so on.
- *Digital In/Out*. It defines methods to read and write digital inputs and outputs respectively. It supports high-speed timed input/output, simultaneous reads/writes, bi-directional lines, and so on.
- *Counter/Timer*. It defines methods used for counting/timing. Currently, the specification is under development.

The ODAS standard gives the following benefits:

- *Faster time to market, cost savings*. The ODAS standard allows vendors to develop a single version of their driver that communicates with all other compliant hardware and software. The costs of developing multiple I/O drivers disappear and products can be delivered faster to market.
- *Focus on value-add functionalities*. Vendors can focus their efforts on adding value to their hardware and software products, rather than on implementing many drivers for the same products.
- *More robust & reliable products*. By making use of a single, well-designed standard, vendors will be assured of a high quality, reliable, robust interface to all compliant products.

The ODAS model is described in Figure 12. An *ODAS Manager* manages many *ODAS Drivers*. An *ODAS Driver* manages many *ODAS Subsystems*. An *ODAS Subsystem* is an abstract representation for *ODAS Analog Input*, *ODAS Digital Input*, *ODAS Counter/Timer*, *ODAS Analog Output*, and *ODAS Digital Output*. An *ODAS Subsystem* accesses an *I/O Device*, which represents a real input/output device.

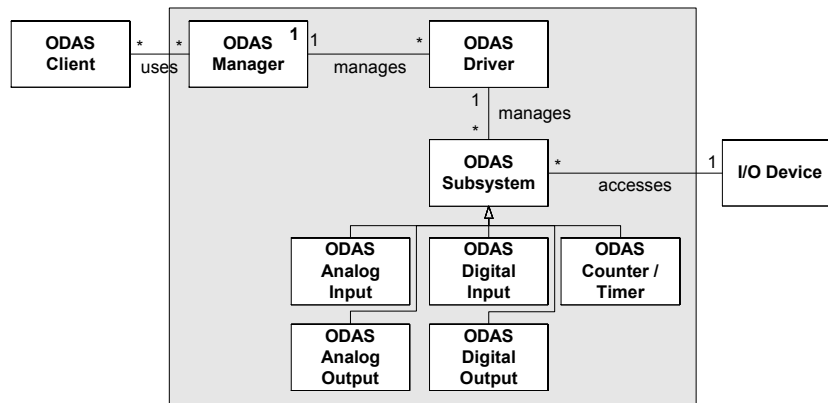


Figure 12 ODAS Model

4.5 Summary

In this chapter we reviewed the state of the art of DASs. We gave a list of analysis and design patterns that are commonly used in the design and development of DASs. In this thesis we propose an external specification of a generic architecture for DASs. An external specification of a generic system can be seen as a large domain-specific analysis pattern. Additionally, in this thesis we propose new patterns for DASs. We also explained the OMG's Data Acquisition from Industrial Systems (DAIS) Request for Proposal (RFP). We found this RFP very useful. We used this RFP as a guideline to specify a generic DAS. Our generic DAS specification is not a response to this RFP as it does not provide, as requested by this RFP, CORBA interfaces for DAS. Rather, our generic DAS specification is complementary to the OMG's DAIS RFP, as our specification describes the concepts related to any DAS and the DAS functionalities by means of conceptual and role-based use case modeling. Our specification remains at a high level of abstraction, being possible to specify the OMG's DAIS RFP and its responses using such a specification. We described the OPC, IVI and ODAS standards. We put forth that most standards have similar concepts and provide similar functionalities. However, they use different naming conventions and they describe their functionalities in different ways. This makes it rather difficult to compare them.

5. Case Study – The RoMain System: A Remote Data Acquisition System Applied to Railway Equipment

Parts of this work appeared in [3, 8, 9].

5.1 Introduction

In this chapter we describe the RoMain system, which is a web-based monitoring tool for trains to support maintenance work that we developed. This system is based on the GLASS system, which is a generic system to provide remote monitoring capabilities to a large range of industrial facilities based on Internet technologies. We used the development of the RoMain system as a case study to acquire the required knowledge about DASs. In one sense, the external specification of a generic architecture for DASs is a generalization of the model build for the RoMain system with some extensions and improvements.

5.2 The GLASS System

The Internet, having caused a revolutionary impact on office automation, is currently heavily influencing the industrial automation and information systems. The emergence of the Internet provides a framework for communication with any piece of hardware or software, independently of where it is physically located. Heterogeneous distributed embedded systems, which were commonly isolated in the past, are increasingly connected to networks and integrated within information systems. The management of distributed embedded systems is becoming an immense task for embedded systems providers, operators, and service organizations that want to offer their customers a high quality of service. The interconnection of distributed embedded systems and information systems brings significant benefits and offers new business opportunities. One of the applications of the Internet in the industry is for the remote monitoring of embedded systems. Some examples of these applications can be seen at [1-3, 52-54].

The Global Access for Service and Support (GLASS) system provides remote monitoring capabilities to a large range of industrial facilities, from unmanned substations to large plants or devices on mobile platforms like trains or ships [2]. The GLASS system, developed by ABB Corporate Research Ltd. in Switzerland, is based on Internet technologies and uses Common Off-The-Self (COTS) technology

wherever possible. It was initially developed for power substations and then generalized to almost any industrial facility. We closely collaborated in the application of the GLASS system in the railway domain, as part of the ROSIN European project.

The GLASS system, shown in Figure 13, consists of a server and many clients interconnected through a secure TCP/IP network (typically the Internet or an Intranet). The server is a PC connected to a proprietary fieldbus that retrieves raw data from the devices of the industrial facility. The TCP/IP link allows clients to retrieve the monitoring information of the industrial facility from the server. The clients are standard web browsers with applets that visualize the data in the appropriate way. The server and the client applications have been developed in Java.

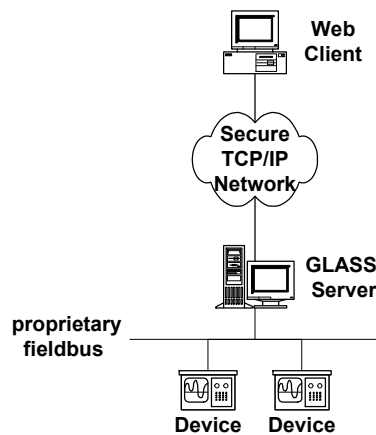


Figure 13 The GLASS System

The GLASS server is responsible for the management of the so-called *Proxy* objects. A *proxy* object is a representative of a *real world object*, that means, of an industrial device. A *proxy* object is responsible for gathering the real data from the device it represents. Additionally, a *proxy* object can extend a device, e.g., with sophisticated algorithm for the prediction of maintenance related events. In the GLASS system, a specific *proxy* class is implemented for each type of device.

5.3 The RoMain System

We developed, in the frame of the Railway Open System Interconnection Network (ROSIN) European project, a web-based monitoring tool for trains that supports maintenance work. This monitoring tool was called Railway Open Maintenance tool (RoMain) [3, 8]. The objective of this tool was not to replace the existing control network, but rather to enhance it with a parallel, low-cost, on-line data network for railways, in order to support maintenance work. This data network allows maintenance staff to supervise railway equipment from anywhere at anytime. It also enables experts at different locations to collaborate and to anticipate maintenance tasks. The user requirements for such a tool were: (i) ubiquitous access to the information; (ii) low cost; (iii) user-friendly interface with textual and graphical views of the information; and (iv) easy update of equipment documentation. Taking into account all these requirements, we decided to take an approach based on the Internet. The Internet has had a revolutionary impact on office automation, and now

there is a clear trend towards using Internet technologies for industrial automation. The introduction of Internet technologies for accessing embedded systems is mostly cost driven, thus bringing significant benefits:

- Reduction of the development costs of an application, by enabling the use of Common Off-The-Shelf (COTS) software components.
- Elimination of the costs of a proprietary communication network, by using the common Internet network.
- Reduction of the costs of development of a client application for each different platform, by using a standard web-browser as a single client interface for heterogeneous platforms.
- Elimination of the costs of installing proprietary client applications, as the client interface is a standard web browser usually pre-installed on the client machine.
- Reduction of the costs of maintaining up-to-date equipment documentation, by offering a simple way (hyperlinks) to publish documents accessible immediately from anywhere in the world.
- Reduction of maintenance personal travel costs, by the possibility of ubiquitous access to the information.
- Reduction of maintenance scheduling costs, by the possibility of ubiquitous access to the information at any time.

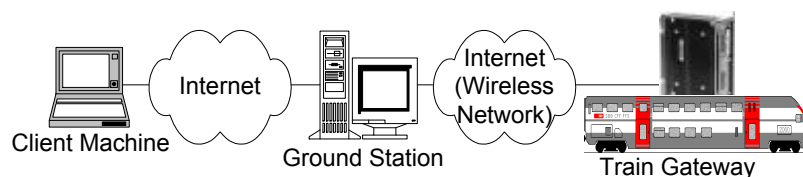


Figure 14 The RoMain System

The architecture of the RoMain system, shown in Figure 14, is composed of:

- Train Gateways* - connected to the train network gather actual train data.
- Ground Stations* - automatically establish connections to *Train Gateways* over wireless networks.
- Client Machines* - run a standard web browser to access train data.

Additionally, *Name and Directory Servers* may provide information about the train component models and train directory, and *Manufacturer Servers* may provide on-line information (e.g. fact sheets, user manuals, or installation instruction) about train components. All these systems are interconnected by means of a secure, wired or wireless, *TCP/IP Network*; usually the *Internet*, or eventually an *Intranet* or *Virtual Private Network*.

In the following sections, we describe two prototypes that we developed of the RoMain system.

5.4 RoMain Java: Monitoring of all Devices on a Single Train

The first prototype has a 2-tier architecture and is based on Java technology and Java Remote Method Invocation (Java RMI) [60-62] as communication protocol. Java, promoted by Sun Microsystems, is basically a programming language and a running platform. The Java language is an easy to learn but quite efficient object-oriented programming language; the Java Virtual Machine (JVM) enables platform independence; and the Java Application Programming Interface (API) provides software developers with a rich library of classes. The Java API provides many ways to enable network connectivity. One of them is Java RMI, which is a communication middleware that enables the communication between objects running in different locations. Java RMI brings a Remote Procedure Call (RPC) like mechanism to execute methods of object located remotely.

The main goal of this prototype was to specify an API for a Data Acquisition System (DAS) on-board a train. This API defines the interface between client and server, and it therefore enables the implementation of new client applications. As the API is object-oriented and as the applications are distributed, we had a choice between different middleware products: DCOM [63, 64], CORBA [63, 65, 66], or Java RMI. We opted for the latter as we strived for a 100% pure Java solution. Therefore, we developed the on-board DAS as a Java RMI server. It offers remote interfaces for, discovering train configuration, and accessing train, vehicle and equipment data - to give just two examples. Based on the API we developed a client system, as a Java RMI client within a Java applet, that uses these remote interfaces to display the current state of a train within a web browser.

Two different updating mechanisms were implemented based on *pull* and *push* technologies. Using *push* technology the update of data is triggered by the server, if and only if, there are changes in that data. Hence a GSM connection is only open during notification subscription and notification, even if there is an arbitrary, long time span in between.

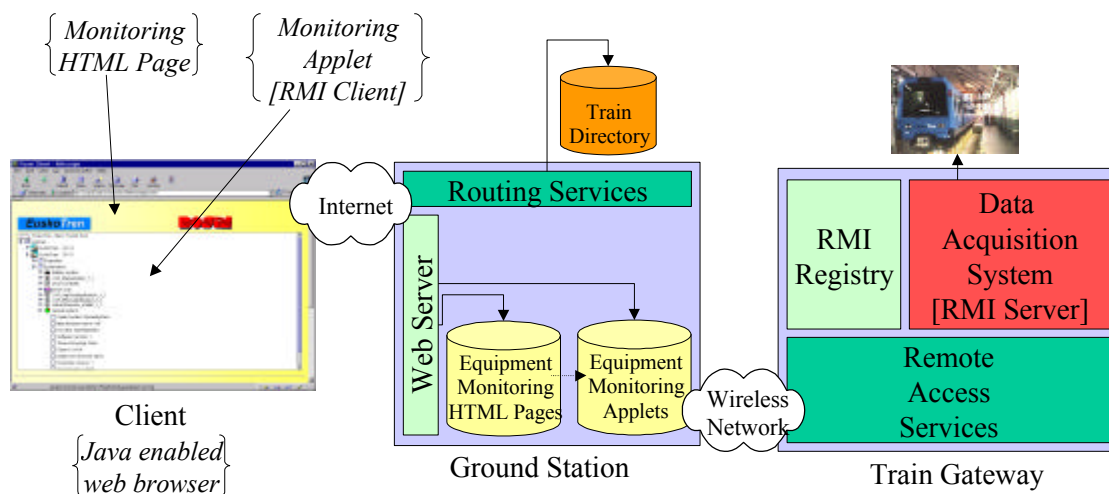


Figure 15 Monitoring of all Devices on a Single Train

The architecture of the system is shown in Figure 15. The remote *Train Gateway* is usually reachable by means of a wireless network, in our case by GSM. A *Ground Station* is responsible for transparently establishing a TCP/IP connection to the remote *Train Gateway* via this wireless network. As the bandwidth of GSM is still relatively low, the downloading of the monitoring HTML page, plus the downloading of the associated monitoring Java applet, is slow. In order to improve the performance of this download, we investigated an alternative: The monitoring HTML page and the Java applet were moved from the remote *Train Gateway* to a ground based web server, in our case to the *Ground Station*. The *Train Gateway* then became a pure data server.

In this prototype we also investigated security issues. As the user must grant the client Java applet certain privileges for stepping out of the sandbox (the restrictive security policy implemented by the browser), he must be able to check whether this Java applet is trustworthy or not. Therefore, the developer of the Java applet must sign it with a digital signature obtained from a certificate authority. We use the same technology that is used to make e-commerce applications more secure.

The main problem with this prototype is the problem of accessing data for a client behind a firewall. This can be partially overcome with HTTP tunneling, but this slows down the communication and the use of server side *push* technology is no longer possible.

During two years we run a demonstration of this prototype. In this demonstration a *Train Gateway* was connected to a train network installed in a laboratory at the CAF (a Spanish train manufacturer) facilities in Beasain (Spain). This train network was exactly the same as the network that was installed on a real train for a demonstration in February'99. The data was collected from real devices, which were interconnected via the Train Communication Network (TCN) [67]. The *Ground Station* was installed at the ICA institute. The only difference with the real demonstration was that the connection of the *Ground Station* with the *Train Gateway* was not done by a wireless network but by an Internet wired connection. This was done in order to reduce costs. However, this was totally transparent for any client using the application.

5.5 RoMain XML: Monitoring of all Devices on a Fleet of Trains

The second prototype has a 3-tier architecture and is based on XML/XSL [64, 68-70] technology and HTTP as communication protocol. The eXtensible Markup Language (XML) is the “de facto” standard for data exchange over the Internet. This new standard was specified, similarly to the Hypertext Markup Language (HTML), by the World Wide Web Consortium (W3C) from a subset of the historical Standard Generalized Markup Language (SGML). XML data looks very much like HTML. However, XML allows developers to define a specific grammar for a specific application. This grammar can be specified by the means of a Document Type Definition (DTD). There are already many standard DTDs that were agreed upon by companies working in the same business domain. These standards enable data exchange among heterogeneous systems. XML data is easy to create, parse, combine and transform into other formats. The eXtensible Style Language (XSL) is an

advanced style sheet language designed for the use with well-formed XML documents. XSL documents contain a series of XSL elements that apply to particular patterns of XML documents. When a particular XML pattern is found, the XSL element outputs a combination of text. The HyperText Transfer Protocol (HTTP) is an application protocol that defines a set of rules for exchanging data over the Internet. HTTP is based on TCP/IP, the transport and session standard protocols of the Internet.

In this prototype we implemented a system with a three-tier architecture to investigate how data from an entire fleet of trains can be integrated, but also to overcome the problem we had with the previous prototype. This allows a component manufacturer to supervise, for example, all door controllers, regardless on which trains they are. We investigated technologies that allow the client to choose among different views, and to receive data combined from *Train Gateways* on a single page. Choosing among different views means that we need a way to separate what is data and what is presentation format. Combining data from different sources means that we have to parse the data and create new data. All these features are easily implemented by the use of XML. As XML also enables the definition of new domain specific markup languages, data can be transmitted together with some metadata that describe them. This makes it easy to parse and combine XML documents. Moreover, the presentation format can be added by means of a separate XSL file, which defines how an XML document should be displayed. Therefore, it is easy to implement different views of the same XML document by providing different XSL files.

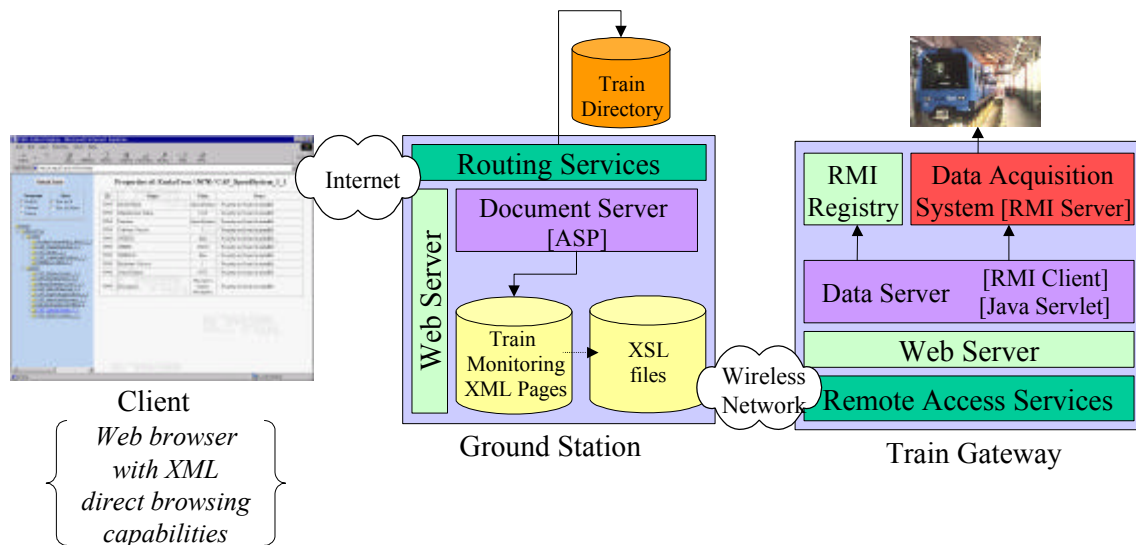


Figure 16 Monitoring of all Devices on a Fleet of Trains

The architecture of the system is shown in Figure 16. It has a three-tier architecture composed of:

- (i) A Java servlet [71] on-board a train that gathers data from the Java RMI server developed for the second prototype and that replies to an HTTP request of data with XML documents giving the requested information
- (ii) A middle tier that receives data from different sources in XML format, combines them into a single XML document and adds the style sheet

- corresponding to the client view
- (iii) A thin client, which is like in the previous prototype, a web browser. Note that in the first prototype the Java applet also is loaded from the *Ground Station*, but then it connects directly to the data server on the train; this means in the first prototype we had a two-tier architecture.

As in the previous prototype, a *Ground Station* is used in order to access, via a wireless network, the remote *Train Gateway*. In this prototype there is no monitoring HTML page or Java applet. Instead there is a document server on the middle-tier, which is responsible for integrating data from different *Train Gateways* and for adding the corresponding style sheet to make the output readable by the client. In our case, the entire middle-tier is on the *Ground Station*.

This architecture is scalable, extensible and adaptable for future evolution, and it runs over firewalls. The main drawback is that it is not possible to use server side *push* technology.

During two years we run a demonstration of this prototype. In this demonstration the train data source was the same as in the demonstration of the previous prototype. The *Ground Station* was installed at ICA. The communication with the *Train Gateway* was done by means of an Internet wired connection.

5.6 Plug&Play

One of the most relevant issues of the RoMain system was the development of a fully *Plug&Play* system. We differentiate between two kinds of *Plug&Play*. We used the term *System Plug&Play* to refer to the functionality that allows that a system, in our case a DAS, configures itself and gets ready to run without human intervention. We used the term *Network Plug&Play* to refer to the functionality that allows that a system, in our case a DAS, intercommunicate spontaneously with other systems by offering some services and using services offered by other systems in a network community.

5.6.1 System Plug&Play

Maintenance staff does not want to spend time installing or configuring anything in order to monitor their systems. They just want to perform maintenance. Consequently, our idea was to have an automatic configuration, which enables the maintenance system to configure itself when it is attached to a new vehicle. To enable this automatic configuration, some “extra” configuration information for maintenance must be stored in the vehicle itself. We refer to this information as *maintenance information metadata*. This information will enable:

- The retrieval of the vehicle and equipments documentation from somewhere on the Internet.
- The automatic generation of the vehicle and equipments *proxies* on the *Train Gateway*. These *proxies* allow the *Train Gateway* to access the equipments through the specific fieldbus (e.g. TCN in our case) and update their web pages with the actual value of their properties.

Maintenance Information Metadata

The most flexible solution is to store, on the vehicle, the smallest dataset sufficient to identify the different components. It must contain enough information to automatically build component proxies. This dataset should be defined by equipment manufacturers and vehicle assemblers during the vehicle building process, and it is stored on a dedicated device, usually the train bus *node*, located in every vehicle. The *maintenance information metadata* structure is graphically shown in Figure 17 and described briefly as follows:

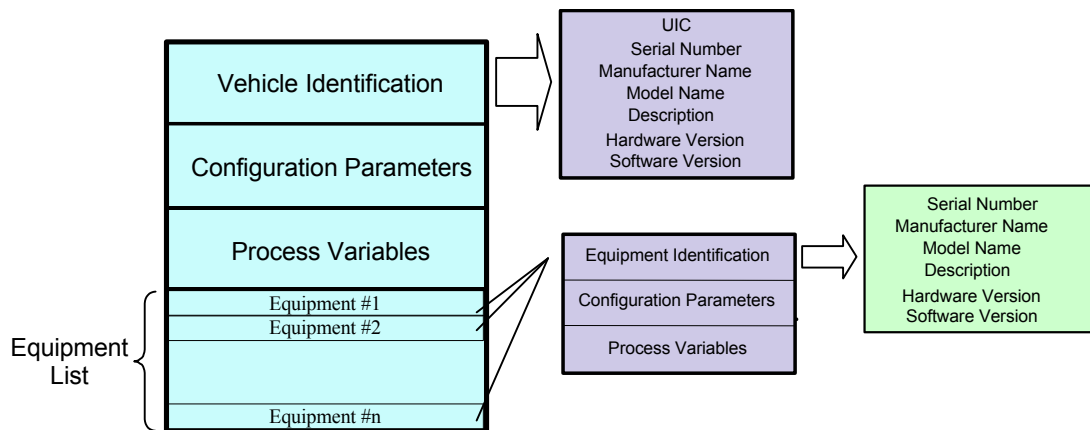


Figure 17 Maintenance Information Metadata Structure

- *Vehicle Identification*. Information about how to identify a vehicle (e.g. the last revision date). A vehicle is identified through:
 - *UIC Number*. Unique identifier for the vehicle, which is universally assigned by the Union Internationale des Chemins de Fer (UIC) international association.
 - *Serial Number*. Serial number of the vehicle. This field typically identifies a vehicle inside a manufacturer company.
 - *Manufacturer Name*. Name of the vehicle manufacturer.
 - *Model Name*. Name of the model of the vehicle.
 - *Description*. Textual field that describes the vehicle.
 - *Hardware Version*. Version of the hardware where the vehicle software is installed.
 - *Software Version*. Version of the software that implements the vehicle.
- *Configuration Parameter*. Specific information about the vehicle (e.g. the last revision date). A configuration parameter is described through:
 - *Name*. Name of the parameter.
 - *Type*. Type of the parameter.
 - *Description*. Textual field that describes the configuration parameter.
 - *Value*. Actual value with free format and length.
- *Process Variables*. The list of network variables exported by this vehicle, which corresponds to the internal application variables useful for maintenance. Each process variable is described with the following fields:

- *Name*. Logical name for the variable on the bus.
 - *Type*. Type of the process variable.
 - *Description*. Textual field that describes the process variable.
 - *Address*. Network address of a process variable.
- *Equipment List*. The list of equipments installed on the vehicle with, for each equipment, the following fields:
 - *Equipment Identification*. Similar to the *Vehicle Identification*. There is not a unique identifier for an equipment, thus we built it with a combination of *Manufacturer Name*, *Model Name*, *Hardware Version* and *Software Version* fields. This unique identifier is analogous to the *UIC Number* on a vehicle.
 - *Configuration Parameters*. The same as in a vehicle.
 - *Process Variables*. The same as in a vehicle.

System Initialization

When a *Train Gateway* is connected to a new vehicle the initialization process is started. After this process, the *Train Gateway* presents to the maintenance staff the vehicle and equipment web pages with actual values. The initialization of the *Train Gateway* is shown in Figure 18 and consists of the following steps:

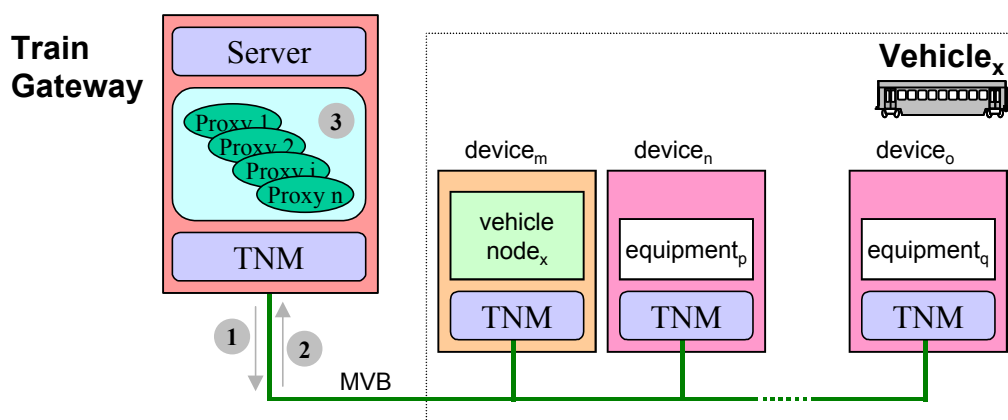


Figure 18 System Initialization

1. The *Train Gateway* asks the *vehicle node* for its *maintenance information metadata*. This is made through a Train Network Management (TNM) message called *Call_Read_Maintenance_Information*⁴.
2. The *vehicle node* replies to the *Train Gateway* with its *maintenance information metadata* using a Train Network Management (TNM) message called *Reply_Read_Maintenance_Information*⁴.
3. The *Train Gateway* generates a local component *proxy*, which is responsible for obtaining the actual state of the variables running on the real component.

⁴ *Call_Read_Maintenance_Information* and *Reply_Read_Maintenance_Information* have been formally specified, using a data representation and notation specified in [55] and proposed as an extension to the TNM standard [56].

5.6.2 Network Plug&Play

The administrator of the system is responsible for updating, removing or adding services on the *Train Gateway* and *Ground Station* machines. Due to the distributed nature of the system, this is a complex and tedious task, especially when a service must be installed on hundreds of trains. Another problem is that as trains are mobile systems, remote on-board services may be available only during certain periods of time, e.g., when a train enters a train station with a WLAN. It is difficult for a *Ground Station* to manage this dynamic behavior of services. In addition, the clients' perception of the quality of service of the application server is poor because *Ground Stations* may offer services that are not currently available.

Jini, a new paradigm for the development and management of distributed services promoted by Sun Microsystems, provides a possible solution for the efficient management of train services. The Jini technology [57-59] provides a simple infrastructure for providing services in a network. It enables the spontaneous interactions between applications. The result is a network of services connected together dynamically. Services can join or leave the network in a robust manner. Clients can rely upon the availability of visible services. The purpose of Jini is to federate groups of hardware and/or software components into a single, dynamic, distributed system. The resulting federation provides the simplicity of access and ease of administration. It guarantees the reliability and scalability of the whole system.

In order to be able to run Jini enabled services, it is necessary to introduce at least one machine, running a Jini Lookup Service (JLS), connected in the same LAN to the *Ground Station*. Eventually, we could also run the JLS on the same machine as the *Ground Station*. Additionally, some other JLSs all over the Internet, train stations and garages may be used. These JLSs would forward service subscriptions and other events to each other by tunneling. Tunneling is just an optimization mechanism that, for simplicity, is not described. The Jini-enabled system architecture is shown in Figure 19.

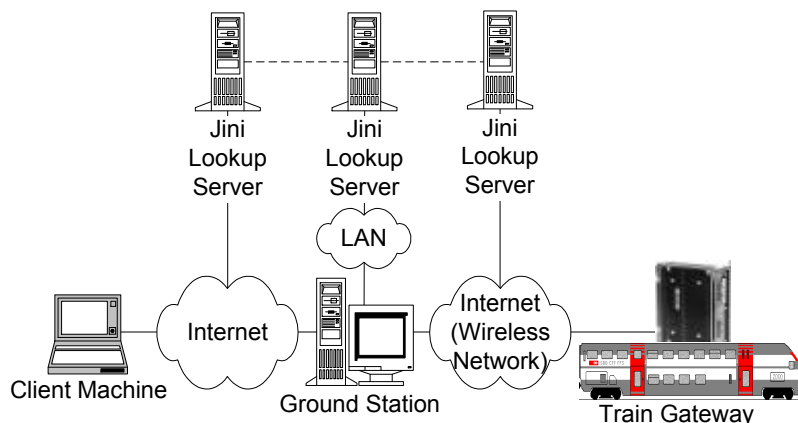


Figure 19 Jini-enabled System Architecture

In this system the JLS replaces the system administrator. The JLS allows us to automate the actions that were manually performed by a system administrator. The JLS allows on-board services to automatically register within the Jini community.

On-board services are lease based and they must renew periodically their lease in order not to be removed from the community. A *Ground Station* acts as a Jini client. It subscribes to receive notifications when on-board services appear or disappear from the community. The new Jini-enabled system is shown in Figure 20.

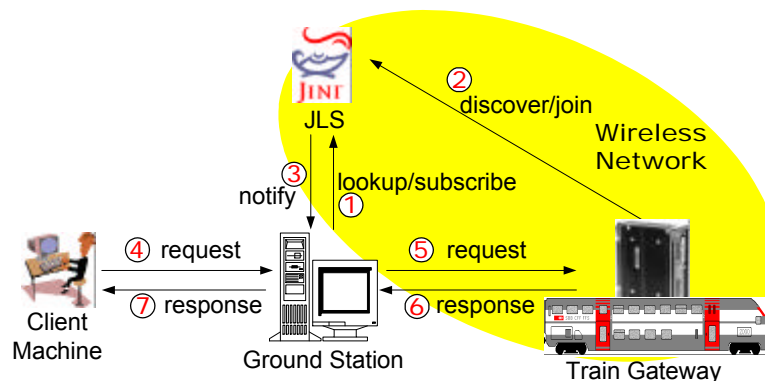


Figure 20 Jini-enabled System

The new sequence of actions is the following:

1. A *Ground Station* registers in the JLS for a particular remote event, namely an on-board DB service joining the federation. This registration enables the *Ground Station* to be notified when trains take part in the Jini community.
2. After startup, a Jini service on-board a train discovers and joins the JLS.
3. The *Ground Station* receives a notification in form of a remote event that contains a reference to the *proxy* object of the service.
4. Clients contact the *Ground Station* and request access to an on-board service on a particular train, which figures in the set of registered trains. This set of trains is up-to-date as trains that do not renew the lease of their joint operation are removed from the system.
5. The *Ground Station* searches for the corresponding *proxy*, which in turn contacts the on-board service on the associated train.
6. The on-board service processes the request and sends the response back to the *proxy* in the *Ground Station*.
7. The *Ground Station* sends the response back to the client.

The use of Jini technology simplifies the development of distributed systems because Jini forces distributed systems developers to deal with the network in early stages of development. Jini is not just a programming library to implement distributed systems, but a new paradigm for distributed system development. Using Jini, distributed systems developers can automate the, usually tedious, configuration process of such systems. Jini enables the searching for particular services based on complex attributes. Jini provides self-healing communities of services as it uses the concept of leases. Regardless of minor problems with the current implementation of Jini and the lack of standardization of services, Jini offers an efficient approach for developing distributed applications. However, Jini is not yet a mature technology. Even if the concepts are well defined, currently there are few Jini services available. Jini will have real success when service developers can dispose of a large number of standardized services all over the world. Before Jini becomes ubiquitous, it will be found in distributed applications where the application is a service provider and, at

the same time, the only user of these services.

5.7 Summary

In this chapter we described the RoMain system, which is a web-based monitoring tool for trains to support maintenance work that we developed. The main conclusion from the RoMain system is that cost-effective maintenance of globally distributed devices, which is an important issue in industrial applications, can be achieved by using Internet technology. Internet technology here means, the world wide TCP/IP network, protocols as HTTP and PPP, markup languages as HTML and XML, Internet browsers capable of downloading and running Java applets, web servers and application servers, etc. We also described two prototypes of the RoMain system. We compared these prototypes in order to evaluate their advantages and disadvantages. The results from this evaluation, described in Appendix A, demonstrated that when a high performance remote monitoring system is required, Java and Java RMI are the right technologies and that if flexibility on evolution is a strong requirement, a three-tier system, which is rather simple to develop using technologies such as HTTP and XML, may be a better choice. We used the development of the RoMain system as a case study to acquire the required knowledge about DASs. In one sense, the external specification of a generic architecture for DASs is a generalization of the model build for the RoMain system with some extensions and improvements.

6. External Specification of a Generic Architecture for Data Acquisition Systems

Parts of this work appeared in [10].

6.1 Introduction

In this chapter we describe an external specification of a generic architecture for DASs. This generic DAS specification enables the comparison of existing DAS products and standards. Additionally, it provides the DAS developers that aim to develop a specific DAS with a starting point for the design of a specific DAS. The generic DAS specification is composed of a conceptual model and a role-based use case model of a generic DAS. These models give DAS developers a high-level abstraction of DASs.

6.2 Data Acquisition Conceptual Model

In this section we present a conceptual model of a generic DAS. We show the main concepts that make up any DAS and their relationships. For a better understanding of the model, we group the concepts into five main packages. These packages and their inter-dependencies are shown in Figure 21.

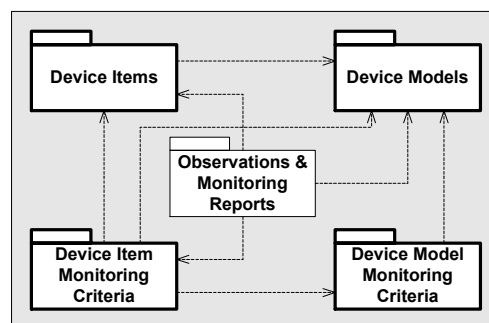


Figure 21 Data Acquisition Main Packages

- *Device Models*. This package groups all the concepts regarding device models. A device model represents a model that characterizes a set of device items.

- *Device Items*. This package groups all the concepts regarding device items. A device item represents a real world device that satisfies a device model.
- *Device Model Monitoring Criteria*. This package groups all the concepts regarding the definition of monitoring criteria, for generating monitoring reports, predefined in a device model and common for all the corresponding device items.
- *Device Item Monitoring Criteria*. This package groups all the concepts that allow for the definition of monitoring criteria, for generating monitoring reports, specific for a device item.
- *Observations & Reports*. This package groups all the concepts regarding observations and reports taken on a system. Observations are classified as quantitative (*measurements*) or qualitative (*category observations*), according to the *Observations and Measurements* analysis pattern described by Fowler in [40]. Monitoring reports are classified as reports that record a change in the composition of the system (*composition reports*), reports that record a snapshot of the system at a specific time (*status reports*), and reports that indicate a certain state of the system (*event reports*).

In the following sub-sections we describe each of these packages. In the models, we distinguish between *operational* and *knowledge-level* concepts. We adopt this idea from Fowler [40]. At the *operational-level* the model records the day-to-day events of the domain, whereas at the *knowledge-level* the model records the general rules that govern this structure. We represent *knowledge-level* concepts by using a box with a thick border, and a box with a thin border represents *operational-level* concepts.

6.2.1 Device Models

The *Device Models* model is shown in Figure 22. A device model is an instance of *Device Model* that represents a model, created in the design process, that characterizes a set of *real world devices*. A device model can be composed of many device models, each of them implementing a specific *function* on the parent device model. We used our own variation of the *Composite* pattern, named *Model Composite* (further discussed in chapter 7), to manage the composition of device models. A device model defines many measurement points. An instance of *Measurement Point* defines a measurement point associated with a phenomenon type and a measurement type. An instance of *Phenomenon Type* represents something that can be quantitatively (e.g. temperature), or qualitatively (e.g. door_status), observed. A phenomenon type defines the units on which observations of this phenomenon type are expressed. We adopt the convention of using standard SI (metric) units, as proposed in [1], for phenomenon types. A phenomenon type also specifies the range within which a value is qualified as “normal”. Eventually, a phenomenon type records the set of potential qualitative values that a measurement of such a phenomenon type can take (e.g. temperature_low, temperature_medium, and temperature_high). Each of these values is an instance of *Phenomenon*. *Phenomenon* records the range of quantitative observations of a phenomenon type that corresponds to a qualitative observation. This enables the automatic recording of an occurrence of this phenomenon upon a quantitative

observation, of the corresponding phenomenon type, with a value within the range of the phenomenon. An instance of *Measurement Type* is associated with a measurement point to give some semantic information about the measurements taken at this measurement point.

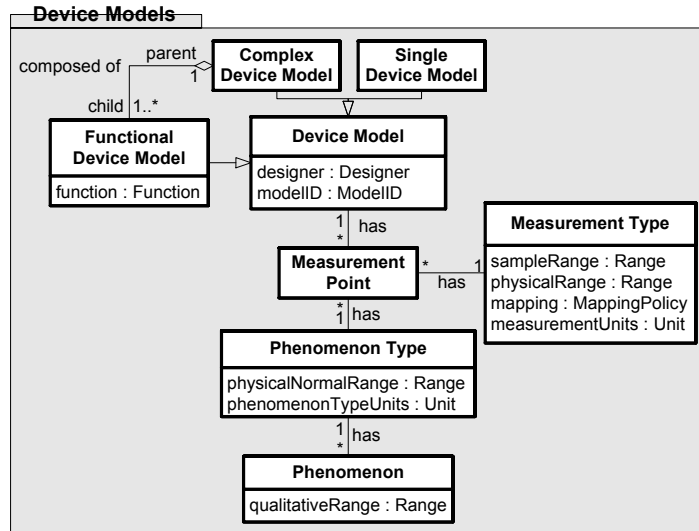


Figure 22 Device Models

6.2.2 Device Items

The *Device Items* model is shown in Figure 23. A device item is an instance of *Device Item* that represents a real world device created in the manufacturing process. A device item is characterized by a device model. As specified by its device model, a device item can be composed of many device items, which are also characterized by the associated device models. We organize device items using the *Composite* pattern. The association class *Device Address* allows us to record the address within a complex device item where a child device item, characterized by an instance of *Functional Device Model*, is installed. Each device item defines many measurement addresses. An instance of *Measurement Address* defines the actual location in a device item associated with a measurement point, where observations of a phenomenon type are taken.

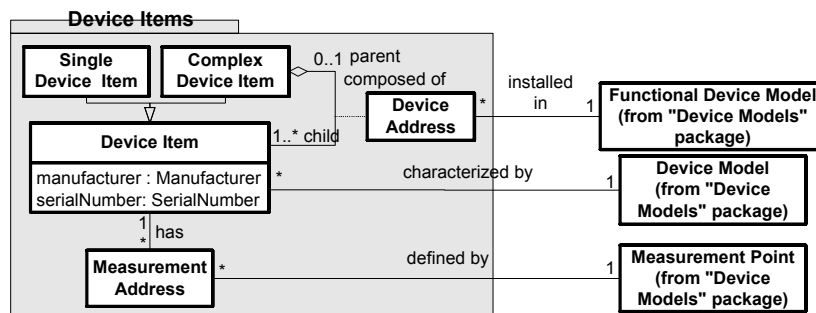


Figure 23 Device Items

6.2.3 Device Model Monitoring Criteria

The *Device Model Monitoring Criteria* model is shown in Figure 24. The

ability to define reports, with a consistent status, of a set of data is one of the requirements of any DAS. The OMG's DAIS RFP [7] refers to this as *Dataset*, and OPC [4] as *OPC Group*. Our approach for defining monitoring criteria for recording monitoring reports is inspired by *Mansouri-Samani and Sloman [72]*. In the design process of a device model, a designer can define certain monitoring criteria specific to this device model and common to all the device items characterized by this device model. *Device Model Trigger Condition* enables the definition of conditions in order to automatically trigger the recording of monitoring reports at a specific time or when the system is in a certain state. We distinguish three kinds of *Device Model Monitoring Criteria*: *Device Model Composition Monitoring Criteria* enables the recording of changes on the composition of the system. A device model composition monitoring criteria is associated with a set of device models; *Device Model Status Monitoring Criteria* enables the recording of a snapshot of the system at a specific time. Status monitoring criteria are associated with a set of measurement points; and *Device Model Event Monitoring Criteria* enables the recording of a certain state of the system.

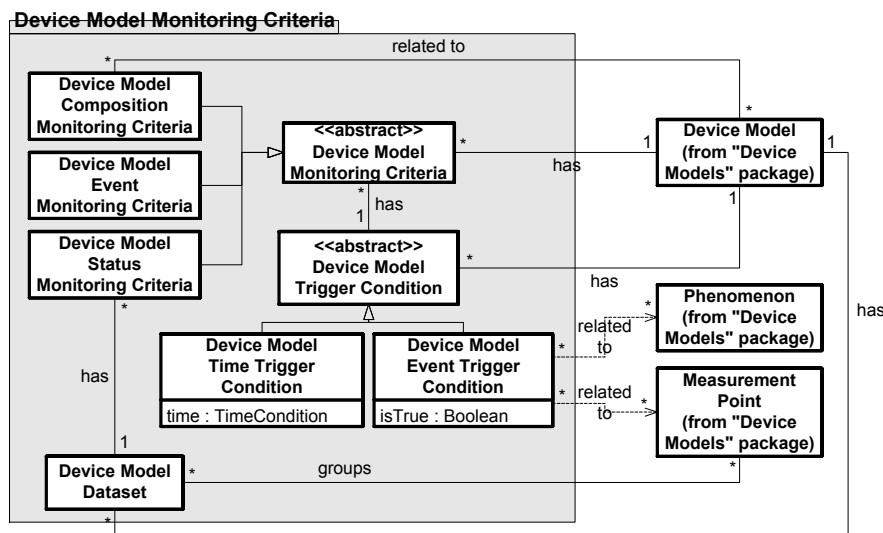


Figure 24 Device Model Monitoring Criteria

6.2.4 Device Item Monitoring Criteria

The *Device Item Monitoring Criteria* model is shown in Figure 25. In the installation and maintenance phases administrators administer datasets, trigger conditions and monitoring criteria for device items. The DAS system will record monitoring reports corresponding to device item monitoring criteria. Device item datasets, trigger conditions and monitoring criteria can be *predefined*, meaning that there are already defined in the corresponding device model, or *custom*, meaning that there are defined in the device item. A custom monitoring criteria may use any combination of custom or predefined datasets and trigger conditions. Device item monitoring criteria can be *public*, meaning that any supervisor of the system can access monitoring reports of such monitoring criteria, or *private*, meaning that only the creator of the monitoring criteria is allowed to access monitoring reports corresponding to such monitoring criteria.

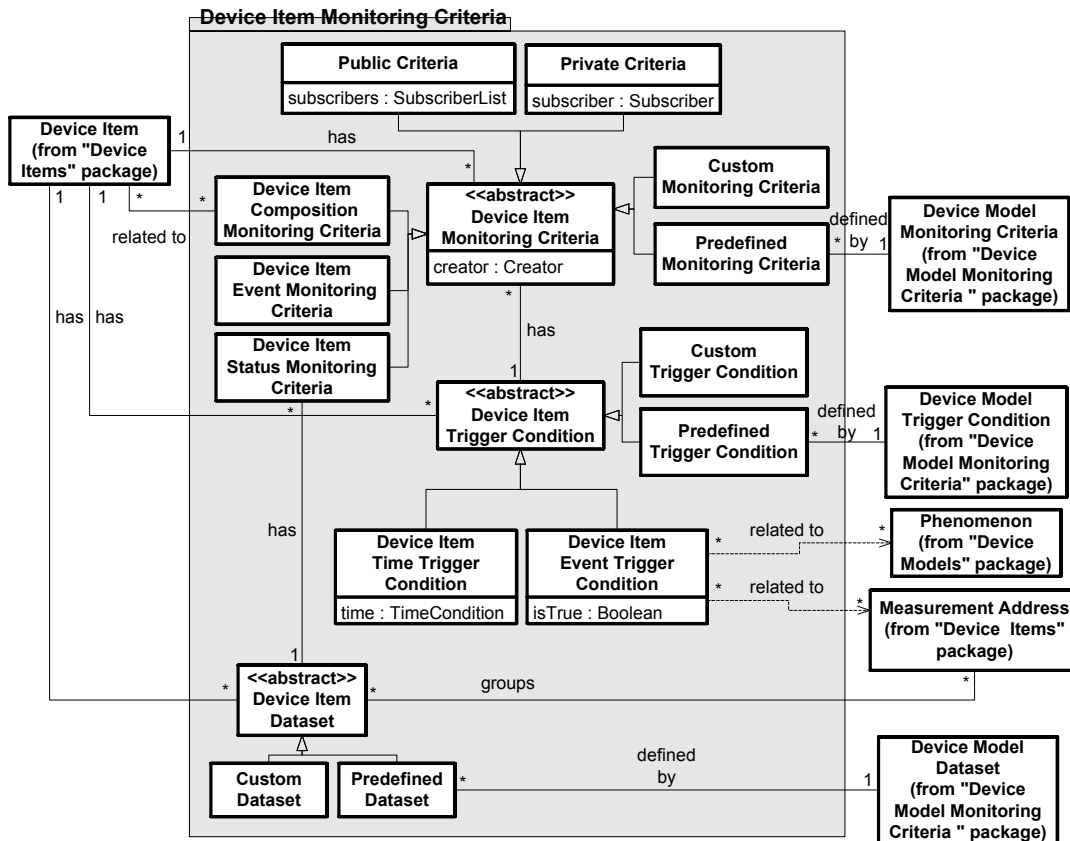


Figure 25 Device Item Monitoring Criteria

6.2.5 Observations & Monitoring Reports

The *Observation and Monitoring Reports* model is shown in Figure 26. In this package we present concepts that allow us to record observations and monitoring reports taken on a device item. Our observation model is inspired by the *Observations and Measurements* analysis pattern described by Fowler in [40]. In a measurement address we can record many observations with different timestamps. *Observation* is an abstract concept that represents both quantitative and qualitative observations. An observation records a timestamp corresponding to the time an observation was taken. *Measurement* represents quantitative observations. A measurement records the physical value corresponding to the measurement, which is represented by the *value* attribute. A measurement is associated with a phenomenon type, and a phenomenon type can have many measurements. A *Category Observation* represents a qualitative observation. A category observation is associated with a phenomenon, and a phenomenon can have many category observations. Sometimes recording that a phenomenon is absent is as important as recording its presence. The *isPresent* Boolean attribute of category observation is added to enable recording the absence or presence of a phenomenon. *Monitoring Report* enables the recording of an occurrence of fulfilled monitoring criteria. A monitoring report is always associated with monitoring criteria of a device item. A monitoring report is also associated with the observations that generate the monitoring report.

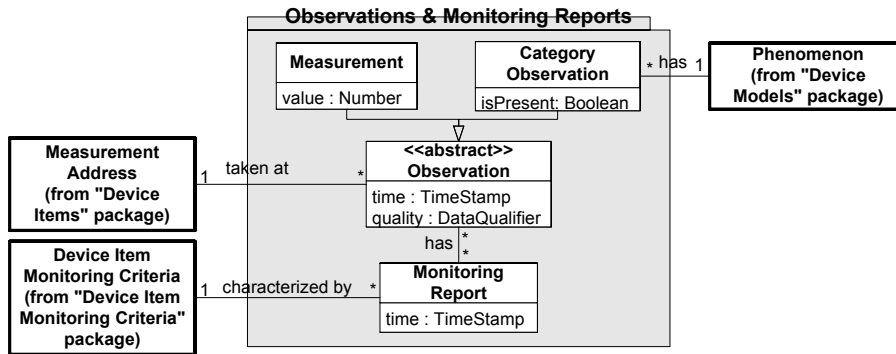


Figure 26 Observations & Monitoring Reports

6.2.6 Detailed Concepts

In this section we explain some detailed concepts of the conceptual model. These concepts are: *Measurement Type*, *Mapping Policy*, *Time Condition*, *Device Model Event Trigger Condition*, *Device Item Event Trigger Condition*, *Timestamp* and *Data Qualifier*.

Measurement Type

The *Measurement Type* model is shown in Figure 27. A measurement type defines the permissible ranges of sampled and physical values, a mapping policy between sampled and physical values, and the units of the measurement. This information could have been embedded in a phenomenon type, but measurement type allows us to reuse the same information for several phenomenon types. The measurement type information depends on how measurements are actually measured in a measurement point. Consequently it may happen that the measurement units are not the same as the phenomenon type units, being then necessary to transform the physical value in measurement type units to the physical value in phenomenon type units, which will be recorded in the system.

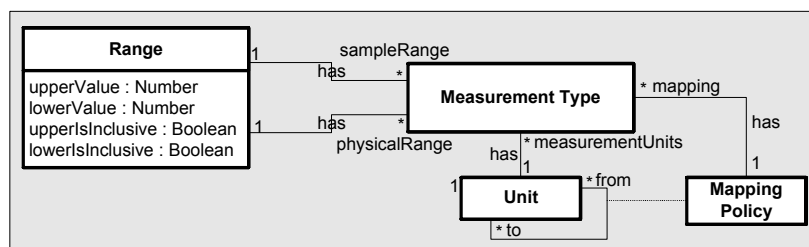


Figure 27 Measurement Type

Mapping Policy

The *Mapping Policy* model is shown in Figure 28. An instance of *Mapping Policy* defines the conversion between two numerical values. *Linear Mapping Policy* represents a mapping policy with a linear function ($y=Ax+B$); where x corresponds to the original value and y to the calculated value. *Function Mapping Policy* represents a mapping policy with a more complex function ($y=f(x)$). *Function Mapping Policy* seems to be a good scenario for applying mobile code [73]. In this way device designers could easily upload the code corresponding to this function.

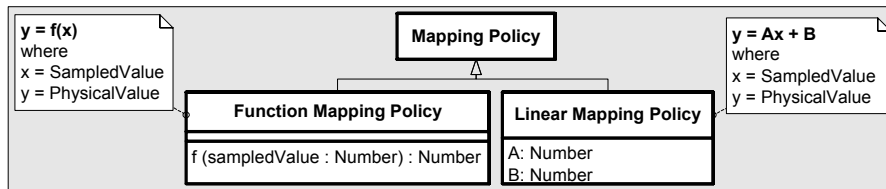


Figure 28 Mapping Policy

Time Condition

The *Time Condition* model is shown in Figure 29. *Time Condition* enables the definition of periodical or scheduled time conditions to record monitoring reports. *Period* enables the definition of a time condition as a period of time in milliseconds. *Schedule* enables the definition of a schedule when a monitoring report will be recorded. A *Schedule* is composed of a *Recurrence Time*, a *Recurrence Pattern* and a *Recurrence Range*. *Recurrence Time* enables the definition of a 24-hour time when a monitoring report will be recorded. *Recurrence Pattern* enables the definition of many time patterns to record recurrently monitoring reports: *Daily*, *Weekly*, *Monthly* and *Yearly* allow us to define monitoring criteria to record monitoring reports every certain number of days, every certain number of weeks on some specific days of a week, every certain number of months on a specific day of a month, and every certain number of years on a specific day of a month, respectively. Finally, *Recurrence Range* enables the definition of a *Begin Time* (just a *Begin Date*) to start recording monitoring reports and an *End Time* to stop recording monitoring reports. *End Time* enables the definition of an end time by a number of occurrences, by a specific end date or with no end (meaning that monitoring reports will be generated “forever”).

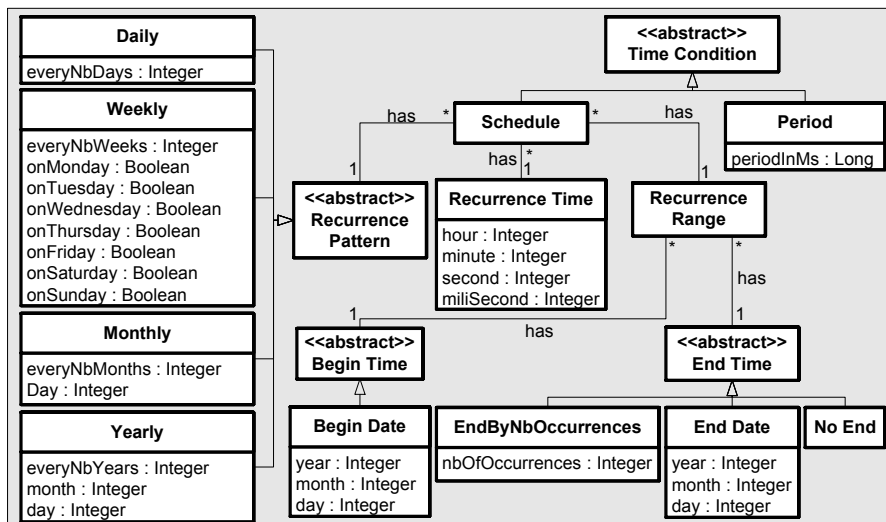


Figure 29 Time Trigger Condition

Device Model Event Trigger Condition

The *Device Model Event Trigger Condition* model is shown in Figure 30. The *Device Model Event Trigger Condition* model enables the definition of conditions, common to all the device items characterized by a device model, to trigger an event. In order to explain device model event trigger conditions, we make use of a Boolean

algebraic notation⁵. *Device Model Event Condition* enables the recording of $X=A$ and $X=A'$; where A means that a certain condition has been satisfied. There are two kinds of *Device Model Event Conditions*. *Device Model Boolean Event Condition* enables the recording of a condition that is satisfied when a certain phenomenon has been observed in a measurement point. *Device Model Function Event Condition* enables the recording of a condition that is satisfied when the result of applying a certain function in a measurement point returns *true*. *Device Model Function Event Condition* seems to be a good scenario for applying mobile code [73]. In this way device designers and/or administrators could easily upload the actual code corresponding to the function to be satisfied. *Device Model Event Condition Set* enables the recording of conditions such as $X=A.B$ and $X=(A.B)'$; where A, B are *Device Model Event Conditions*. *Device Model Event Trigger Condition* allows us to record trigger conditions such as $X=A+B$ and $X=(A+B)'$; where A, B are *Device Model Event Condition Sets*. This enables the recording of any device model event trigger condition, because any device model event trigger condition can be expressed by means of an algebraic combination of device mode event conditions with the AND logical operator and an algebraic combination of device model event condition sets with the OR logical operator. A transformation of any algebraic expression into these terms is possible by applying one of *De Morgan's laws*⁶.

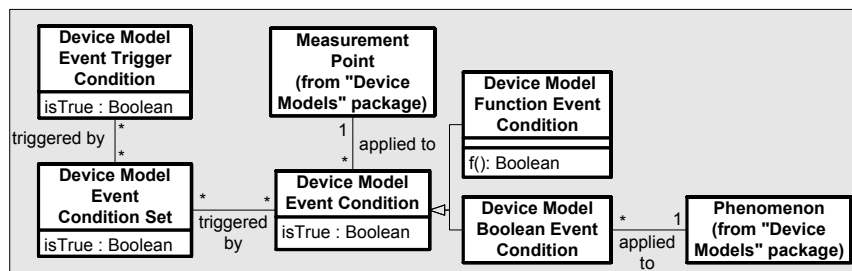


Figure 30 Device Model Event Trigger Condition

Device Item Event Trigger Condition

The *Device Item Event Trigger Condition* model is shown in Figure 31. The *Device Item Event Trigger Condition* model enables the definition of conditions, specific to a device item, to trigger an event. The reasoning is analogous to the device model event trigger condition, but the difference is that the condition is applied to a specific measurement address of a device item rather than to a measurement point of a device model.

⁵ “ . ” corresponds to the AND logical operator; “ + ” corresponds to the OR logical operator; and “ ’ ” corresponds to the NOT logical operator

⁶ The two laws, known as *De Morgan's*, are: $(A+B)' = A' . B'$; and $(A.B)' = A' + B'$

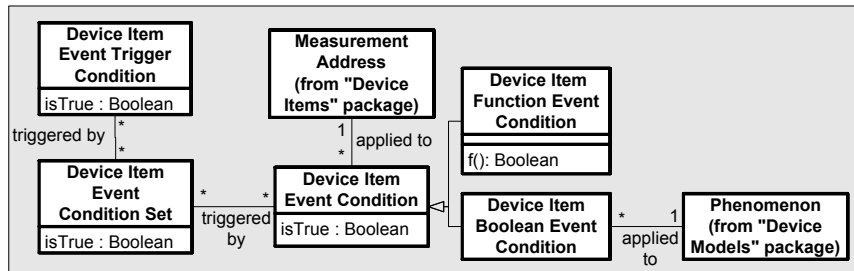


Figure 31 Device Item Event Trigger Condition

Timestamps

Recording the time an observation is taken is a key issue for enabling a subsequent analysis of observations. In order to avoid anomalies due to inconsistent time formats (e.g., because of different time zones), we adopted the convention of storing all timestamps using Universal Coordinated Time (UTC) format. This, further discussed by *Olken et al.* [1], is a common practice in DASs.

Data Qualifiers

In DASs it is also a common practice to include a *data qualifier* (see Figure 32) with an observation. According to [7] a *Data Qualifier* includes information about the *Validity* (valid, held from a previous value, suspect, not_valid or substituted manually), the *Current Source* (metered, calculated, entered, or estimated) and the *Normal Value* (normal or abnormal) of an observation.

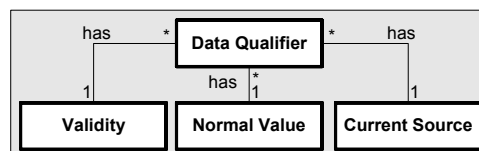


Figure 32 Data Qualifiers

6.2.7 Complete DAS Conceptual Model

The complete conceptual model of a generic DAS is shown in Figure 33. To simplify the model we only show the main attributes of a concept and some concepts have been intentionally designed as attributes of higher-level concepts.

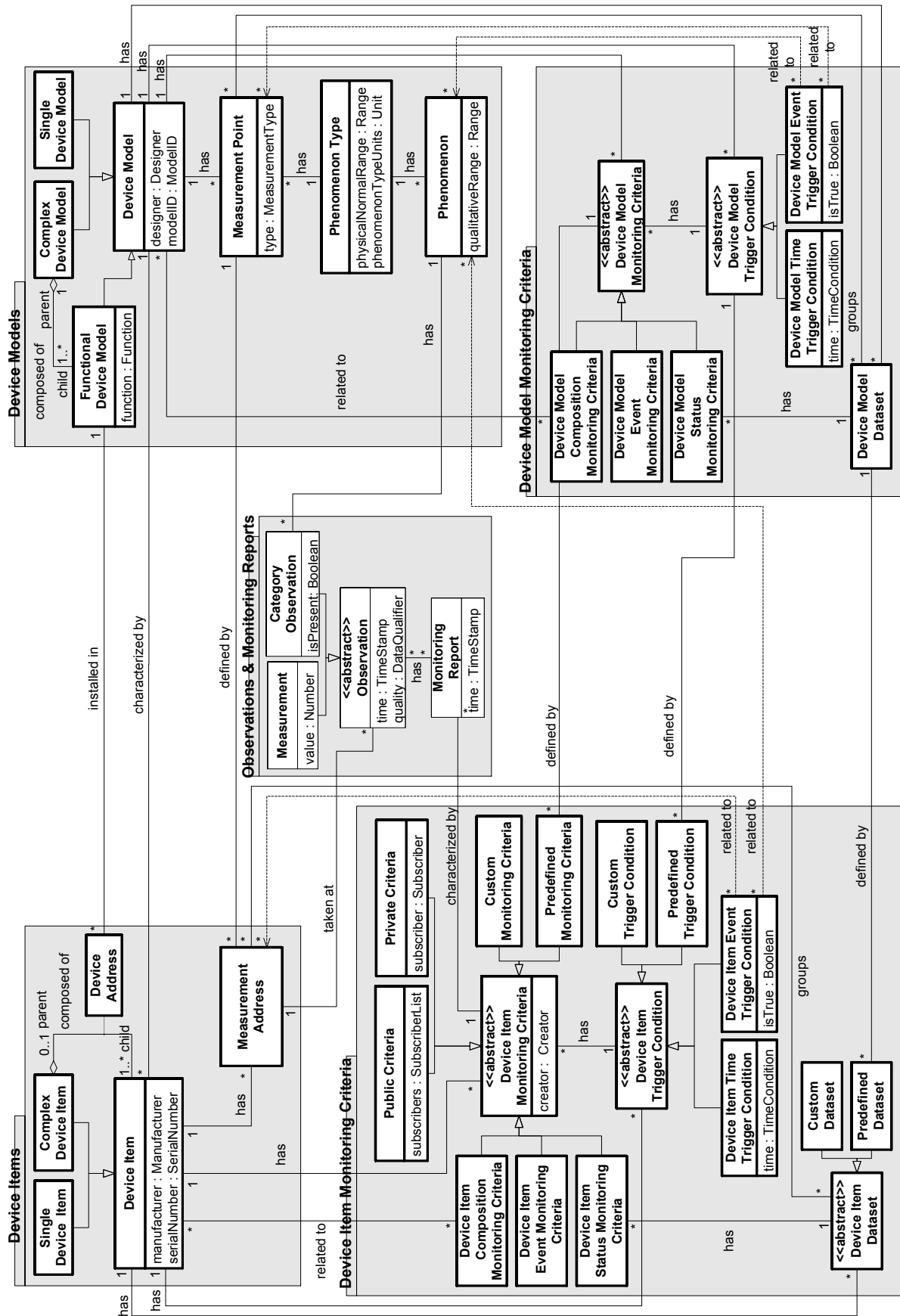


Figure 33 Complete DAS Conceptual Model

6.3 Data Acquisition Role-based Use Case Model

In this section we present a role-based use case model for a generic DAS. This role-based use case model specifies the common functionalities of any DAS. To give a context to the data acquisition process we begin by positioning this process as a part of the device lifecycle. Then, we describe in detail all the use cases corresponding to the acquisition of data. Finally, we refine all of these use cases using role-based use case modeling. The role-based use cases are specified in Appendix B. The elementary roles are specified in Appendix C.

In Figure 34, we show the main use cases corresponding to the device lifecycle. We use an activity diagram to specify the sequence in which these use cases are carried out.

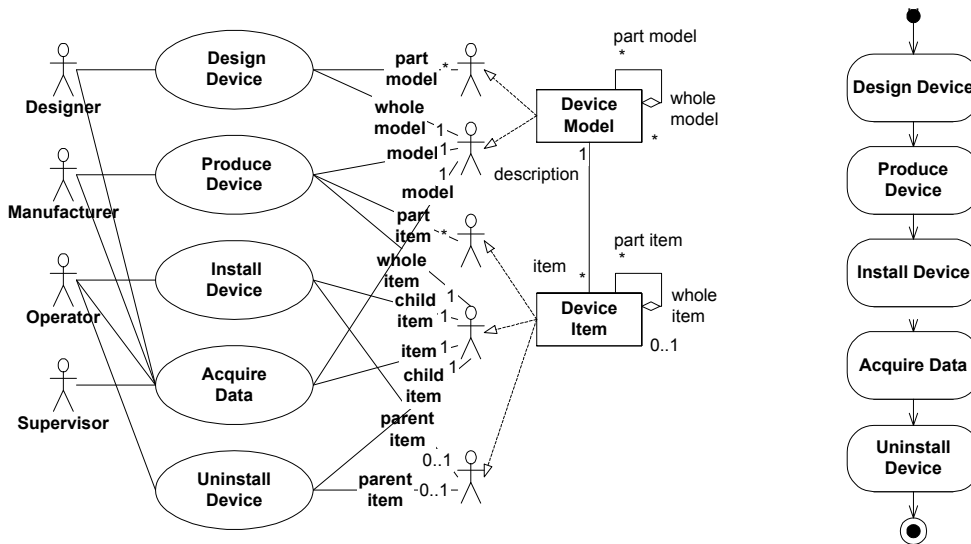


Figure 34 Device Lifecycle

In a device lifecycle, a *Designer* designs a *Whole Model*, which is an instance of *Device Model*. *Whole Model* can be composed of zero to many *Part Models*, which are also instances of *Device Models*, forming a *part-whole* hierarchy of instances of *Device Models*. A *Manufacturer* produces a *Whole Item*, which is an instance of *Device Item* that satisfies a device model, which is an instance of *Device Model*. A *Whole Item* can be composed of zero to many *Parts Items*, which are also instances of *Device Item*, forming a *part-whole* hierarchy of instances of *Device Items*. An operator installs a *Child Item*, which is an instance of *Device Item*, into a *Parent Item*, which is another instance of *Device Item*, forming a *child-parent* hierarchy. In fact, this hierarchy is analogous to the *part-whole* hierarchy of instances of *Device Item* formed during the production of an instance of *Device Item*. A *child* is to its *parent* the same as a *part* to its *whole*. The only difference is the nature of the link between each other: in the case of *part-whole* hierarchy the link is established during the creation of an instance, whereas in the case of *child-parent* hierarchy the link is established during the installation of an instance. A *Supervisor* acquires data from an instance of *Device Item*. Operational data is acquired from a device item. However,

knowledge-level data and some other information can be distributed and provided by *Designers*, *Manufacturers*, and/or *Operators*. An operator uninstalls *Child Item*, which is an instance of *Device Item* from its *Parent Item*, which is another instance of *Device Item*.

In this thesis we focus on the use case corresponding to the acquisition of data (*Acquire Data*). In Figure 35, we show the refined use cases corresponding to the acquisition of data. We mapped each of the functional requirement defined by the OMG in the DAIS RFP [7] into a use case. We have two additional actors: the *DAS*, which represents the system (the representation of the system in the use cases is further discussed in Chapter 7); and the *Administrator*, which takes care of the administration of the system.

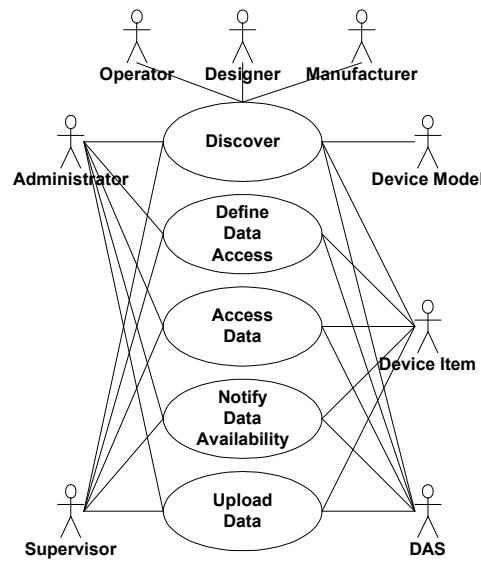


Figure 35 Data Acquisition Use Cases

We use an activity diagram, shown in Figure 36, to specify the sequence in which these use cases are carried out:

- (a) Discover device items, device models, their composition and related information until all the information has been discovered. Then, go to b), to define data access; to c), to access data; to d), to notify availability of new data; or to e), to upload new data.
- (b) Define data access for the device items until all information has been defined. Then, go to a), to discover new data; to c), to access data; to d), to notify availability of new data; or to e), to upload new data.
- (c) Access data until all the information to be accessed has been accessed. Then, go to b), to define data access; to d), to notify availability of new data; or to e), to upload new data.
- (d) Notify availability of new data until all the notifications have been notified. Then, go to b), to define data access; to c), to access data; or to e), to upload new data.
- (e) Upload new data until all the data has been uploaded. Then, go to b), to define data access; to c), to access data; or to d), to notify availability of new data.

These uses cases can be carried out many times, therefore there is not an explicit end point.

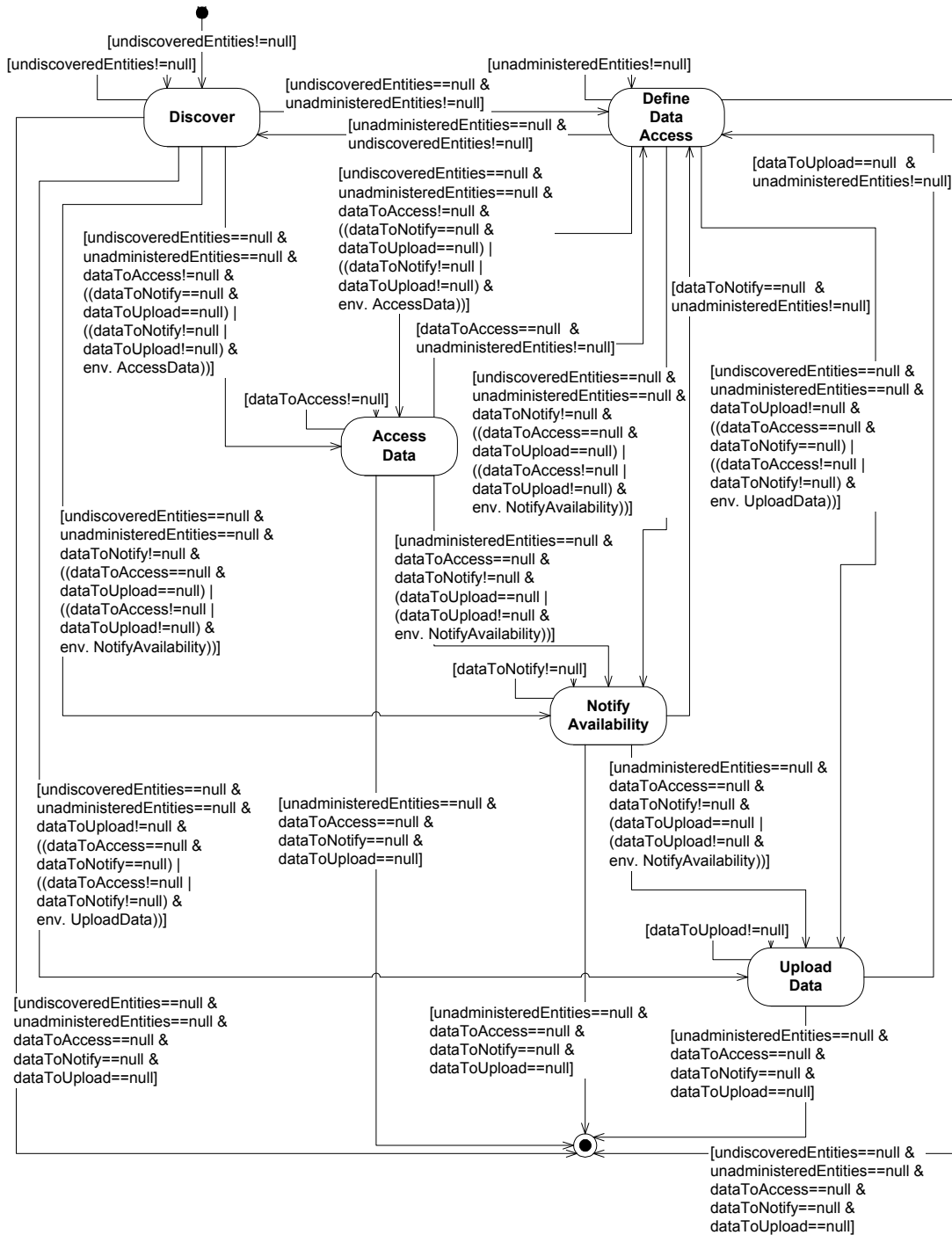


Figure 36 Data Acquisition Activity Diagram

In the following sections we describe in detail each of the use cases corresponding to the data acquisition process. We also refine all of these use cases using role-based use case modeling.

6.3.1 Discover

This use case implements the “discovery of remote system and device schema” functional requirement, which is defined by the OMG in DAIS RFP [7] as:

“Mechanisms for discovering accessible remote devices, measurements, discrete/incremental information, permissible ranges and/or sets of values, alarms and industrial system sourced events. ...A means shall be provided for a client system to determine the data types and quantities (i.e. cardinality) of data elements available from a particular entity within an industrial system, as well as the identifiers and some of the semantics associated with those data elements.”

Discovering allows supervisors to obtain, by request, the current composition of an instance of *Device Item*, the different kinds of data values that can be acquired from this instance and their types and semantics. During the discovering process supervisors discover *knowledge-level* information such as the composition of instances of *Device Items* installed on the systems, the instances of *Device Models* associated with them, the phenomenon types and measurement types of these instances of *Device Model*, and so on.

We refined the *Discover* use case into the following role-based use cases, shown in Figure 37:

- (i) *Discover Item*. An *Item Information Requester* obtains, by request, the information specific to a device item (e.g. its manufacturer or serial number).
- (ii) *Discover Item Composition*. An *Item Information Requester* obtains, by request, the current composition of an instance of *Device Item*.
- (iii) *Discover Model*. A *Model Information Requester* obtains, by request, the device model of a device item. Together with the device model some *knowledge-level* information such as phenomenon types, measurement types, and so on, may be discovered.
- (iv) *Discover Model Composition*. A *Model Information Requester* obtains, by request, the composition of an instance of *Device Model*.
- (v) *Discover Datasets*. A *Dataset Information Requester* obtains, by request, the datasets defined on device item.
- (vi) *Discover Trigger Conditions*. A *Trigger Condition Information Requester* obtains, by request, the trigger conditions defined on a device item.
- (vii) *Discover Monitoring Criteria*. A *Monitoring Criteria Information Requester* obtains, by request, the monitoring criteria defined on a device item. A *Monitoring Criteria Information Requester* not being the creator of such monitoring criteria can discover only monitoring criteria defined as *public*. Monitoring criteria predefined on the corresponding device model are considered *public* and therefore they are accessible for any *Monitoring Criteria Information Requester*.

In these use cases we introduced, for the first time, the *DAS Broker* role. This role allows us to decouple the rest of the roles allowing us to remain independent of design choices. All the messages between roles pass through the *DAS Broker*. Typically, the same actor (e.g., the *Supervisor* of the system) will implement all the roles X *Information Requester*, being X any entity that can be discovered

(device item, device item composition, device model, device model composition, datasets, trigger conditions and monitoring criteria). But, there may be systems where only certain users with special privileges (e.g., the *Administrator* of the system) are allowed to discover certain entities (e.g. monitoring criteria). The use of different elementary roles for any entity that can be discovered makes our specification more independent from decisions that should be taken later, in the design phase of a particular DAS.

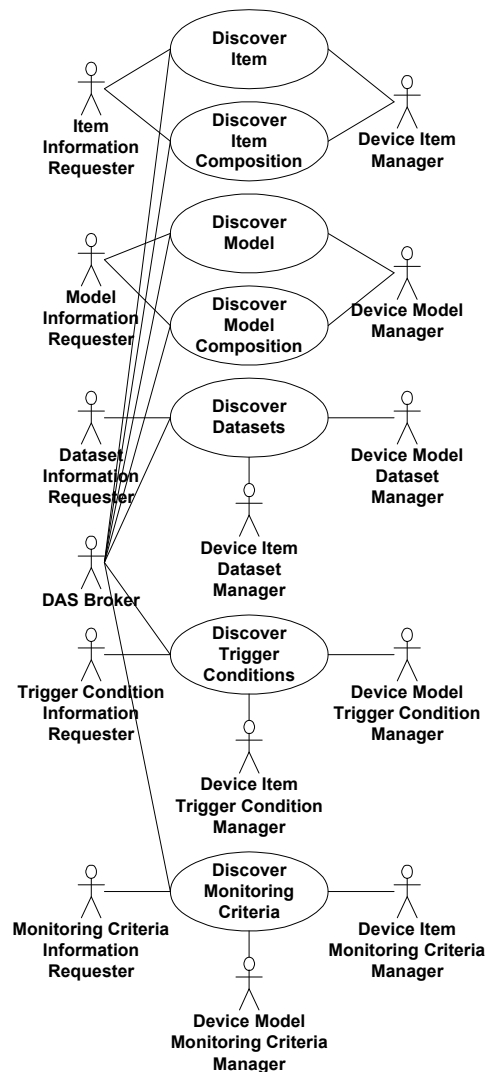


Figure 37 Discover Use Case

We introduced a manager role, *X Manager*, for each entity *X* that can be discovered, as we consider that we have to offer DAS developers the choice of implementing the management of all these entities independently. We wanted to make explicit the discovery of device item composition as we consider that this is a sufficiently different use case from the use case corresponding to the discovery of device item information. However, we did not consider necessary to use different roles for the roles that discover or manage the device items and their composition, as this information is always discovered or managed by the same entities. The same thinking applies to device models and their composition.

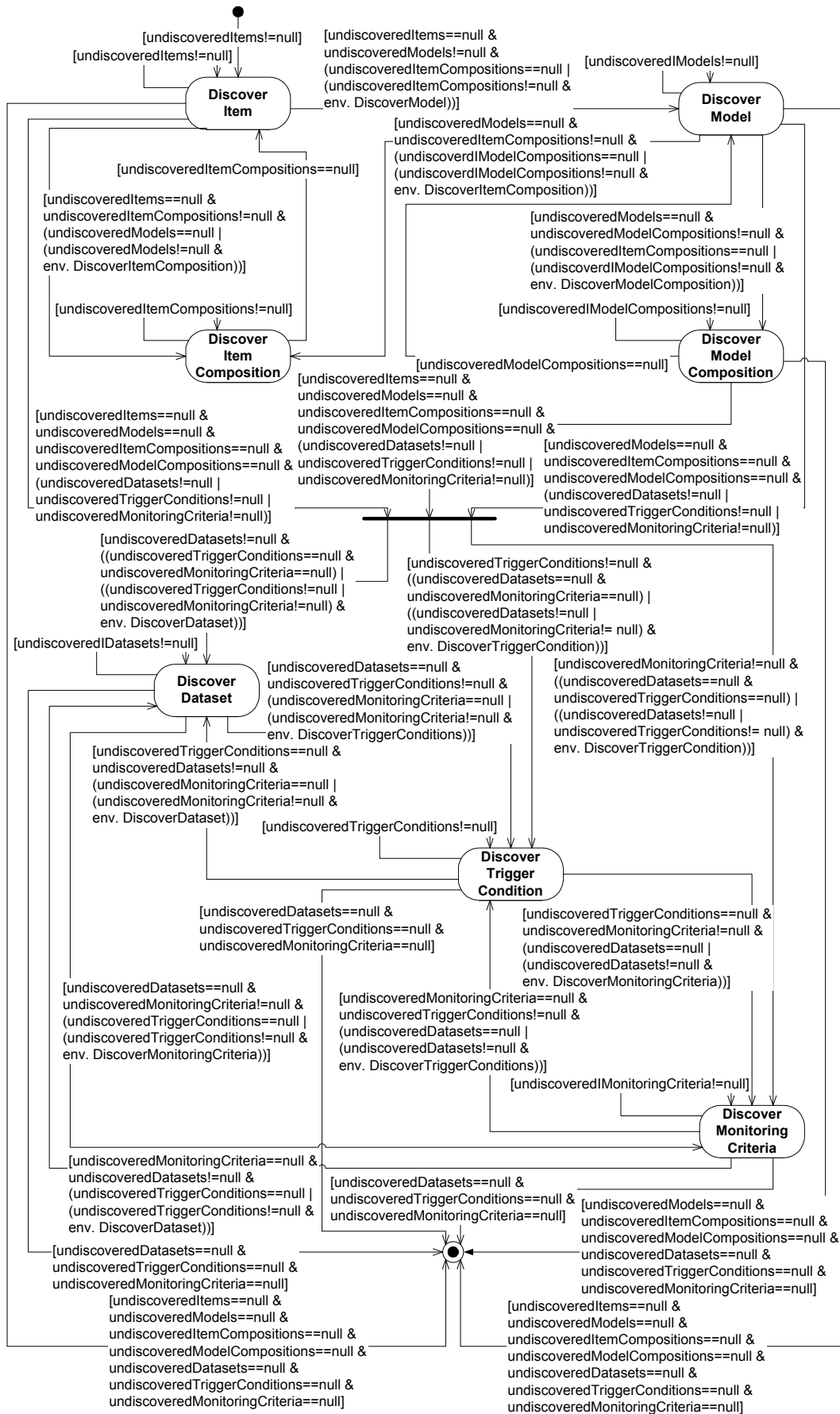


Figure 38 Discover Activity Diagram

We use an activity diagram, shown in Figure 38, to specify the sequence in which these use cases may be carried out:

- (a) Discover device items until all device items have been discovered. Then, go to b), to discover device models; to c), to discover device item composition; to e), f), g) to discover datasets, trigger conditions or monitoring criteria respectively; or to the end, if all the information has been discovered.
- (b) Discover device models until all device models have been discovered. Then, go to c), to discover device item composition; to d), to discover device model composition; to e), f), g) to discover datasets, trigger conditions or monitoring criteria respectively; or to the end, if all the information has been discovered.
- (c) Discover device item composition until all device item composition has been discovered. Then, go to a), to discover device items.
- (d) Discover device model composition until all device model composition has been discovered. Then, go to b), to discover device models; to c), to discover device item composition; to e), f), g) to discover datasets, trigger conditions or monitoring criteria respectively; or to the end, if all the information has been discovered.
- (e) Discover datasets until all datasets have been discovered. Then, go to f) or g) to discover trigger conditions or monitoring criteria respectively; or to the end, if all the information has been discovered.
- (f) Discover trigger conditions until all trigger conditions have been discovered. Then go to e) or g) to discover datasets or monitoring criteria respectively; or to the end, if all the information has been discovered.
- (g) Discover monitoring criteria until all monitoring criteria has been discovered. Then go to e) or f) to discover datasets or trigger conditions respectively; or to the end, if all the information has been discovered.

6.3.2 Define Data Access

This use case implements the “defining data access request” functional requirement, which is defined by the OMG in DAIS RFP [7] as:

“Mechanisms for defining (and deleting) a set of data and how the set of data should be retrieved. Data sets are collections of data, defined by the client, by a third party, or pre-existing data on the device, that are transferred in response to an event or single read request. The request for data retrieval can be triggered on-demand, or based on time, exception and/or event. A client could register to receive event notifications for the availability of the data requested.”

Defining data access allows supervisors to define monitoring criteria of a device item. There are three kind of device item monitoring criteria: composition, event and status monitoring criteria. A composition monitoring criteria enables the definition of interest on the change on the composition of a set of device items. Composition monitoring criteria make it possible to implement a *Plug&Play* functionality. A status monitoring criteria enables the specification of snapshots of the system to be taken at a specific time or upon the occurrence of an event. A status monitoring criteria is always associated with a dataset, which represents the set of

measurement points where to take the observations. The values of a dataset must be collected and sent at the same time to ensure consistency of data. A status monitoring criteria is typically associated with a time trigger condition, which represents the condition when the observations must be taken. Eventually, a status monitoring criteria can be associated with an event trigger condition instead of a time condition. In this case the observations will be taken upon the occurrence of an event. An event monitoring criteria enables the recording of the occurrence of an event. An event monitoring criteria is always associated with an event trigger condition.

There are two ways a supervisor can retrieve data: based on the *pull* model or based on the *push* model. The *pull* model is based on the *request/response* paradigm; a client sends a request to the server, then the server answers. This is functionally equivalent to the client *pulling* the data off the server. The *push* model is based on the *publish/subscribe/distribute* paradigm; a client subscribes for receiving updates of data from a server, later the server takes the initiative to *push* the data to the client. *Martin-Flatin* discusses in detail these two paradigms, applied to web-based management, in [74]. Defining data access must allow a supervisor to define data access requests based on both models.

We refined the *Define Data Access* use case into the following role-based use cases, shown in Figure 39:

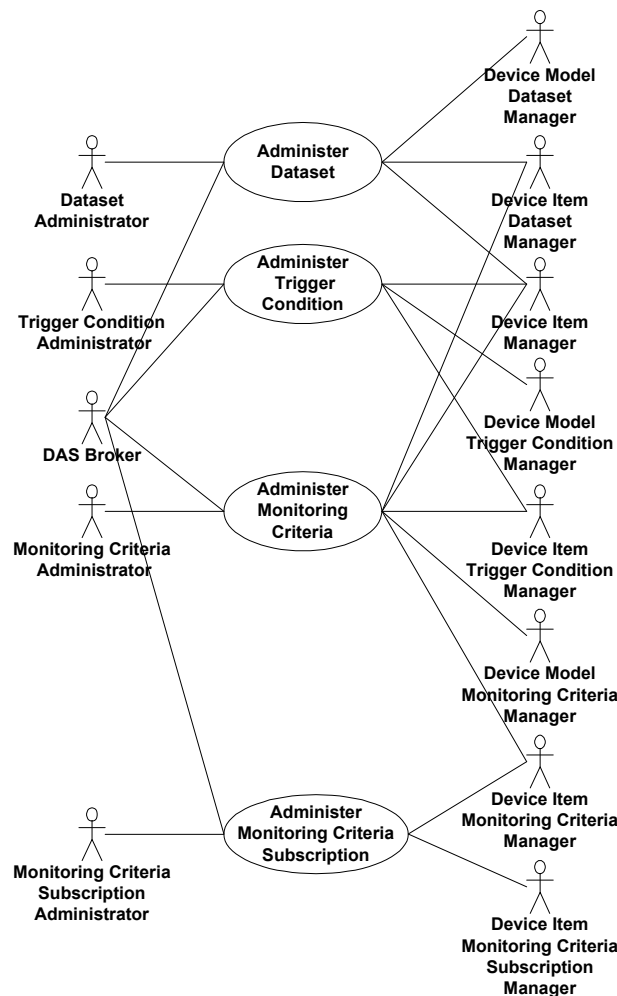


Figure 39 Define Data Access Use Case

- (i) *Administer Datasets.* A *Dataset Administrator* administers (creates, modifies or removes) a set of data of interest of a device item. A *dataset* consists of a set of measurement points to observe. *Dataset Administrators* can define an entirely new dataset for a device item (*custom datasets*), or define a dataset for a device item from a dataset predefined on a device model (*predefined datasets*). All *Datasets* are public.
- (ii) *Administer Trigger Conditions.* A *Trigger Condition Administrator* administers (creates, modifies or removes) a *trigger condition* of a device item. A trigger condition can be based on time or based on an event. A *Trigger Condition Administrator* can define an entirely new trigger condition for a device item (*custom trigger condition*), or define a trigger condition from a trigger condition predefined on a device model (*predefined trigger condition*). All *Trigger Conditions* are public.
- (iii) *Administer Monitoring Criteria.* A *Monitoring Criteria Administrator* administers (creates, modifies or removes) monitoring criteria of a device item. Monitoring criteria allow for the recording of the status of a system at a specific time, the occurrence of an event or the recording of the change on the composition of a device item. *Monitoring Criteria Administrators* can define entirely new monitoring criteria for a device item (*custom monitoring criteria*), or define monitoring criteria from monitoring criteria predefined on a device model (*predefined monitoring criteria*). *Monitoring Criteria Administrators* can define monitoring criteria as *public*, meaning that any supervisor of the system can access monitoring reports of such monitoring criteria, or *private*, meaning that only the creator of the monitoring criteria is allowed to access monitoring reports corresponding to such monitoring criteria.
- (iv) *Administer Monitoring Criteria Subscription.* A *Monitoring Criteria Subscription Administrator* administers (creates, modifies or removes) subscriptions of interest on certain monitoring criteria. The subscriber can chose between being automatically uploaded with monitoring reports corresponding to such monitoring criteria when available, or receiving a notification of the availability of monitoring reports corresponding to subscribed monitoring criteria.

In all these use cases, we used the term *administer* as a generic term to refer to *create, modify* and *remove*. Typically, the *Administrator* of the system will implement the roles corresponding to the administration of datasets, trigger conditions, monitoring criteria, and subscriptions to monitoring criteria. But, there may be systems where a *Supervisor* is allowed to administer some things such as its subscriptions to monitoring criteria, for instance. The use of different elementary roles for the administration of datasets, trigger conditions, monitoring criteria, and subscriptions to monitoring criteria makes our specification more independent from decisions that should be taken later, in the design phase of a particular DAS.

We use an activity diagram, shown in Figure 40, to specify the sequence in which these use cases may be carried out:

- (a) Administer datasets until all datasets have been administered. Then, go to b), to administer trigger conditions; to c), to administer monitoring criteria; to d), to administer monitoring criteria subscriptions; or to the end, if all the datasets, trigger conditions, monitoring criteria and monitoring criteria subscriptions have been administered.
- (b) Administer trigger conditions until all trigger conditions have been administered. Then, go to a), to administer datasets; to c), to administer monitoring criteria; to d), to administer monitoring criteria subscriptions; or to the end, if all the datasets, trigger conditions, monitoring criteria and monitoring criteria subscriptions have been administered.
- (c) Administer monitoring criteria until all monitoring criteria have been administered. Then, go to d), to administer monitoring criteria subscriptions; or to the end, if all the datasets, trigger conditions, monitoring criteria and monitoring criteria subscriptions have been administered.
- (d) Administer monitoring criteria subscriptions until all monitoring criteria subscriptions have been administered. Then, go to the end.

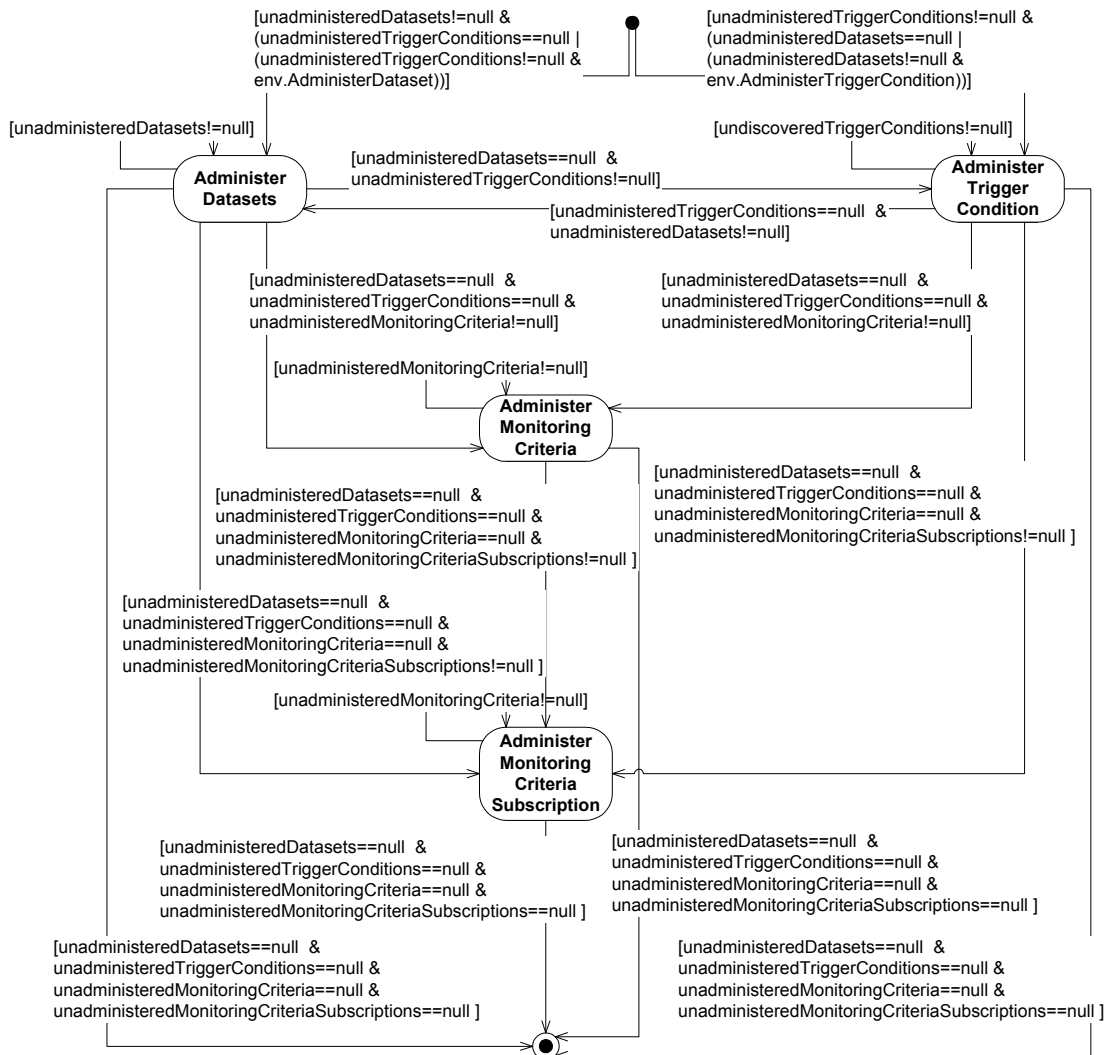


Figure 40 Define Data Access Activity Diagram

6.3.3 Access Data

This use case implements the “data access/retrieval” functional requirement, which is defined by the OMG in DAIS RFP [7] as:

“Mechanisms to define immediate data access retrieval upon request. The data elements transferred may be simple or structured types. A client could define a set of data to be retrieved at a time.”

Data accessing allows supervisors to obtain the current value (a quantitative or qualitative measurement) of a measurement point or the current monitoring reports corresponding to a certain monitoring criteria (e.g. the current values of a full dataset).

We refined the *Access Data* use case into the following role-based use cases, shown in Figure 41:

- (i) *Access Observations*. An *Observation Requester* obtains, by request, observations corresponding to the values of one or more measurement points. The system may offer *Observation Requesters* ways to specify filters to access specific observations of measurement points (e.g. the last observations, the observations within a specific interval of time, the observations that exceed certain values).
- (ii) *Access Monitoring Reports*. A *Monitoring Report Requester* obtains, by request, monitoring reports taken on a device item. The system may offer *Monitoring Report Requesters* ways to specify filters to access specific monitoring reports. (e.g. the last monitoring report of a certain monitoring criteria, the monitoring reports of a certain monitoring criteria within a specific interval of time).

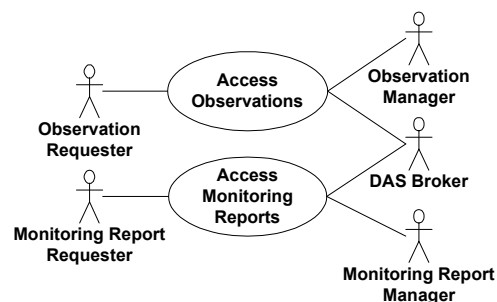


Figure 41 Access Data Use Case

In these use cases we introduced a manager role for observations and a manager role for monitoring reports, as we consider that we have to offer DAS developers the choice of implementing the management of observations and monitoring reports independently. Typically, the same actor (e.g., the *Supervisor* of the system) will implement the roles *Observation Requester* and *Monitoring Report Requester*. But, there may be systems where supervisors are only allowed to access observations of the systems, and certain users with special privileges (e.g., the *Administrator* of the system) are allowed to access monitoring reports of the systems. In other systems, users might be only allowed to access monitoring reports. Again,

the use of different elementary roles for anything that can be accessed makes our specification more independent from decisions that should be taken later, in the design phase of a particular DAS.

We use an activity diagram, shown in Figure 42, to specify the sequence in which these use cases may be carried out:

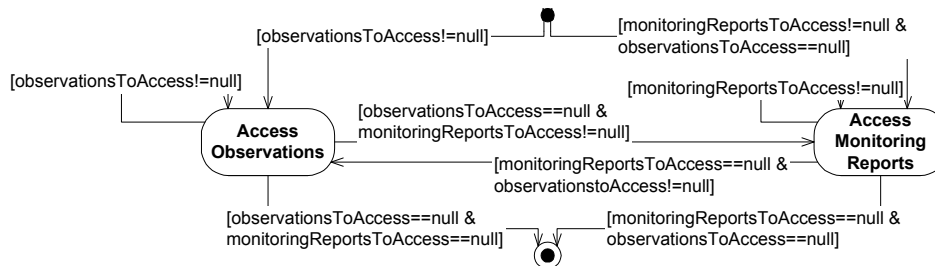


Figure 42 Access Data Activity Diagram

- (a) Access observations until all the observations to access have been accessed. Then, go to b), to access monitoring reports; or to the end, if all observations and monitoring reports to access have been accessed.
- (b) Access monitoring reports until all the monitoring reports to access have been accessed. Then, go to a), to access observations; or to the end, if all observations and monitoring reports to access have been accessed.

6.3.4 Notify Data Availability

This use case implements the “event notification for availability of data” functional requirement, which is defined by the OMG in DAIS RFP [7] as:

“Mechanisms to allow the industrial system broadcasting events outside itself to which clients can subscribe in order to receive a notification that new data is available to be accessed.”

This use case allows supervisors to be notified when relevant data is available. The notification process is based on the *push* model. Monitoring criteria and subscriptions for receiving notifications for availability of them are defined by means of the *Define Data Access* use case. The DAS will notify supervisors of monitoring reports corresponding to monitoring criteria subscribed by them.

We refined the *Notify Data Availability* use case into a single role-based use case, shown in Figure 43:

- (i) *Notify Data Availability*. A *Monitoring Criteria Subscriber* receives a notification that new data is available. Data are monitoring reports corresponding to monitoring criteria subscribed by the *Monitoring Criteria Subscriber*. The notification process is based on the *push* model.

Typically, the *Supervisor* of the system will implement the role *Monitoring Criteria Subscriber*. But, there may be systems where supervisors are not allowed to receive notifications, and only certain users with special privileges (e.g., the

Administrator of the system) are allowed to receive notifications of monitoring reports of the systems. Again, the use of a role for the subscriber of monitoring reports makes our specification more independent from decisions that should be taken later, in the design phase of a particular DAS.

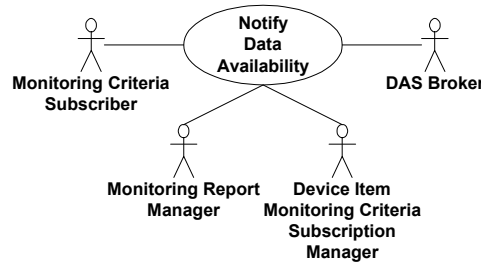


Figure 43 Notify Data Availability Use Case

6.3.5 Upload Data

This use case implements the “event driven data upload” functional requirement, which is defined by the OMG in DAIS RFP [7] as:

“Mechanisms to define event driven data retrieval sequence, by which data delivery can be done automatically upon the occurrence of a notification for availability of data.”

This enables the automatic sending of relevant data to supervisors when available. The upload process is based on the *push* model. Monitoring criteria and subscriptions for receiving uploads of data are defined by means of the *Define Data Access* use case. The DAS will upload to supervisors monitoring reports corresponding to monitoring criteria subscribed by them.

We refined the *Upload Data* use case into a single role-based use case, shown in Figure 44:

- (i) *Upload Data. Monitoring Criteria Subscribers* receive monitoring reports corresponding to monitoring criteria subscribed by them.

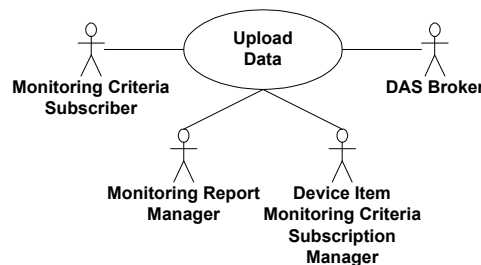


Figure 44 Upload Data Use Case

As with the notification of availability of data, typically, the *Supervisor* of the system will implement the role *Monitoring Criteria Subscriber*. But also there may be cases where only certain users with special privileges (e.g., the *Administrator* of the system) are allowed to receive uploads of monitoring reports of the systems. Again,

the use of an elementary role for the subscriber of monitoring reports makes our specification more independent from decisions that should be taken later, in the design phase of a particular DAS.

6.4 Summary

In this chapter we described a conceptual model and a role-based use case model of a generic DAS. These models give DAS developers an abstraction of DASs. They also provide DAS developers with an external specification of a generic architecture for DASs. This generic DAS specification enables the comparison of existing DAS products and standards. Additionally, it provides the DAS developers that aim to develop a specific DAS with a starting point for the design of a specific DAS. We based our specification on conceptual and role-based use case modeling, which makes it quite generic and independent from design and implementation decisions for a particular DAS. The role-based use cases are specified in Appendix B. The elementary roles are specified in Appendix C.

7. Discussion

Parts of this work appeared in [10].

7.1 Introduction

In this chapter we discuss key issues about the conceptual and role-based use case models. We also discuss the development process that we followed in this thesis and we explain the benefits of this development process versus traditional development processes.

7.2 Conceptual Model

In this section we discuss the relationship between device models and device items; we define how to assign a *Global Unique Identifier* (GUID) for device models and device items; we discuss the composition of device models and device items introducing a new pattern, called *Model Composite*, for the management of the composition of models; we explain how our conceptual model supports the notion of *Plug&Play*; and finally we explain the mapping between sampled and physical values.

7.2.1 Device Models vs. Device Items

We refer as *device* to a generic concept for any industrial system or sub-system. In our model, we represent a *real world device* as an instance of *Device Item*. We represent the type of a device, commonly known as its *model*, as an instance of *Device Model*. The relationships between instances of *Device Item* and instances of *Device Model* are shown in Figure 45.

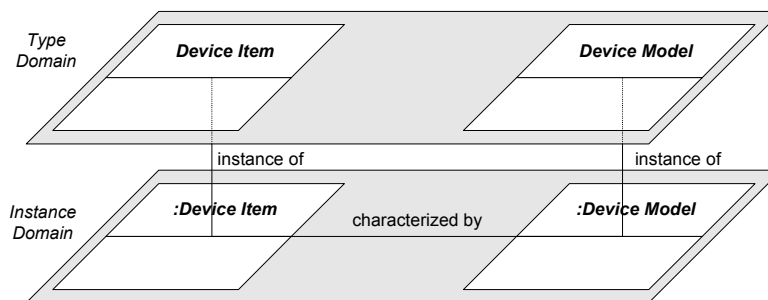


Figure 45 Device Models vs. Device Items

An instance of *Device Item* is always characterized by an instance of *Device Model*. An instance of *Device Model* characterizes a set of instances of *Device Item*.

7.2.2 Naming Management

A Global Unique Identifier (GUID) must be assigned to each device model and device item. In this section we define how to assign a GUID for device models and device items.

Device Model Identifier

Device model designers are responsible for assigning a designer specific model identifier to their device models. This identifier, which we named *modelID*, enables the distinction between two different device models belonging to the same designer. A designer identifier, which we named *designerID*, enables the distinction between two different device model designers. As a result, a *deviceModelGUID* is obtained from the concatenation of *designerID* and *modelID*.

$$\text{deviceModelGUID} = \text{designerID} \ \& \ \text{modelID}$$

Device Item Identifier

Device manufacturers are responsible for assigning a unique identifier, which is named *serialNumber*, to each device item. *serialNumber* uniquely identifies device items of the same device model manufactured by a manufacturer. In order to be able to globally identify a device item, it is necessary to include the *deviceModelGUID*. As a result, a *deviceItemGUID* is obtained from the concatenation of its corresponding *deviceModelGUID*, the *manufacturerID* and a *serialNumber*.

$$\begin{aligned} \text{deviceItemGUID} &= \text{deviceModelGUID} \ \& \ \text{manufacturerID} \ \& \ \text{serialNumber} \\ \text{deviceItemGUID} &= \text{designerID} \ \& \ \text{modelID} \ \& \ \text{manufacturerID} \ \& \ \text{serialNumber} \end{aligned}$$

7.2.3 Composition Management

Device Items Composition Management

In DASs, tree structures allow us to efficiently define an industrial system. An industrial system is usually composed of many parts, which can also be composed of many other parts in a *part-whole* hierarchy. For example, an HVAC (heating, ventilation and air conditioning) system is composed of subsystems such as heating coil, cooling coil, supply fan, etc., that can be composed of other subsystems such as temperature sensors, ventilation sensors and so on. *Part-Whole* relationships are commonly used to model the construction of composite objects out of individual parts. *Part-Whole* relationship categories and their application in object-oriented analysis are further discussed by *Motschnig-Pitrik and Kaasboll* in [75]. One way to represent *part-whole* relationships of systems is by using the *Composite* [41] pattern. We used this pattern to organize device items.

Device Models Composition Management

An instance of *Device Item* is characterized by an instance of *Device Model*. As a result, there is an analogous relationship between pairs of device models and

pairs of the corresponding device items. But, in the case of device models, the same device model may be used many times as part of the same complex device model. This impedes the use of the *Composite* pattern for the management of device models, as it would be not possible to distinguish between the different instances of the same *Device Model*.

Example: as shown in Figure 46, an instance of `VehicleModelA` is composed of two instances of `DoorModelB`. This is typically the case of vehicles with a left door and a right door of the same model. The problem is that there is no way to distinguish between the two instances of the same model `DoorModelB`, which are both part of `VehicleModelA`.

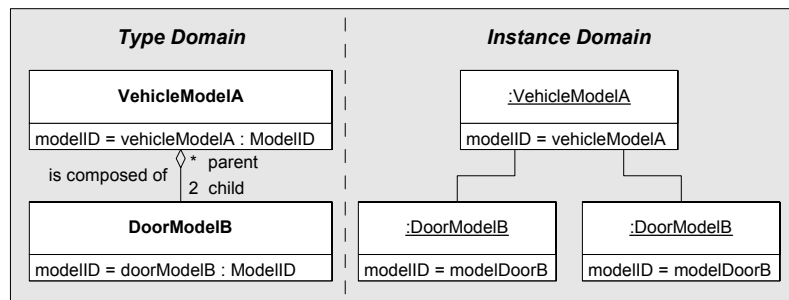


Figure 46 Example of Model Composition without Functional Model

An elegant manner to solve this problem is to use the *Model Composite* pattern. The *Model Composite* pattern, shown in Figure 47, is our own variation of the *Composite* pattern, to represent *part-whole* hierarchies of models. *Model* implements default behavior for a model. *Complex Model* defines behavior for a model that is composed of other models. *Functional Model* is a specialization of *Model* that represents a model that is part of a complex model. We called it *functional* because it implements a function within a complex model. Using this pattern we may have many different instances of *Functional Model* that inherit from the same instance of *Model*, but each of them implements a different function on a complex model.

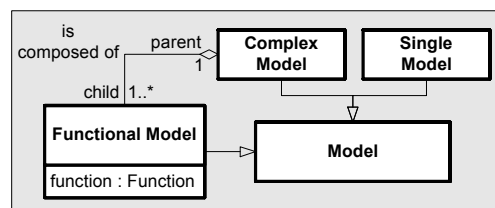


Figure 47 Model Composite Pattern

Example: as shown in Figure 48, a `VehicleModelA` is composed of a `VehicleModelALeftDoorDoorModelB`, implementing the function of `leftdoor` and a `VehicleModelBRightDoorDoorModelB`, implementing the function of `rightdoor`, which inherit both from `DoorModelB`. In this way we can distinguish between the two instances of the same model `DoorModelB`.

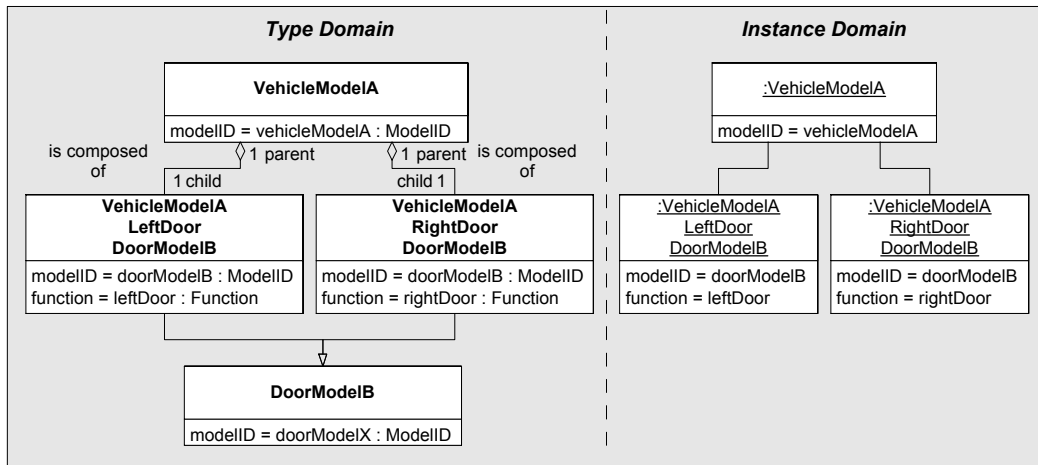


Figure 48 Example of Model Composition with Functional Model

7.2.4 Plug&Play

Plug&Play indicates that a system has the ability to automatically configure itself. The system must be able to detect changes in its composition to adapt its configuration to the new composition. One of the most known *Plug&Play* initiatives is Microsoft's *Plug&Play* (PnP) [76], which is a framework architecture for PCs to enable the automatic configuration of expansion cards and other devices. Other initiatives, such as Universal *Plug&Play* (UPnP) [77] and Jini [58], enable the *Plug&Play* of systems, or services, in a network.

A *Plug&Play* DAS would allow supervisors of devices to register to be notified when changes in the composition of the system happen. A *Plug&Play* DAS automatically detects changes in the composition of the system and notifies to interested supervisor of such changes. If the real world devices support the *Plug&Play* functionality, a *Plug&Play* DAS may subscribe itself in the devices to receive notifications when changes on the composition of such devices happen. Otherwise, a *Plug&Play* DAS may check periodically the composition of device items to detect eventual changes on their composition.

Our conceptual model supports the notion of *Plug&Play* through the concepts of *Device Model Composition Monitoring Criteria* and *Device Item Composition Monitoring Criteria*; these concepts allow for the definition of composition monitoring criteria, predefined in a device model or defined specifically in a device item, respectively. A composition monitoring criteria groups a set of devices. A supervisor may subscribe to receive a notification, by means of a monitoring report, when a change in the composition of, at least, one of the devices of such monitoring criteria happens. Our conceptual model defines the concepts that are necessary to enable the development of a *Plug&Play* DAS, but it does not force developers the use a specific technology. In an actual implementation, developers will chose the *Plug&Play* technology that fits better with their specific requirements.

7.2.5 Physical Values vs. Sampled Values

In DASs, it is very common for the value actually measured (we refer to this value as *sampled value*) to not correspond to the *physical value*. We need a way to

record a mapping policy that makes it possible to calculate the *physical value* from the *sampled value*. Eventually, the units in which a measurement is taken could not correspond to the measurement of the associated phenomenon type. This could happen because the measurement units depend on the sensor uses to acquire the measurement and not on the phenomenon type. Thus, we also need to record the units of the measurement in order to makes it possible to calculate the *physical value* in *phenomenon type units* from the *physical value* in *measurement units*. In order to record this information we introduced the concept of measurement type and mapping policy, shown in Figure 49.

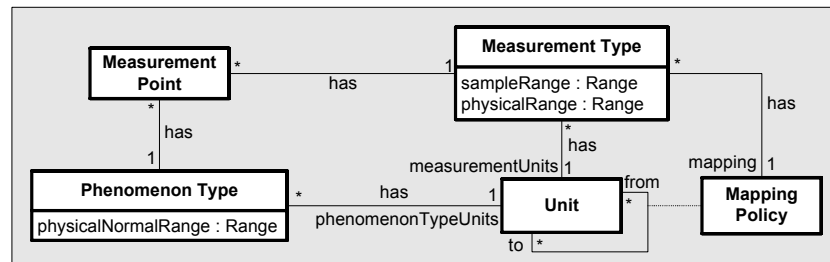


Figure 49 Physical Values vs. Sampled Values

7.3 Role-based Use Case Model

In this section we discuss the possible representations of the system in the use cases; we discuss the use of elementary roles rather than actors; we give some techniques to specify the interactions across use cases and the scenarios of use cases; we explain the use of the *Broker* pattern in the use cases; we describe a new pattern, called *Administrator-Manager*; and finally, we present the templates that we used for specifying role-based use cases and elementary roles.

7.3.1 Elementary Roles vs. Actors

We used actors to represent not only outside users but also the system itself. Additionally, actors can play several elementary roles in use cases. Therefore, an actor can be seen as a composition of elementary roles, each of them playing a single function in a use case. We used elementary roles rather than actors in the use cases, because this allows us to specify the system independently of architectural choices, requirements, QoS, and/or available technologies specific for a particular system. The resulting specification can, then, be reused in different implementations of similar systems by mapping roles into actors according to the requirements of a specific system. This process is illustrated with an example in Figure 50. In this process we first identify the use cases using real world actors that cooperate to realize a use case. Then, we focus on the elementary roles that cooperate to realize a use case. After that we can specify the use case by means of sequence diagrams using elementary roles. As a result of this specification, we obtain the interfaces of the elementary roles. Consequently, this specification is generic and we can map each elementary role to actors depending on the requirements of particular systems.

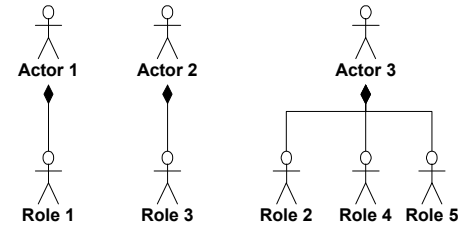
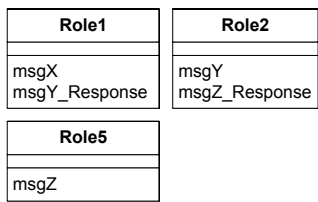
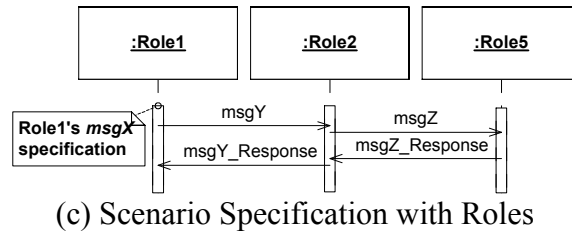
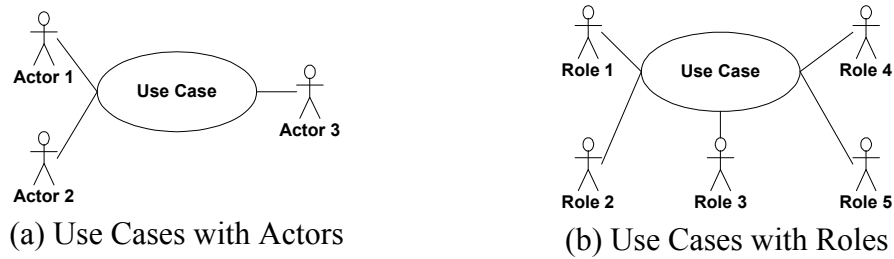


Figure 50 Elementary Roles vs. Actors

7.3.2 Representation of the System

People use different approaches to represent the system in the use cases. These approaches are mainly:

- (i) *Not to represent the system in the use cases.* The system does exist in the use cases but it does not appear explicitly in the design of the use cases. It is implicit.
- (ii) *Represent the system as a box containing the use cases.* The system appears explicitly in the design of the use cases.
- (iii) *Represent the system itself in the use cases.* The system appears as another actor who takes part in the use case.

We found it very useful to represent the system itself in the use cases, because in this way it is easier to define the role of the system on each use case and to find out the interfaces that the system has to provide to other participants in the use case in order to carry out this use case. Therefore, we adopted the approach described in (iii), shown in Figure 51.

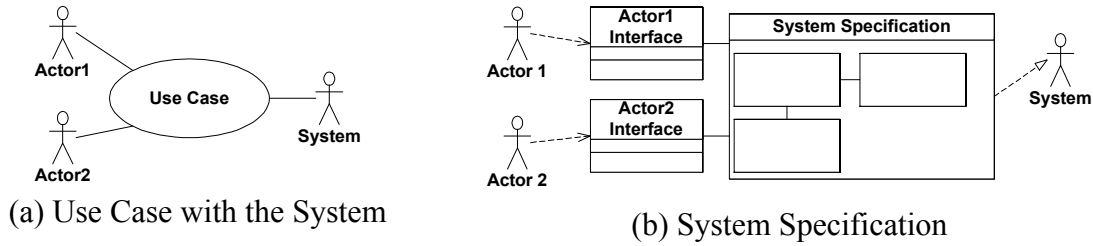


Figure 51 Representation of the System in the Use Case

7.3.3 System Behavior Modeling

An external specification of a generic system must allow the specification of different system behaviors depending on specific execution constraints and types of control flow. In this section we give some techniques to specify the interactions across use cases and the scenarios of use cases in such a way that the specification enables the implementation of systems with different execution constraints and types of control flow.

Modeling of the Interactions across Use Cases

We propose the use of UML activity diagrams to specify the interactions across use cases. An activity diagram is the UML notation for an activity graph, which is a special form of state machine intended originally to model computations and workflows. According to the UML notation [29] of activity diagrams stick arrowheads represent control flow; dashed arrows with stick arrowheads represent object flow; and labels in the arrows represent conditions to be satisfied to pass from one state to another state (or eventually a pseudo-state).

An external specification of a generic system must be able to specify many particular systems with different interactions across use cases depending on particular execution constraints. To achieve this, we propose to:

- (i) Represent each use case as an action (or state) in the activity diagram.
- (ii) Use stick arrowheads to represent the interactions across use cases.
- (iii) Define the transitions across use cases depending on the actual values of environment variables. We represented environment variables with the “env.” prefix. Depending on the actual values of these environment variables a particular system will implement a behavior or another.

Example: as shown in Figure 52, we specified that from the UseCase1, if condition1 is *false*, and condition2 and condition3 are *true*, we can go either to the UseCase2 or to the UseCase3. Both transitions are possible. The actual behavior of the system depends on the env.UseCase2 and env.UseCase3 environment variables.

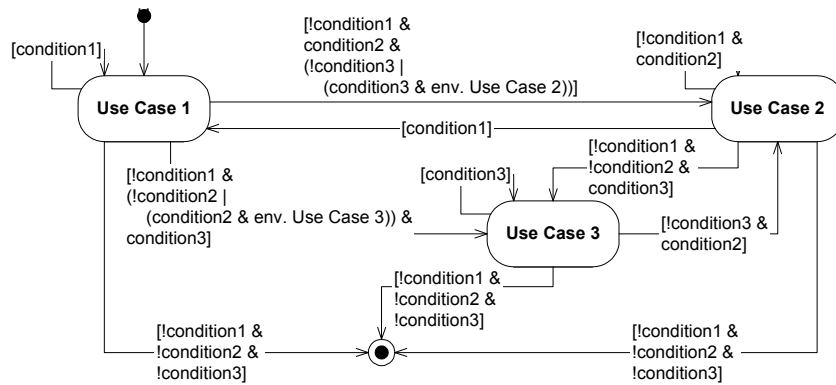


Figure 52 Example of Modeling of the Interactions across Use Cases

Modeling of the Scenarios of Use Cases

We propose the use of sequence diagrams to specify scenarios of use cases. A sequence diagram is a UML notation for an interaction graph that focuses on time sequences. According to the UML notation [29] of sequence diagrams filled solid arrowheads are used to represent procedure calls or other nested flows of control; stick arrowheads are used to represent flat flows of control; half stick arrowheads are used to represent asynchronous messages; and dashed arrows with stick arrowheads are used to represent “return” messages from procedure calls.

An external specification of a generic system should be independent of a specific type of control flow, enabling the design of systems with nested or flat flows of control. To achieve this, we propose to:

- (i) Use stick arrowheads to represent any kind of message `msgX`.
- (ii) Use `msgX_Response` as a convention to name the return message corresponding to `msgX`.

Example: as shown in Figure 53, we specified that `Role1` sends the `msgX` message to `Role2`, and `Role2` replies with the `msgX_Response` message. If the implemented system is an asynchronous system `Role1` can continue performing processes, `msgX_Response` being sent later asynchronously by `Role2`. If the implemented system is a synchronous system `msgX` blocks `Role1` until `Role2` replies with `msgX_Response`. Normally, the interface of `Role2` implements a function that handles the `msgX` message, and `Role1` implements a function that handles the `msgX_Response`. However, in synchronous systems implemented by means of procedural calls, the interface of `Role2` implements a function that handles the `msgX` message, but `Role1` does not implement a function that handles the `msgX_Response`, as `msgX_Response` corresponds to the return of the `msgX` message.

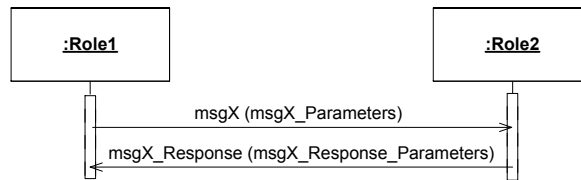


Figure 53 Example of Modeling of the Scenarios of Use Cases

7.3.4 Broker Pattern

The *Broker* pattern is defined in [22]:

“The *Broker* pattern can be used to structure distributed software systems with decoupled components that interact by remote service invocations. A broker component is responsible for coordinating communication, such as forwarding requests, as well as for transmitting results and exceptions”.

We made a wide use of this pattern in our specification of a generic DAS. Due to the genericness of our specification we cannot predict if a role will be implemented internally by a component of the system, or an outside user or system. The *System Broker* allows us to decouple roles allowing us to remain independent of design choices. The *System Broker* adds an extra level of indirection that allows roles to ignore whether other roles are implemented internally or externally. We introduce the *System Broker* as another role that takes part on all the use cases, as shown in Figure 54.

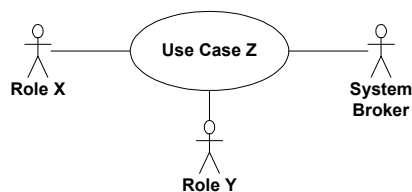


Figure 54 Example of Representation of the System Broker in the Use Cases

Roles register on the *System Broker* to handle certain messages, as shown in Figure 55.

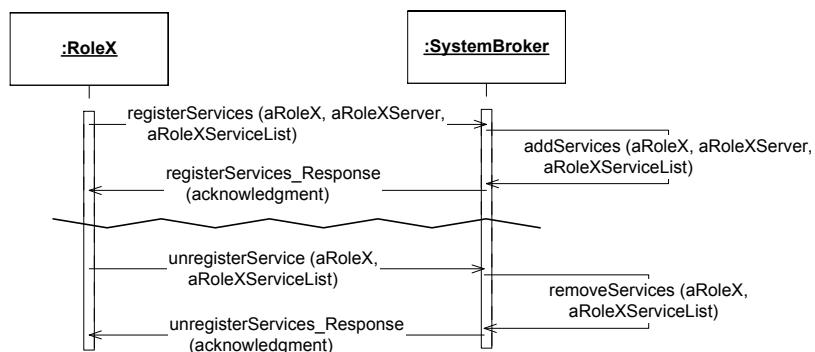


Figure 55 Example of (Un)Registration of Services in the System Broker

All the messages between roles pass through the *System Broker*. The *System Broker* is responsible for establishing a communication between a particular role

(RoleX) and another particular role (RoleY). The *System Broker* finds the server of a certain role that implements a certain service and forwards the request to it. Eventually, an asynchronous response would pass through the *System Broker* in the same way. In Figure 56 we show an example of communication between roles using the *System Broker*.

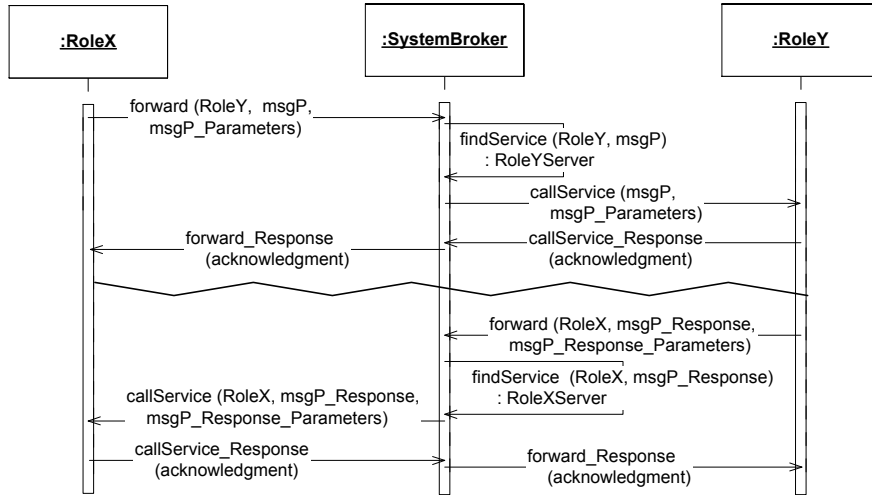


Figure 56 Example of Comm. between Roles using the System Broker

For simplicity in the sequence diagrams, we represent the communication between roles through the *System Broker* in the simplified way shown in Figure 57.

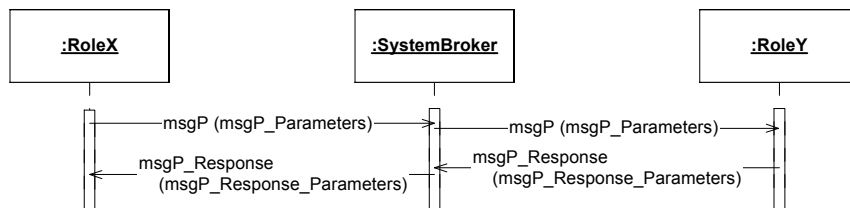


Figure 57 Example of Simplified Comm. between Roles using the System Broker

7.3.5 Administrator-Manager Pattern

The so-called *Administrator-Manager* use case pattern is shown in Figure 58. This new pattern concerns the administration of instances of a particular type X. With the term `Administer X` we refer to the creation of new instances of X, and the modification or deletion of existing instances of X. There are always two roles that take part in an `Administer X` use case. The X Administrator is responsible for administering instances of X, that means, for creating new instances of X or for modifying or removing existing instances of X. The X Administrator is typically an outside user with special privileges. The X Manager is responsible for managing instances of X, that means, recording them into a persistent data support, check duplication of instances and so on. The X Manager may typically be an interface to an internal or external database.



Figure 58 Administrator-Manager Pattern

Some examples of the utilization of this pattern can be found in the refined use cases, specified in Appendix B, of the *Define Data Access* use case.

7.3.6 Specification of Role-based use cases

In our external specification of a generic architecture for DASs we used the following template to specify role-based use cases:

Use Case	The name of the use case
Roles	The names of the roles that take part in the use case. In the high-level use cases we use actors whereas in the logical use cases we use roles.
Type	We categorized use cases according to the criteria defined by <i>Larman</i> in [78]. By one way, use cases are classified as <i>primary</i> , <i>secondary</i> or <i>optional</i> : <i>primary</i> use cases represent major common processes, <i>secondary</i> use cases represent minor processes and <i>optional</i> use cases represent processes that many not be tackled. By other way, use cases are classified as <i>essential</i> or <i>real</i> : <i>essential</i> use cases are expressed in an ideal form that is implementation independent, whereas <i>real</i> use cases describe the process in terms of a specific design. In this thesis we focused only on <i>primary</i> and <i>essential</i> use cases.
Description	A short description or overview of the use case.
Pre-conditions	Some conditions that must be fulfilled before running the use case.
Post-conditions	Some conditions that must be fulfilled after having run the use case.
Use Case Diagram	The diagram corresponding to the use case.
Known Concepts	The set of concepts and relationships of the system that the roles that take part in the use case have to know to carry out the use case. We represent the roles in the conceptual model to specify the relationships and cardinalities among roles, and between roles and concepts. We used the stereotype <i>object</i> in the concepts to specify that these concepts correspond to a generic representation of objects of the system and not to an internal representation of concepts in the context of a particular role.
Example Scenario	A sequence diagram that specifies a possible scenario, as an example.

7.3.7 Specification of Roles

In our external specification of a generic architecture for DASs we used the following template to specify roles:

Role	The name of the role
Description	A short description or overview of the role.
Policies	Some policies that dictate the role behavior.
Interfaces	The interfaces that the role offers.
Known Concepts	The set of concepts and relationships of the system that the role has to now to carry out its expected behavior. We represent the role concept in the conceptual model as <i>Myself</i> . This allows us to specify the cardinalities between the role and some concepts. The concepts correspond to the representation of concepts of the system in the context of the role. For simplicity, we used the same names to represent the same concepts of the system in different roles, but these concepts correspond to different representations as they correspond to different role contexts.

7.4 Development Process

In this section we discuss the development process that we followed in this thesis and we explain the benefits of this development process versus traditional development processes.

Traditional development processes have the following phases:

- *Analysis*. The main objectives of this phase are to understand the business/domain, understand the problem(s) to solve, collect the client requirements, find possible solutions, choose the solution that fits better with the client requirements and specify this solution.
- *System Design*. The objective of this phase is to design the solution.
- *System Implementation*. The objective of this phase is to implement the solution.

Many development processes, such as Catalysis, go throughout these phases recursively, non-linearly, iteratively and in parallel.

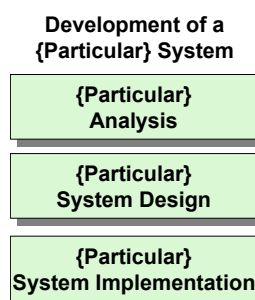


Figure 59 Traditional Development Process

In this thesis we propose to develop an external specification of a generic system from the analysis of multiple systems and/or standards, and to apply such a specification to the development of many particular systems. The development process followed is shown in Figure 60.

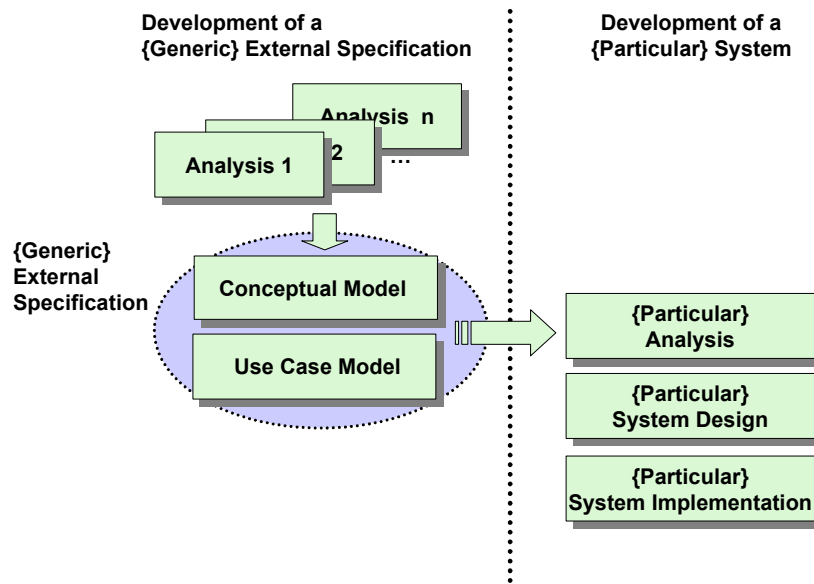


Figure 60 Followed Development Process

An external specification of a generic system is independent of particular design decisions. A generic system specification can be applied in the analysis phase of a particular system to better understand the problem and to easier specify the best solution, depending on the specific requirements of a particular system. A generic system specification can also be used as a starting point for the design phase of a particular system. As a result, a generic system specification will save time and reduce the costs of the development of a specific system. As a generic system specification can be applied to the development of many particular systems, the number of particular systems developed multiplies the benefits.

The drawback of the followed development process is the complexity of the design of an external specification of a generic system. This makes that this development process gives only real benefits when designers have to develop many similar systems. In this thesis it was impossible for us to quantify the savings on the development costs of systems by the application of this development process. However, we strongly believe that this development process gives substantial benefits when developers have to develop at least two similar systems.

7.5 Summary

In this chapter we discussed key issues about the DAS conceptual model and the DAS role-based use case model. We also discussed the development process that we followed in this thesis and we explained the benefits of this development process versus traditional development processes.

8. Application and Validation

8.1 Introduction

In this chapter we explain the applications of an external specification of a generic system. The most direct application of a generic system specification is for the writing of a RFP for a new standard. Another potential application of a generic system specification is for the evaluation of existing systems, standards or RFP responses. We illustrate this by using our generic DAS specification to compare the OMG's DAIS RFP, the different DAS standards and the RoMain system. Finally, a generic system specification can be applied in the development of a particular system. This will significantly reduce the development costs of a specific system. We illustrate this by means of an example of development of a DAS for railway equipment based on our generic DAS specification.

8.2 Issuing/Replying a RFP

This is probably the most direct application of a generic system specification. *RFP Issuers* may use a generic system specification as a guide to specify the static concepts that a proposal of standard in request for a new RFP must support and the functionalities that such a standard must deal with, creating and writing down a specific RFP. *RFP Repliers* may use a generic system specification to easier understand and analyze the requirements of this RFP. Additionally, *RFP Repliers* can use a generic system specification to describe their proposal of standard in request to this RFP. In this way a generic system specification acts as an efficient communication mechanism between *RFP Issuers* and *RFP Repliers*, facilitating the creation, writing, understanding and replying of a RFP. A generic system specification can be seen as an actor that actively collaborates in all these actions, as shown in Figure 61.

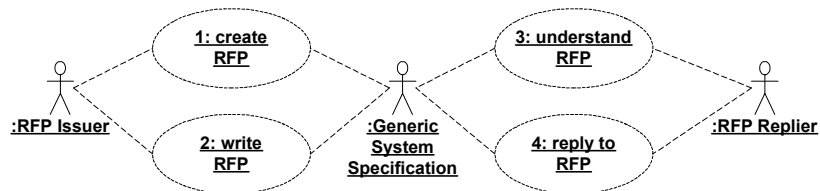


Figure 61 Issuing/Replying a RFP

8.3 Evaluation of Existing Systems or Proposals

Another potential application of a generic system specification is the evaluation of existing systems or standards. Developers may use a generic system specification to check the concepts and functionalities that these systems or standards support.

8.3.1 OMG's DAIS RFP vs. DAS Standards vs. RoMain

In this section, as an example, we use the external specification of a generic architecture for DASs that we developed to compare the OMG's Data Acquisition from Industrial Systems (DAIS) RFP, many DAS standards (OPC, IVI and ODAS) and the RoMain application.

RFP vs. Standard vs. Application

We do not intend to compare RFPs, standards and applications between each other. RFPs, standards and applications have different scopes and purposes, and they exist at different levels. A standard is a specification of concepts and functionalities describing how applications based on this standard should be. An application is a real implementation of a system. If an application is built based on the requirements specified by a standard then we say that this application “supports” the standard. A RFP is a request to the industrial community for proposals to write a new standard. A RFP is more generic, but also less explicit, than a standard. A standard is more explicit, and at the same time less generic, than a RFP. An application is the most explicit of them because in an application there is no ambiguity; it does what it does and no less and no more. These different levels of genericness and explicitness, shown in Figure 62, make it not fair to compare RFPs, standards and applications among each other. Rather than comparing them among each other, we compare each of them to the generic DAS specification. For each of them, we checked whether they have the concepts specified in the generic DAS specification and whether they implement the functionalities described in this specification. The results of these comparisons are shown in Table 2 and Table 3. We have to take in account that when we say that RFPs, standards or applications have a certain concept or implement a certain functionality, it does not mean exactly the same thing. When we refer to an application, it means that the application has this concept or implements this functionality. When we refer to a standard, it means that the standard specifies this concept or functionality. When we refer to a RFP, it means that the RFP introduces this concept or functionality even if not directly.

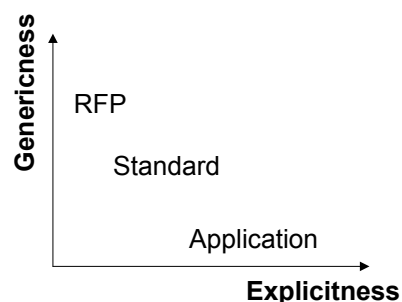


Figure 62 RFP vs. Standard vs. Application

OMG's DAIS RFP

The conceptual model with the concepts the OMG's DAIS RFP supports corresponds to the complete conceptual model shown in Figure 33. The OMG's DAIS RFP requests standard interfaces to access data from heterogeneous industrial systems. As any RFP, it is quite generic and the specification rather ambiguous. This RFP does not refer directly to *Device Models* but information regarding model may be considered part of the device schema that describes the *knowledge-level* data of a device. A *Measurement Point* is called measurement variable, point, data source or data element. *Measurement Type*, *Phenomenon Type* and *Phenomenon* may be part of the semantics associated with data elements. This RFP does not speak directly about complex devices but it may support such systems. This RFP aims to enable a *Plug&Play* functionality, so a concept similar to *Composition Monitoring Criteria* may be implemented. Data can be triggered based on exception or event, so a concept similar to *Event Monitoring Criteria* may be implemented. Data can also be triggered as a self-consistent dataset at a specific date/time, so a concept similar to *Status Monitoring Criteria* may be implemented. The RFP does not make any reference to *public* or *private* monitoring criteria, but the RFP does not avoid implementing such monitoring criteria. Finally, similar concepts to *Measurement*, *Category Observation* and *Monitoring Report* may be implemented to allow recording measurements and monitoring reports taken on a system. The OMG's DAIS RFP specifies *Discover*, *Define Data Access*, *Access Data*, *Notify Data Availability* and *Upload Data* as the core functionalities of any DAS.

OPC

The conceptual model with the concepts OPC supports is shown in Figure 63. OPC is a standard for process control in the automation field. In an *OPC Server* there are many *OPC Items*. An *OPC Item* is quite similar to a *Measurement Point*. It also provides some semantic information such as that corresponding to *Measurement Type* (data type, units, ranges and scales), *Phenomenon Type* and *Phenomenon* (implicit within the data type). As OPC does not support the concept of *model*, this semantic information of an *OPC Item* is duplicated in all *OPC Items* corresponding to the same kind of *Measurement Point*. Additionally, it is not possible to define *Monitoring Criteria predefined* in a device model. We can say that OPC supports partially the notion of *composition* because an *OPC Server* can manage a flat list of devices. However, OPC does not deal with complex hierarchical structures of devices. OPC allows clients to define *custom trigger conditions* and *custom datasets*. *knowledge-level* are called *OPC Groups*, which group a set of *OPC Items* to be retrieved at the same time. However, OPC does not allow clients to define *Composition Monitoring Criteria* to enable the detection of changes on the composition of the system. Therefore, OPC does not enable a *Plug&Play* functionality. OPC specifies all the functionalities defined in the generic DAS specification except, obviously, *Discover Model*, as OPC does not support the notion of *model*.

IVI

The conceptual model with the concepts IVI supports is shown in Figure 64. IVI is a standard for instrumentation to enable the interchangeability among instruments of different vendors. In the IVI specification they refer to *instrument* or *assets* rather than devices. IVI defines many *instrument classes* such as *Oscilloscope*

(IviScope), *Digital Multimeter* (IviDmm), *Function Generator* (IviFGen), *Switch* (IviSwch), and *Power Supply* (IviPower) classes. An *IVI Class* can be considered a similar concept to a generic *Device Model*. IVI classes only deal with single devices. Additionally, IVI specifies the *IVI Measurement and Stimulus System* (IVI-MSS), which allows building complex *virtual instruments* composed of many real instruments. Thus, we can state that IVI supports somehow the *composition* of devices through IVI-MSS. IVI is a standard defined to enable not only the interchangeability among instruments but also the *Plug&Play* of instruments. IVI is based on the VXIPlug&Play standard to enable a *Plug&Play* functionality. IVI classes specify *predefined trigger conditions* and *predefined datasets*. These *trigger conditions* may be *enabled* or *disabled* on a specific device item. Additionally, IVI MSS allows the definition of *custom trigger conditions* and *custom datasets*. Monitoring criteria defined in an IVI instrument class or IVI MSS system is always *public* to any IVI client, as IVI does not support the notion of *private*. IVI specifies all the functionalities defined in the generic DAS specification.

ODAS

The conceptual model with the concepts ODAS supports is shown in Figure 65. ODAS is a standard for PC-based data acquisition systems. ODAS defines standard interfaces for *analog/digital inputs and outputs*. The scope of ODAS is single devices. ODAS does not support the notion of *model*. It does not support either the notion of *composition*. Only *custom datasets* and *custom event-based trigger conditions* may be defined by an ODAS client application. This allows clients to define *status monitoring criteria*. *Monitoring Criteria* defined by a client is always *private* for this client, as ODAS does not support the notion of *public monitoring criteria*. ODAS specifies all the functionalities defined in the generic DAS specification except, obviously, *Discover Model*, as ODAS does not support the notion of *model*.

RoMain

The conceptual model with the concepts the RoMain system supports is shown in Figure 66. RoMain is a simple application for the remote monitoring of railway equipment. RoMain deals with three kinds of devices: *trains*, *vehicles* and *equipments*. RoMain supports the notion of *model*, so there are *train models*, *vehicle models* and *equipment models*. These models record some semantic information common to all the device items associated with a device model, for instance, the list of *properties* (similar to *Measurement Point*) of a device. RoMain also supports the notion of *composition*; a *train item* is composed of *vehicle items* that are also composed of *equipment items*. RoMain neither supports the *Plug&Play* of devices nor the reporting of events. A client application may want to receive the current values of the full set of *properties* of a device. This is the only *predefined status monitoring criteria* a client application is allowed to subscribe. Monitoring criteria are *private* to a client application, as RoMain does not support the notion of *public monitoring criteria*. A client of the RoMain system is neither allowed to define *custom datasets*, nor *custom trigger conditions*, nor *custom monitoring criteria*. RoMain specifies all the functionalities defined in the generic DAS specification except the functionalities regarding the definition of *custom datasets*, *custom trigger conditions* and *custom monitoring criteria*, and the *notification of availability of data*.

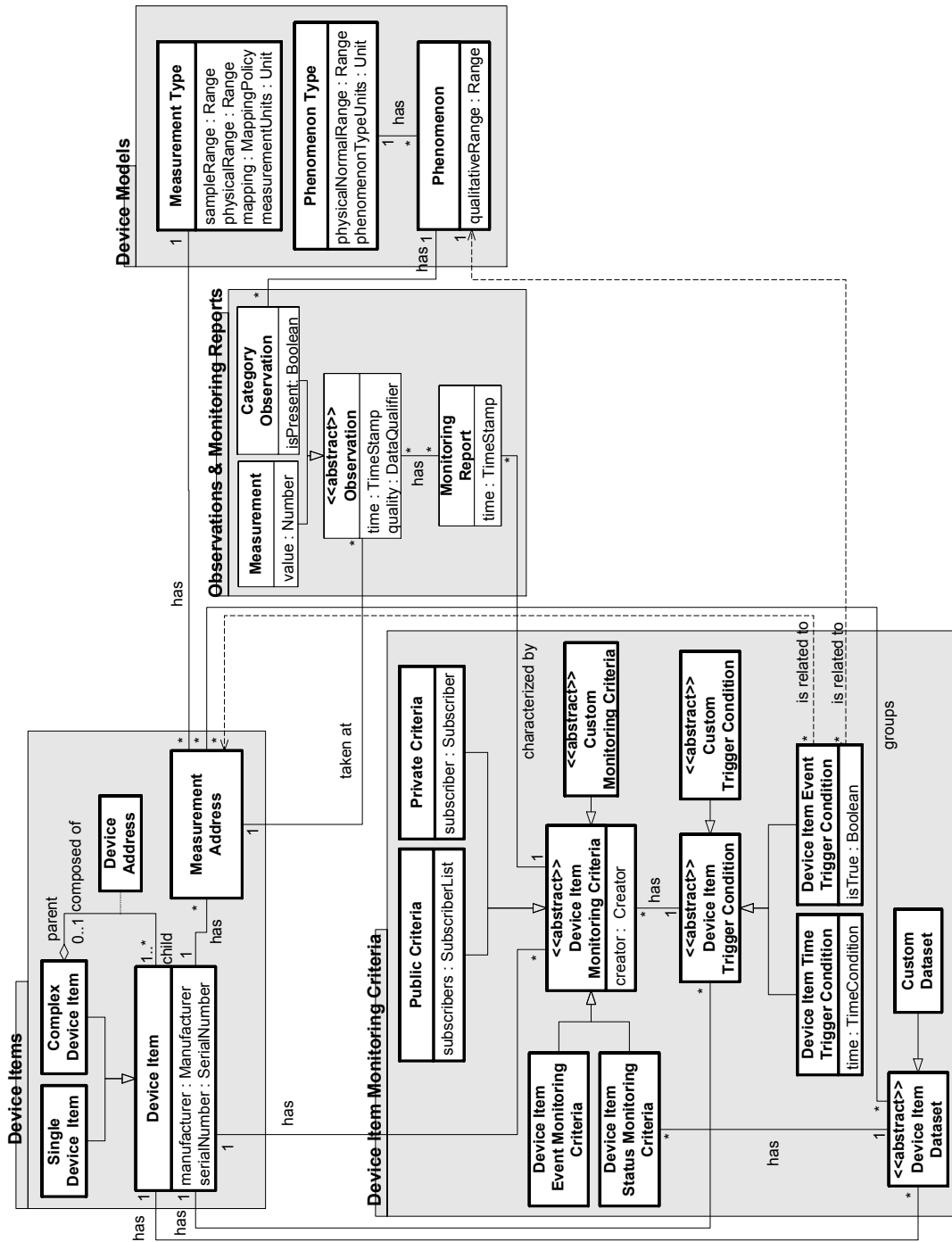


Figure 63 OPC Conceptual Model

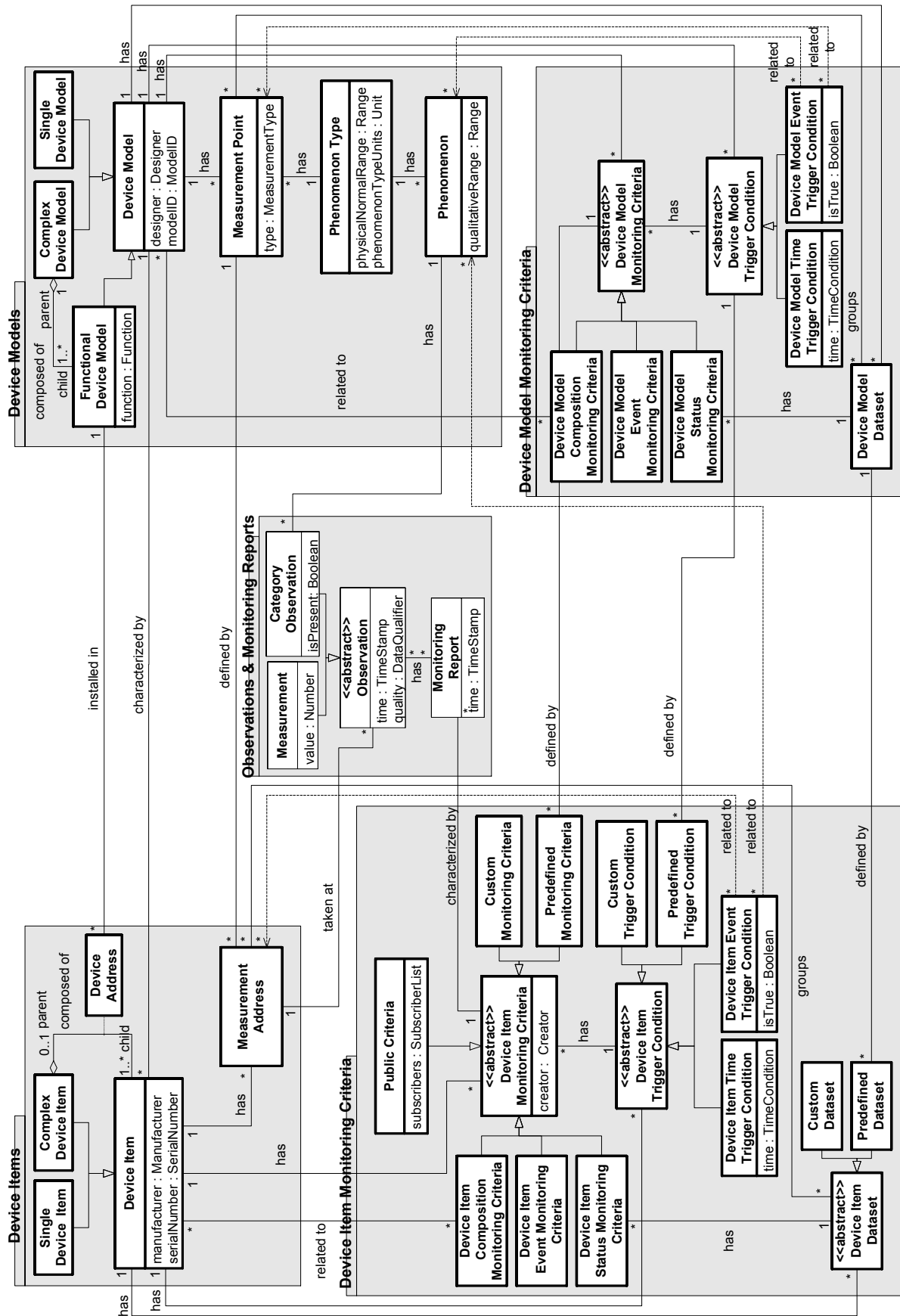


Figure 64 IVI Conceptual Model

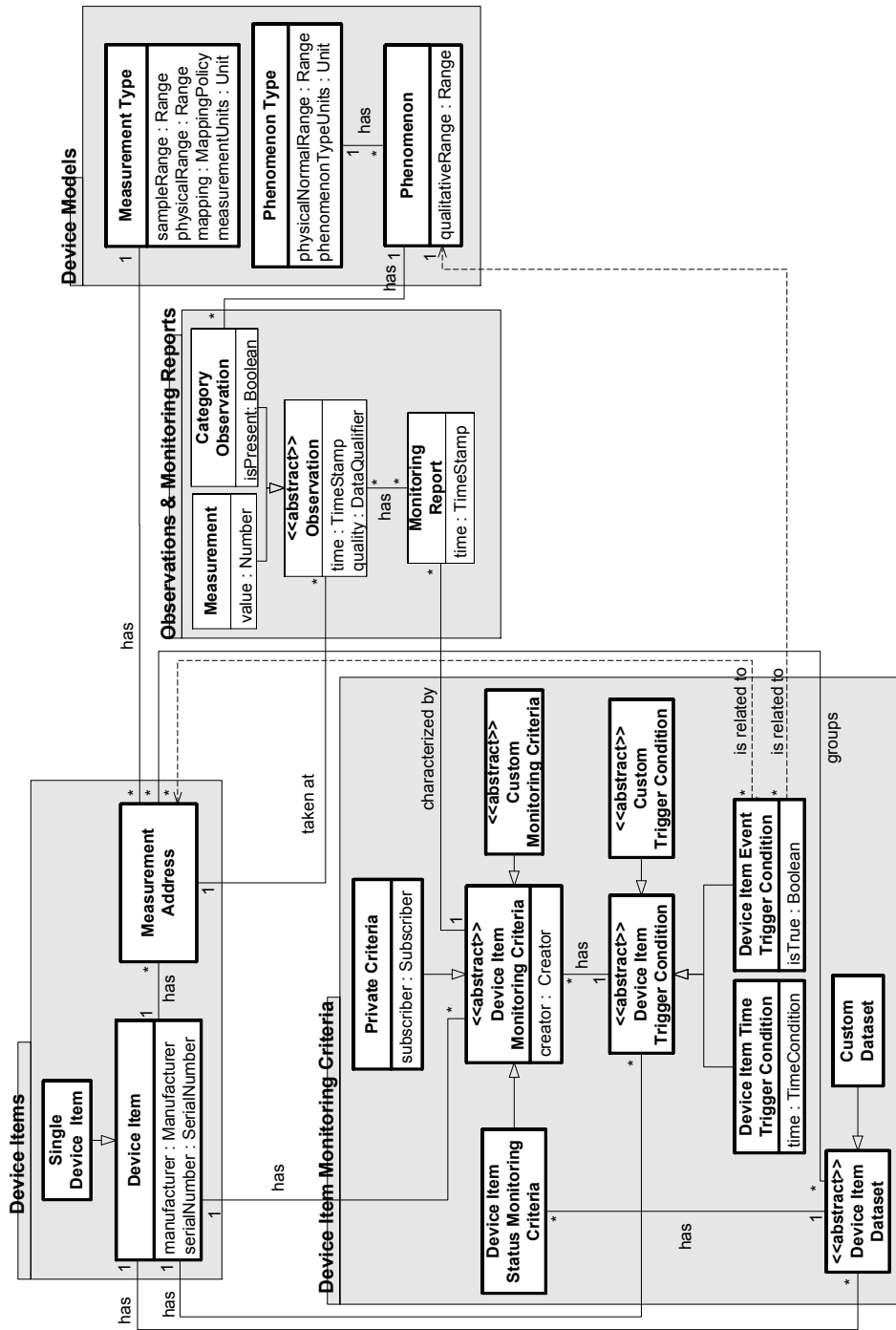


Figure 65 ODAS Conceptual Model

Comparison Summary

In this section we summarize the information that we draw out from the comparisons. The results of these comparisons are shown in Table 2 and Table 3.

- (i) *Lack of Models.* Some systems (such as OPC, IVI and ODAS) do not support the notion of *model*. This implies that many of the concepts regarding models (*Device Model* from the *Device Model* package; all the concepts defined in the *Device Model Monitoring Criteria* package; *Predefined Monitoring Criteria*, *Predefined Trigger Condition* and *Predefined Dataset* from the *Device Item Monitoring Criteria* package) do not exist either. This means that information (such as instances of *Measurement Point*, *Measurement Type*, *Phenomenon Type* and *Phenomenon*) that is common to all the instances of *Device Item* associated with the same instance of *Device Model* must be repeated for each instance. This duplication of *knowledge-level* data is rather inefficient and error prone. As there is not the notion of model, the *Discover Model* functionality has no sense, as there is no model to discover.
- (ii) *Lack of Composition.* Some systems (such as ODAS) do not support the notion of *composition*. This means, they do not deal with complex devices composed of other (single or complex) devices, neither in the domain of models nor in the domain of device items. They only deal with single devices. This is typically the case of a DAS that acquires data from a single device item. Even if there are no complex devices, we cannot state that these systems do not support the *Discover Composition* functionality, in fact they do because they can provide the composition of devices, even if this composition is just one single device.
- (iii) *Plug&Play.* Some systems (such as OMG's DAIS RFP and IVI) support the notion of *Plug&Play*. In these systems it is possible to define *predefined*, or *custom*, *Composition Monitoring Criteria*. Then, a client can subscribe to receive *notifications* when changes on the composition of certain devices happen. Systems that do not support the notion of *Plug&Play* (such as OPC, ODAS and RoMain) do not have concepts regarding *Composition Monitoring Criteria* (*Device Model Composition Monitoring Criteria* from the *Device Model Monitoring Criteria* package and *Device Item Composition Monitoring Criteria* from the *Device Item Monitoring Criteria* package).
- (iv) *Predefined versus Custom Monitoring Criteria.* Some systems (such as OMG's DAIS RFP, OPC, IVI and ODAS) allow client applications to define *custom datasets*, *trigger conditions* or *monitoring criteria* specific to certain device items. Other systems (such as RoMain) only allow clients working with *predefined datasets*, *trigger conditions* and *monitoring criteria*. Systems that do not allow clients to define *custom datasets*, *trigger conditions* or *monitoring criteria* specific to certain device items do not support the *Administer Dataset*, *Administer Trigger Condition* and *Administer Monitoring Criteria* functionalities.
- (v) *Public Monitoring Criteria versus Private Monitoring Criteria.* Some systems (such as IVI) allow clients to define only *public monitoring criteria*. Other systems (such as ODAS and RoMain) allow clients to define only *private monitoring criteria*. Other systems (such as OMG's DAIS RFP and OPC) allow clients to define both *monitoring criteria*.

Table 2 Generic DAS Concept Comparison

Generic DAS Concept	OMG's DAIS RFP	OPC (v2)	IVI	ODAS	RoMain
<i>Device Models</i>					
Single Device Model	✓	✗	✓	✗	✓
Complex Device Model	✓	✗	✓	✗	✓
Functional Device Model	✓	✗	✓	✗	✓
Measurement Point	✓	✗	✓	✓	✓
Measurement Type	✓	✓	✓	✓	✓
Phenomenon Type	✓	✓	✓	✓	✓
Phenomenon	✓	✓	✓	✓	✓
<i>Device Items</i>					
Single Device Item	✓	✓	✓	✓	✓
Complex Device Item	✓	✓	✓	✗	✓
Device Address	✓	✓	✓	✗	✓
Measurement Address	✓	✓	✓	✓	✓
<i>Device Model Monitoring Criteria</i>					
Device Model Composition Monitoring Criteria	✓	✗	✓	✗	✗
Device Model Event Monitoring Criteria	✓	✗	✓	✗	✗
Device Model Status Monitoring Criteria	✓	✗	✓	✗	✓
Device Model Time Trigger Condition	✓	✗	✓	✗	✓
Device Model Event Trigger Condition	✓	✗	✓	✗	✗
Device Model Dataset	✓	✗	✓	✗	✓
<i>Device Item Monitoring Criteria</i>					
Public Monitoring Criteria	✓	✓	✓	✗	✗
Private Monitoring Criteria	✓	✓	✗	✓	✓
Device Item Composition Monitoring Criteria	✓	✗	✓	✗	✗
Device Item Event Monitoring Criteria	✓	✓	✓	✗	✗
Device Item Status Monitoring Criteria	✓	✓	✓	✓	✗
Predefined Monitoring Criteria	✓	✗	✓	✗	✓
Device Item Time Trigger Condition	✓	✓	✓	✗	✗
Device Item Event Trigger Condition	✓	✓	✓	✓	✗
Predefined Trigger Condition	✓	✗	✓	✗	✗
Custom Dataset	✓	✓	✓	✓	✗
Predefined Dataset	✓	✗	✓	✗	✗
<i>Observations & Monitoring Reports</i>					
Measurement	✓	✓	✓	✓	✓
Category Observation	✓	✓	✓	✓	✓
Monitoring Report	✓	✓	✓	✓	✓

Table 3 Generic DAS Functionality Comparison

Generic DAS Functionality	OMG's DAIS RFP	OPC (v2)	IVI	ODAS	RoMain
<i>Discover</i>					
Discover Composition	✓	✓	✓	✓	✓
Discover Model	✓	✗	✓	✗	✓
Discover Datasets	✓	✓	✓	✓	✓
Discover Trigger Conditions	✓	✓	✓	✓	✓
Discover Monitoring Criteria	✓	✓	✓	✓	✓
<i>Define Data Access</i>					
Administer Dataset	✓	✓	✓	✓	✗
Administer Trigger Condition	✓	✓	✓	✓	✗
Administer Monitoring Criteria	✓	✓	✓	✓	✗
Administer Monitoring Criteria Subscription	✓	✓	✓	✓	✓
<i>Access Data</i>					
Access Observations	✓	✓	✓	✓	✓
Access Monitoring Reports	✓	✓	✓	✓	✓
<i>Notify Data Availability</i>					
Notify Data Availability	✓	✓	✓	✓	✗
<i>Upload Data</i>					
Upload Data	✓	✓	✓	✓	✓

8.4 Design of a New System

A generic system specification can be applied in the analysis phase of a particular system to better understand the problem and to easier specify the best solution, depending on the specific requirements of a particular system. A generic system specification can also be used as a starting point for the design of a particular system. As a result, the use of a generic system specification will save time and reduce the costs of the development of a particular system. In order to validate this hypothesis we developed a DAS for railway equipment based on our generic DAS specification. In this section we summarize the major results and conclusions from the development of this DAS.

8.4.1 Development of a DAS for Railway Equipment

The main objective of this development was to validate our generic DAS specification by means of an example. The development of this DAS also gives developers a case study on the development of a particular system based on a generic system specification. Additionally, as part of the development of the DAS for railway equipment, we implemented a generic library, independent from the context of railway equipment, that can be reused and/or extended for the implementation of another DAS based on our generic DAS specification.

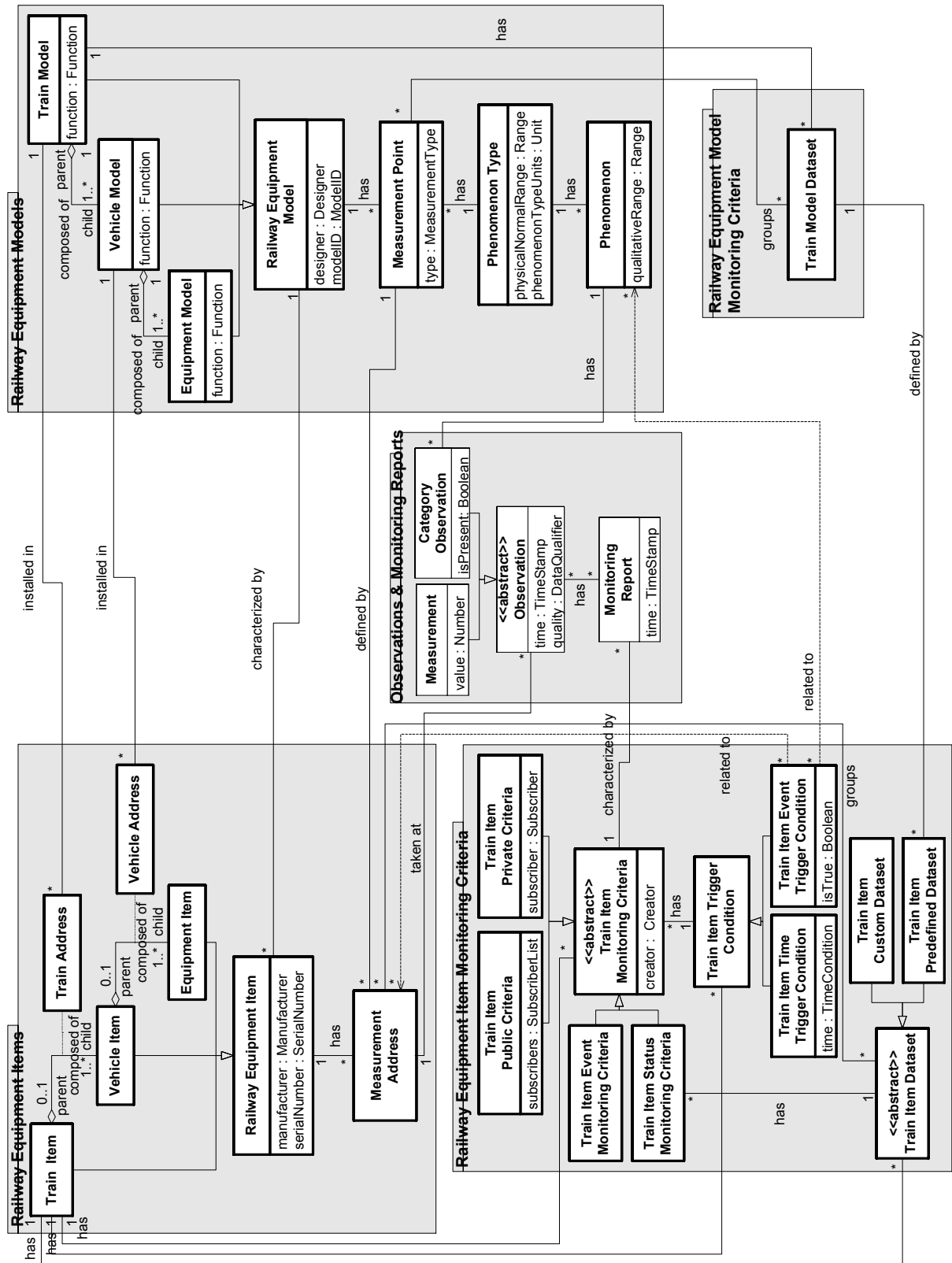


Figure 67 Railway Equipment DAS Conceptual Model

Although our generic DAS specification was partially obtained from the analysis of the RoMain system, this system is not compliant with the generic DAS specification. This is reasonable as our generic DAS specification comes also from the analysis of many DAS standards, the OMG's DAIS RFP, and some software patterns. The objective of the development of a DAS for railway equipment was to re-design the RoMain system by using our generic DAS specification. We considered the re-design of the RoMain system as a good and fast way – as we already have some knowledge of the domain of railway equipment - to obtain a first validation of our generic DAS specification.

Railway Equipment DAS Conceptual Model

We analyzed the generic DAS conceptual model to obtain a DAS conceptual model specialized in the context of railway equipment. This railway equipment DAS conceptual model is shown in Figure 67. We analyzed the generic concepts beginning with the packages with fewer dependencies continuing through the packages that have more dependencies:

1. We analyzed the *Device Models* package in the context of railway equipment. The result of this analysis was a partial conceptual model of the system that consists of a three-level hierarchy of device models: at the top level there are train models that are composed of vehicle models that are composed of equipment models. *Railway Equipment Model* is a specialization of *Device Model*; *Train Model* and *Vehicle Model* are specializations of *Complex Device Model*; and *Equipment Model* is a specialization of *Single Device Model* and *Functional Device Model*.
2. We analyzed the *Device Items* package in the context of railway equipment. From this analysis we included, in the conceptual model of the system, a three-level hierarchy of items: at the top level there are train items that are composed of vehicle items that are composed of equipment items, each of these items being characterized by its corresponding train, vehicle or equipment model. *Railway Equipment Item* is a specialization of *Device Item*; *Train Item* and *Vehicle Item* are specializations of *Complex Device Item*; and *Equipment Item* is a specialization of *Single Device Item*.
3. We analyzed the *Device Model Monitoring Criteria* package in the context of railway equipment. One of the requirements of the system was to be able to use datasets predefined in a model of train to record status monitoring reports of particular train items. Therefore, we included the concept of *Train Model Dataset*, which is a specialization of *Model Dataset* that groups measurement points corresponding to a train model or to one of its vehicle models or equipment models.
4. We analyzed the *Device Item Monitoring Criteria* package in the context of railway equipment. The requirements of the system were: (i) to enable the recording of status monitoring reports and event monitoring reports corresponding to a train item, such reports being triggered either by a time trigger condition or by an event trigger condition; (ii) to enable the definition of both *private* and *public* monitoring criteria; and (iii) to enable the definition of custom and predefined train datasets. Therefore, we introduced the concepts of: *Train Item Monitoring Criteria*, *Train Item Public Monitoring Criteria*, *Train Item Private Monitoring Criteria*, *Train Item Status Monitoring Criteria* and *Train Item Event Monitoring Criteria*, which are specializations of *Device*

Item Monitoring Criteria, Public Monitoring Criteria, Private Monitoring Criteria, Device Item Status Monitoring Criteria and Device Item Event Monitoring Criteria, respectively; *Train Item Trigger Condition, Train Item Time Trigger Condition and Train Item Event Trigger Condition*, which are specializations of *Device Item Trigger Condition, Device Item Time Trigger Condition and Device Item Event Trigger Condition*, respectively; and *Train Item Dataset, Train Custom Dataset and Train Predefined Dataset*, which are specializations of *Device Item Dataset, Custom Dataset and Predefined Dataset*, respectively.

- Finally, we analyzed the *Observations and Monitoring Reports* package in the context of railway equipment. The requirements of the system were that both measurements and category observations could be recorded. Additionally, status and event monitoring reports could also be recorded. Therefore, we introduced the concepts of *Observation, Measurement, Category Observation and Monitoring Report* to the conceptual model of the system.

Railway Equipment DAS Use Case Model

Once we had the railway equipment DAS conceptual model, we analyzed the generic use case model and generic elementary roles taking into account that the DAS system should enable: (i) the discovery of train, vehicle and equipment metadata by supervisors of trains; (ii) the definition, by administrators of the system, of *pull* and *push* based access to train data by supervisors of trains; (iii) the access of train data by supervisors of trains; and (iv) the upload of data to supervisors that are subscribed to certain monitoring criteria. Additionally, an architectural requirement of the system was to implement the DAS system using a three-tier architecture composed of a *Web Interface* to users of the system and a *Ground Station* in the middle tier responsible for the wireless communication with many *Train Gateways*, each of them on-board a particular train. In Figure 67, we present the resulting high-level system use cases.

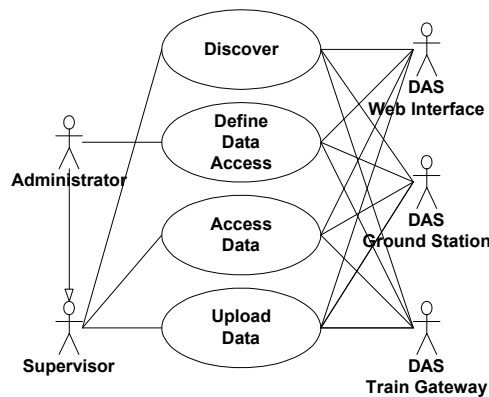


Figure 68 Railway Equipment DAS System Use Case Model

We designed the system iteratively starting from the use cases with fewer dependencies to the use cases that have more dependencies: first, the *Discover* use case; second, the *Define Data Access* use case; third, the *Access Data* use case; and finally, the *Upload Data* use case. For each of these use cases we analyzed the elementary roles that take part in the use case and we designed components that implement such elementary roles. Due to the architecture of the system, a role may

eventually be implemented as a combination of many components distributed in the three-tier architecture that intercommunicate to fulfill the role functionality. In any case, the elementary roles made it easier the identification and design of the required components.

Railway Equipment DAS Summary

The generic DAS specifications had a significant role in the design of the DAS railway equipment. We used the use case model to clearly specify the functionalities that the system should implement. We used the elementary roles to specify some of the components of the system. We used the conceptual model to better organize the data that will be used by the components of the system. The conceptual model also made it easier the design of some of these components of the system. As a result, we had an implementation where a significant amount of the designed classes represent concepts specified in the conceptual model and elementary roles specified in the use case model of the generic DAS specification. The development of this particular DAS demonstrated, by means of an industrial example, the usefulness of a generic system specification for the development of a particular system.

8.5 Summary

In this chapter we explained the applications of the external specification of a generic system. We showed that the most direct application of a generic system specification is for the writing of a RFP for a new standard. Another potential application of a generic system specification is for the evaluation of existing systems, standards or RFP responses. As an example, we showed a comparison of the OMG's DAIS RFP versus the most important DAS standards and the RoMain application. Finally, a generic system specification can be applied in the development of a particular system. This will significantly reduce the development costs of a specific system. We illustrated this by means of an example of development of a DAS for railway equipment based on our generic DAS specification. The development of this particular DAS for railway equipment validates the usefulness of our generic DAS specification for the development of particular DASs. Additionally, the development of this particular DAS for railway equipment provides developers with a case study on the development of particular systems based on generic system specifications.

9. Conclusion

9.1 Introduction

In this chapter we summarize the major findings and contributions from the actual work. We also point out some future work in this field.

9.2 Major Contributions

The major contributions of this thesis are:

- (i) *We provide an external specification of a generic architecture for DASs.* This generic DAS specification enables the comparison of existing DAS products and standards. Additionally, it provides the DAS developers that aim to develop a specific DAS with a starting point for the design of a specific DAS.
- (ii) *We propose patterns and techniques to facilitate the development, based on conceptual and role-based use case modeling, of external specifications of generic systems.* A generic system specification is independent of design choices and it can be applied to the development of many similar systems giving substantial benefits saving time and reducing the costs of the development of a specific system.
- (iii) *We provide a case study on the development, based on conceptual and role-based use case modeling, of external specifications of generic systems.* This case study demonstrates, by means of an industrial example, the advantages of conceptual and role-based use case modeling for the development of external specifications of generic systems.

We also contributed to the TCN standard as part of our work for the ROSIN European project.

9.3 Major Findings

We classify the findings into: findings regarding the conceptual model, findings regarding the role-based use case model, and findings regarding the development process.

9.3.1 Conceptual Model

We described the static concepts of the external specification of a generic system by means of a conceptual model. We found that conceptual modeling is a

useful technique to represent the static concepts that make up a system. The major contributions regarding the conceptual model are:

- (i) *We explain how to represent and identify device models and device items.* We represent a real world device as an instance of *Device Item*. We represent the type of a device, commonly known as its *model*, as an instance of *Device Model*. An instance of *Device Model* characterizes a set of instances of *Device Item*. An instance of *Device Item* is always characterized by an instance of *Device Model*. A GUID is assigned to each device model and device item.
- (ii) *We propose a new pattern: the “Model Composite” pattern.* This pattern is used to represent *part-whole* hierarchies of device models. This pattern is our own variation of the well-known *Composite* pattern.
- (iii) *We describe how to enable a Plug&Play functionality in a system.* We enable this functionality by means of the definition of composition monitoring criteria and the recording of composition monitoring reports.
- (iv) *We define the mappings between sampled and physical values.* We define concepts that allow the representation of complex mappings of sampled values into physical values.

9.3.2 Role-based use case Model

We used use cases to describe the functionalities of a system. We found that use case modeling is a useful technique to represent the dynamic behavior of a system. The major contributions regarding the use case model are:

- (i) *We propose to represent the system itself in the use cases.* In this way it is easier to define the role of the system on each use case and to find out the interfaces that the system has to provide to other participants in the use case in order to carry out this use case.
- (ii) *We propose the use of elementary roles rather than actors in the use cases.* This enables the specification of a generic system independently of architectural choices, requirements, QoS, and/or available technologies specific for a particular system. The resulting specification can, then, be reused in different implementations of similar systems by mapping roles into actors according to the requirements of a specific system.
- (iii) *We propose techniques to model the system behavior.* An external specification of a generic system must allow the specification of different system behaviors depending on specific execution constraints and types of control flow. We give some techniques to specify the interactions across use cases and the scenarios of use cases in such a way that the specification enables the implementation of systems with different execution constraints and types of control flow.
- (iv) *We present a useful pattern: the “Administrator-Manager” pattern.* This pattern concerns the administration of a particular entity X.

9.3.3 Development Process

In this thesis we propose patterns and techniques to develop, based on

conceptual and role-based use case modeling, external specifications of generic systems. The external specification of a generic system is obtained from the analysis of multiple systems and/or standards. This specification can be applied to the development of many particular systems. We strongly believe that this development process provides substantial benefits by saving time and reducing the costs of the development of a specific system. As a generic system specification can be applied to the development of many particular systems, the number of particular systems developed multiplies the benefits.

The drawback of this development process is the complexity of the design of an external specification of a generic system. This development process provides real benefits when designers have to develop many similar systems.

9.4 Future Work

In this thesis it was not possible to quantify the savings on the development costs of systems by the application of an external specification of a generic system. We would have had to apply this method to the development of many systems in various domains. The analysis of multiple systems and standards requires much time and effort, especially to non-experts on the domain. Additionally, it is rather difficult to measure and/or evaluate the savings. For these reasons, we considered the quantification of the savings as a future work out of this thesis. To quantify the savings on the development costs of systems by the application of our development process researchers would need to: (i) develop a method to measure objectively the savings; (ii) apply the same development process to several systems in various domains; and (iii) measure the savings and interpret the results.

Appendix A RoMain Java vs. RoMain XML

In this appendix we present and discuss a comparison of the RoMain Java prototype versus the RoMain XML prototype, under the following criteria:

- Communication Performance
- Client Interface
- Scalability
- Security
- Reliability
- Availability
- Evolvability

Communication Performance

In both prototypes, in order to compare the communication performance, we measured the response time to obtain, for each equipment, the current values of all the properties. A property of an equipment is an attribute that gives some information about it and its current state. In both prototypes the data source is a train network installed in a laboratory at the CAF facilities. The *Ground Station* and the clients are installed in the same machine at ICA. This is because we wanted to compare the communication performance of Java RMI versus HTTP in a single client-server communication. The overhead of the three-tier communication of the RoMain XML prototype is not taken into account. The real Internet is used as communication network. We used Java 1.1.7 to develop the Java RMI server and the performance clients for this evaluation. These clients use the *pull* mechanism to obtain the current values of the properties.

To obtain more reliable results we did one hundred essays of each measurement. We represent the average and standard deviation of the results into a chart diagram. The X-axis represents the different equipment, with the number of properties in brakes. The Y-axis represents the time in milliseconds. A column bar represents the average time in milliseconds to obtain all the properties of an equipment. A line appended to a column represents the standard deviation of the measurements.

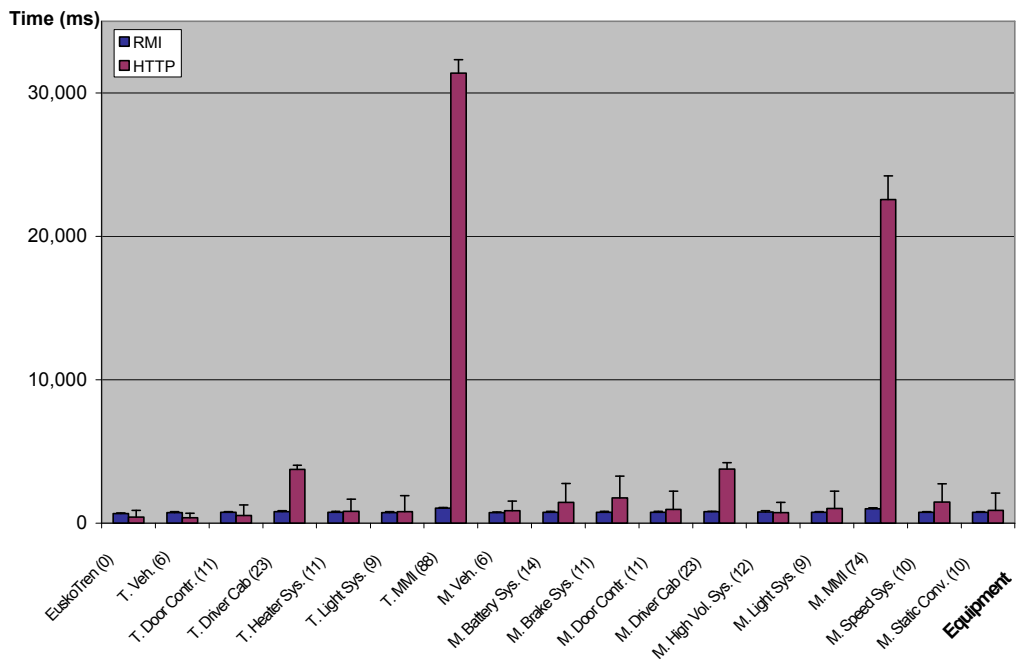
The first evaluation consisted in measuring the time to obtain, for each equipment, one update of all its properties. The results of this evaluation are shown in

Figure 69a. These results demonstrated that, with one update of the properties, the performance of Java RMI is generally much better than the performance of HTTP. Only in the case of an equipment with few properties (that is the case of *Euskotren*) the performance of HTTP is slightly better than the performance of Java RMI. The more properties an equipment has, the longer the difference is between the performances of Java RMI against HTTP. In the case of an equipment with a large quantity of properties (this is the case of *T.MMI* and *M.MMI*) the performance of Java RMI is substantially higher than the performance of HTTP. Therefore, we can conclude that the RoMain Java prototype demonstrated a better communication performance than the RoMain XML prototype.

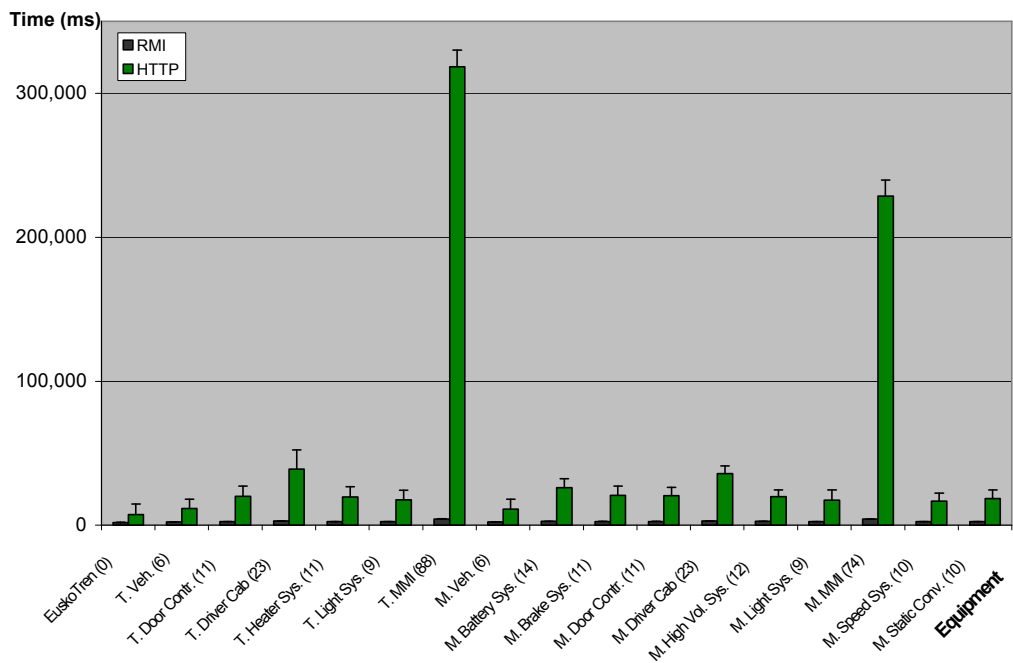
In the RoMain XML prototype, a client application has to perform a new request for update the properties of an equipment. Therefore, we presume that the cost of a second and subsequent updates will be the same as the cost of the first update. However, in the RoMain Java prototype, in the first update a client application initially obtains a local reference (a *proxy* object) to the remote object, and then it invokes a remote call to this object to obtain all the properties of an equipment. In successive updates of the properties, it no longer needs to obtain the *proxy* object. It has only to invoke a remote call to obtain all the properties again. Thus, in Java RMI the cost of a second and subsequent updates should be lower than the cost of the first update. In order to corroborate this we performed a second evaluation. This evaluation consisted in measuring the time to obtain, for each equipment, ten updates of all its properties. The results of this evaluation are shown in Figure 69b.

These results demonstrated that with ten updates of the properties, the performance of Java RMI is always much better than the performance of HTTP. Even more, the differences between the performance of Java RMI and HTTP have increased substantially. This is because the time to obtain ten updates of all properties in the HTTP based prototype is, on average, about ten times longer than the time to obtain the first update. This comparison is shown in Figure 70a. However, in the Java RMI based prototype, the time to obtain ten updates of all properties is, on average, less than four times longer than the time to obtain the first update. This comparison is shown in Figure 70b. The reason for these differences between Java RMI and HTTP is that in Java RMI the cost of successive updates is lower than the cost of the first update. However, in HTTP the time to obtain “n” updates corresponds approximately to “n” times the time to obtain the first update.

Therefore, we can state that the difference between the performances of Java RMI against HTTP will increase as we increase the numbers of updates. In conclusion, Java RMI will perform even better than HTTP when numerous updates are required.

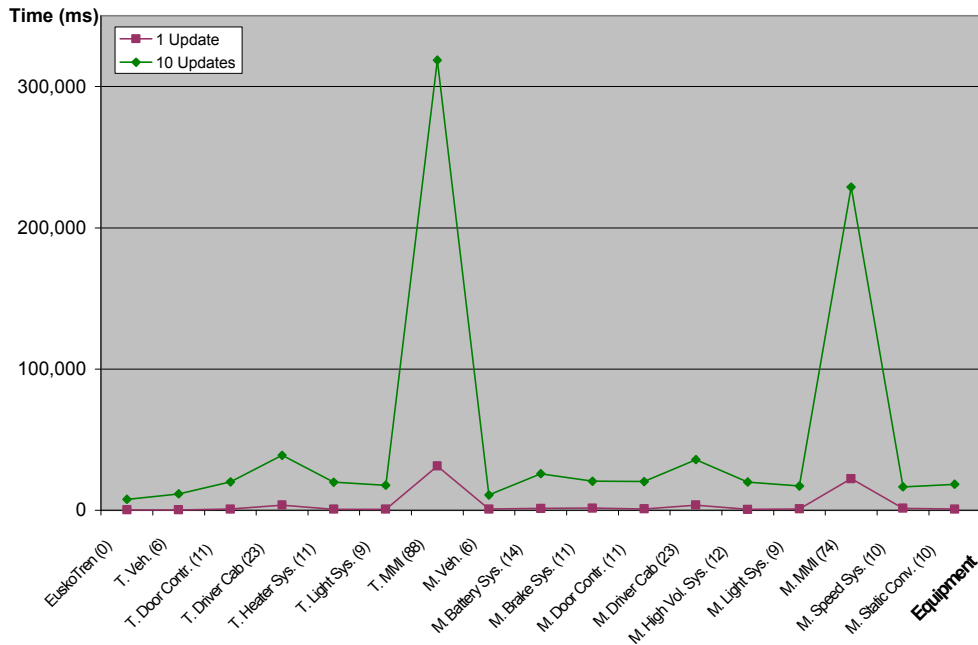


(a) One Update

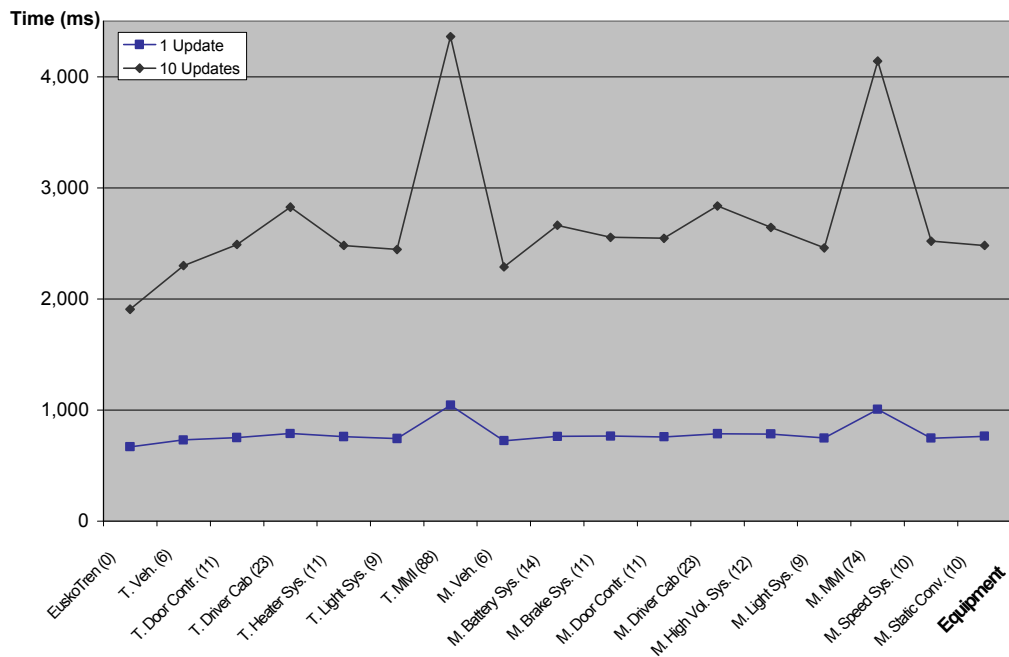


(b) Ten Updates

Figure 69 Communication Performance Comparison



(a) RoMain XML



(b) RoMain Java

Figure 70 One Update vs. Ten Updates Comparison

Client Interface

In both prototypes, clients use Internet browsers to access the current state of the properties of a train. But the mechanisms to present this data to clients are very different.

In the RoMain Java prototype, the client interface is implemented as a Java applet. Java applets run on many standard browsers (such as *Netscape Communicator* and *Microsoft Internet Explorer*) without any pre-installation. We made use of a graphical library, called *Swing*, to present graphically the current state of the properties of a train. This library is an extension to the standard Java API, and it may not be locally installed with a specific version of a browser. In this case a client would need to pre-install locally this library before running the Java applet. Java applets and *Swing* offer a very powerful set of graphical components to design graphical interfaces. In order to save wireless communication bandwidth, the Java applet is downloaded from a *Ground Station* and not from the *Train Gateway*. Once the Java applet is running on the client browser, the client-server communication is established directly between the client Java applet and the *Train Gateway*. A security restriction of Java applets in Java 1.1 allows a Java applet to establish communication only with the server it was downloaded from. To overcome this problem we used special classes that allow us to give special privileges (such as the communication with any server) to Java applets. The problem is that the classes we used are specific to the *Netscape Communicator* browser. Hence, the RoMain Java prototype only works using this browser. Similar mechanisms exist for *Microsoft Internet Explorer* but there is no cross browser compatibility. The problem of the security restriction of Java applets is currently solved with Java 1.2. This new version of Java provides a policy-based, easily configurable, fine-grained access control.

In the RoMain XML prototype, we used XML direct browsing to generate on the client side the view with an XSL file. XSL has demonstrated to be a powerful means to combine and transform XML files into another XML file or another presentation format (such as HTML). XML/XSL direct browsing capabilities offer an elegant manner to generate an XML presentation directly on the client side. Unfortunately, direct browsing of XML files is currently only possible by using *Microsoft IE5* as an Internet browser. A possible solution to enable clients with browsers without XML direct browsing capabilities is to transform, on the server side, the XML data into plain HTML. In this way the data could be displayed on any standard browser.

Using Java applets instead of XML direct browsing for the client view brings significant benefits. A Java applet is an application running on a web browser. Therefore, a Java applet can make possible the implementation of complex features such as receiving notifications of server side updates of data. However, the cost of downloading a Java applet is always significantly higher than the cost of direct browsing of XML. Additionally, direct browsing of XML enables the efficient switching between different client views at the client side, without performing a new communication. An important advantage of Java applets is that they can be displayed on many browsers, whereas direct browsing of XML is only possible, today, using *Microsoft IE5*.

Scalability

In both prototypes the bottleneck is obviously the *Train Gateway*, which is implemented as a Java RMI server. Java RMI servers can support efficiently a few hundred simultaneous clients, but they are not scalable for large systems. Additionally, the performance slows down as the number of clients increase. In the case of our application -the remote monitoring of railway equipment - we do not expect a large amount of simultaneous clients. Therefore, the limitation on the scalability of Java RMI servers does not cause any real problems.

In the RoMain Java prototype, the communication is established directly between client Java applets and the Java RMI server. If scalability had been an important requirement of our system, we would have designed the application in a different way. One solution to designing scalable systems using Java RMI is to implement a middle tier with many application servers, each of them handling requests from many client Java applets. Each application server establishes a single Java RMI communication with the Java RMI server on the *Train Gateway*. A load-balancing manager would be responsible for assigning, at runtime, an application server with enough resources to a client Java applet. In Java RMI there is not a pre-defined load balancing manager, but it should not be too much complicated to develop such a system.

In the RoMain XML prototype, it is easier to scale the system because it is implemented in a three-tier architecture. In the middle tier an application server handles requests from Internet clients and dispatches the requests to a data server installed on the same machine as the Java RMI server. If we need a very scalable system, we have only to deploy the data server on many machines. Then, a load-balancing manager on the application server would decide, at run-time, which data server to contact to dispatch the request to the Java RMI server on the *Train Gateway*.

Security

In both prototypes standard mechanisms for authentication of users can be easily implemented. In the RoMain Java prototype, the authentication of the user can be done by a Java applet that is downloaded from the *Ground Station* before downloading the monitoring Java applets. Once a user has been authenticated, the user is allowed to download the monitoring Java applets. In the RoMain XML prototype, the authentication mechanisms can be implemented in the *Ground Station* as well, as part of the application server.

Both Java RMI and HTTP support the Secure Socket Layer (SSL) for encryption of the information before transmitting it over the Internet. SSL provides simple but efficient mechanisms to encrypt the information in ways that render hacking difficult. Despite other more complex and efficient mechanisms for security, SSL has become the de-facto standard over the Internet for encryption of data.

Reliability

Both prototypes depend heavily on the network. Therefore, the reliability of such systems depends very much on the reliability of the network itself. In our experiment we used the Internet as a communication network. Due to the nature of

the Internet and its fluctuant bandwidth, this network cannot be considered reliable.

Analyzing the results of the evaluations we noticed that the differences between the same measurements in the RoMain XML prototype are larger than the differences between the same measurements in the RoMain Java prototype. This is due to the fact that the RoMain XML prototype requires more use of the network than the RoMain Java prototype, to obtain the same information. Additionally, as the RoMain XML prototype is implemented in a three-tier architecture it makes even more use of the network than the RoMain Java prototype. Effectively, two HTTP calls are needed to obtain the current values of the properties of an equipment and send these values back to a client application. However, in the RoMain Java prototype an update of the properties of an equipment needs only a Java RMI remote call.

In conclusion, we can state that the RoMain Java prototype is more reliable than the RoMain XML prototype.

Availability

The RoMain XML prototype proposes a three-tier architecture, whereas the RoMain Java prototype proposes a two-tier architecture. Therefore, in the RoMain XML prototype there are more agents that interact within the system than in the RoMain Java prototype. Effectively, in the RoMain XML prototype we have a data server that obtains data from a DAS in the *Train Gateway*, a document server in the *Ground Station* and a client running *Microsoft IE5*.

In the RoMain Java prototype, there is only one Java RMI server that obtains data from a DAS in the *Train Gateway* and the Java RMI client running within a Java applet in the client's browser. The probability of failure in the RoMain XML prototype is increased by the inclusion of a middle tier.

In conclusion, the RoMain Java prototype has a greater availability than the RoMain XML prototype.

Evolvability

This is the point that makes the RoMain XML prototype really interesting. The RoMain XML prototype is based on XML. XML is easy to create, parse and transform. It is very simple to integrate data coming from different sources, using XSL style sheets. In the RoMain XML prototype the integration of data coming from different trains is easily done at the middle tier. The RoMain XML prototype offers an architecture that allows a high flexibility in evolution.

The RoMain Java prototype provides a good example of client-server computing. However, it is more complicated to integrate data coming from different trains and to process the data of a train in different ways as is done in the current prototype - just to give some examples.

In conclusion, the RoMain XML prototype brings a higher evolvability than the RoMain Java prototype. A solution to increase the evolvability of the RoMain Java prototype is to re-design this system with a three-tier architecture. As in the RoMain XML prototype, a middle tier would be responsible for establishing

connections and retrieving data from different trains. Furthermore, it would be responsible for integrating this data and sending it back to a client.

Summary

The RoMain Java prototype demonstrated a better communication performance than the RoMain XML prototype. The prototype based on Java RMI has the additional advantage that it can use server side *push* technology. This can optimize data communication by only transmitting data that has been changed on the monitored system. The main drawback of this prototype is that clients within firewalls cannot efficiently use it. It is possible to use Java RMI over firewalls by using the concept of HTTP tunneling, but the efficiency of the communication slows down considerably. The HTTP based prototype enables clients to use the prototype even within firewalls. However, the communication performance is low. Eventually, this performance can be improved by compressing the XML formatted data (using an algorithm as ZIP) on the server and by uncompressing it again on a client. The client interface offered by the RoMain Java prototype enables the implementation of complex features such as receiving notifications of server side updates of data. The client interface of the RoMain XML prototype is much more simple but it enables the switching between different client views at the client side. The scalability and reliability of both prototypes are acceptable for systems like RoMain. The reliability and availability of the RoMain Java prototype are higher than in the RoMain XML prototype. This is mainly due to the fact that the RoMain XML prototype proposes a three-tier architecture, whereas the RoMain Java prototype proposes a, much simpler, 2-tier architecture. For the same reasons, the RoMain XML prototype is also much more flexible in evolution than the RoMain Java prototype.

The conclusion of these experiments is that when a high performance remote monitoring system is required, Java and Java RMI are the right technologies. If flexibility in evolution is a strong requirement, a three-tier system, which is rather simple to develop using technologies such as HTTP and XML, may be a better choice.

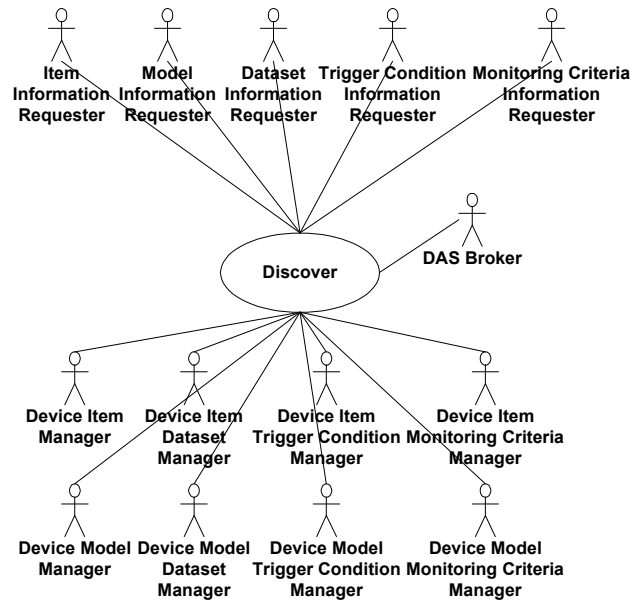
Appendix B DAS Role-based Use Cases

In this appendix we present the role-based use cases corresponding to a data acquisition system:

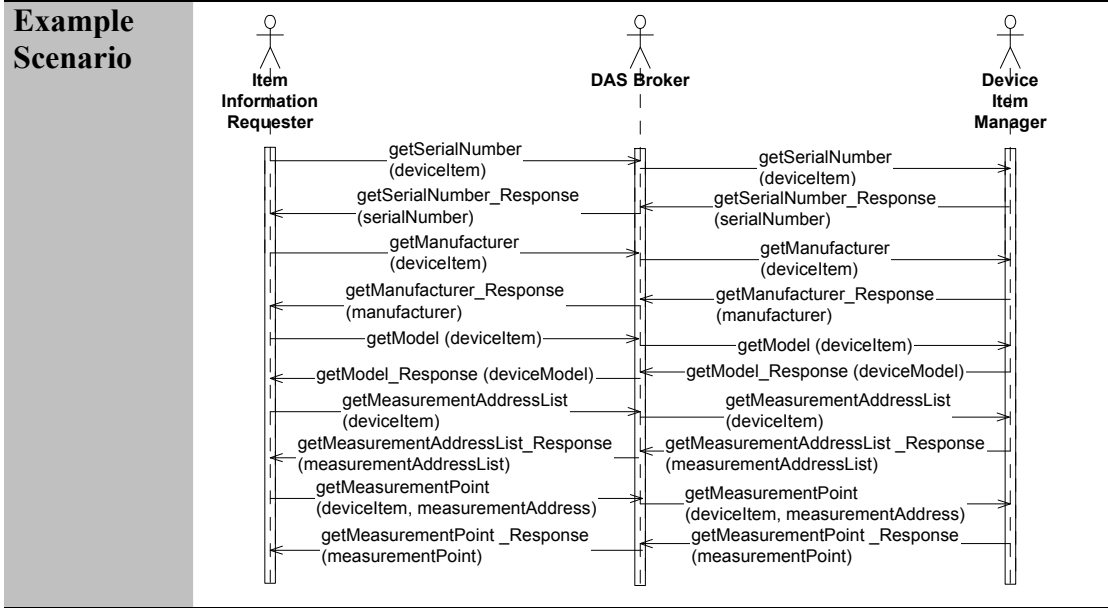
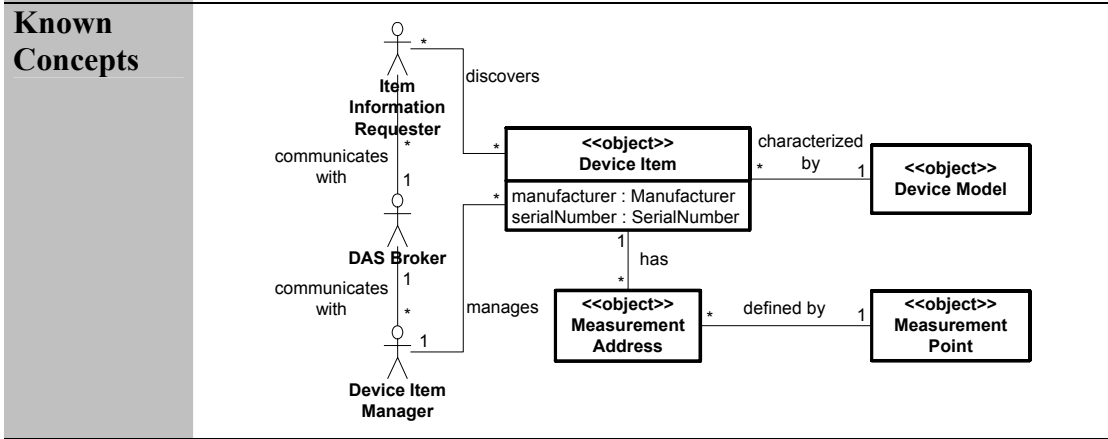
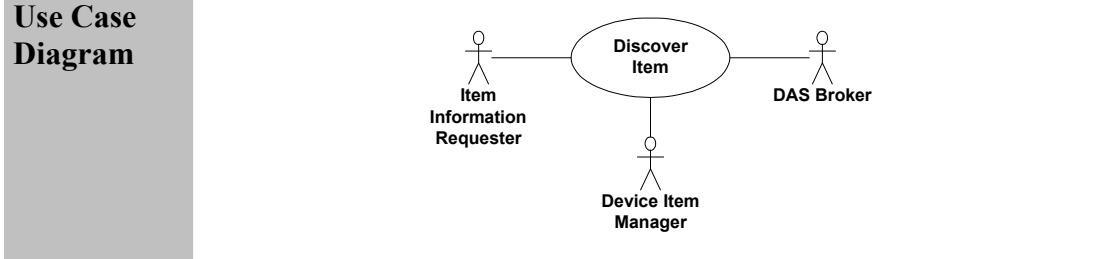
Error! No table of contents entries found.

Use Case	Discover
Roles	<i>Item Information Requester, Model Information Requester, Dataset Information Requester, Trigger Condition Information Requester, Monitoring Criteria Information Requester, Device Model Manager, Device Item Manager, Device Item Dataset Manager, Device Model Dataset Manager, DAS Broker</i>
Type	Primary and essential.
Description	An <i>Item Information Requester</i> obtains, by request, the information specific to device items; their current composition; and their device models. A <i>Model Information Requester</i> obtains, by request, the information specific to device models. A <i>Dataset Information Requester</i> obtains, by request, the information specific to datasets defined in device items. A <i>Trigger Condition Information Requester</i> obtains, by request, the information specific to trigger conditions defined in device items. A <i>Monitoring Criteria Information Requester</i> obtains, by request, the information specific to monitoring criteria defined in device items.
Pre-conditions	The <i>Device Item Manager</i> can provide the information specific to a device item, its composition information and its device model. The <i>Device Model Manager</i> can provide the information specific to the corresponding device model and its composition. The <i>Device Model Dataset Manager</i> can provide the datasets predefined in the device model, and the <i>Device Item Dataset Manager</i> can provide the datasets defined in the device item. The <i>Device Model Trigger Condition Manager</i> can provide the trigger conditions predefined in the device model, and the <i>Device Item Trigger Condition Manager</i> can provide the trigger conditions defined in the device item. The <i>Device Model Monitoring Criteria Manager</i> can provide the monitoring criteria predefined in the device model, and the <i>Device Item Monitoring Criteria Manager</i> can provide the monitoring criteria defined in the device item.
Post-conditions	An <i>Item Information Requester</i> has discovered the information specific to device items; their current composition; and their device models. A <i>Model Information Requester</i> has discovered the information specific to device models. A <i>Dataset Information Requester</i> has discovered the information specific to datasets defined in device items. A <i>Trigger Condition Information Requester</i> has discovered the information specific to trigger conditions defined in device items. A <i>Monitoring Criteria Information Requester</i> has discovered the information specific to monitoring criteria defined in device items.

Use Case Diagram



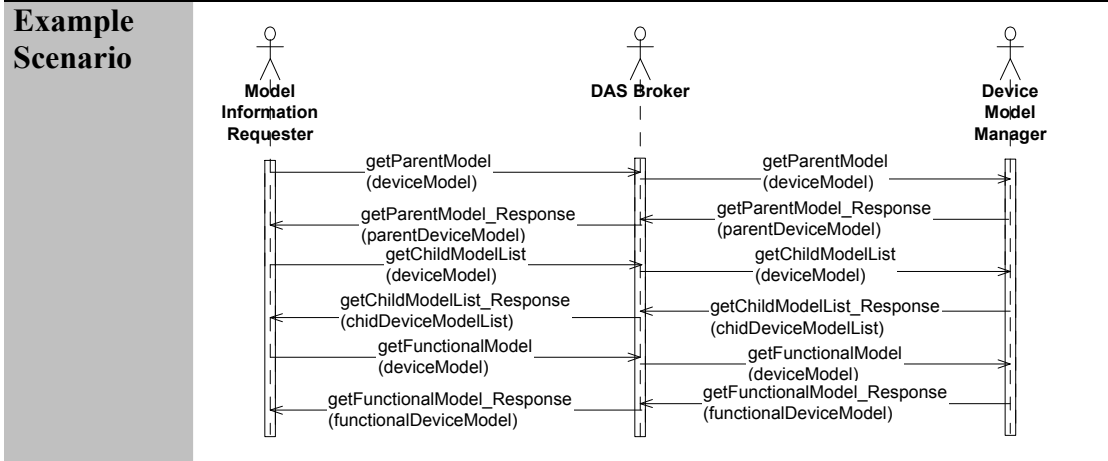
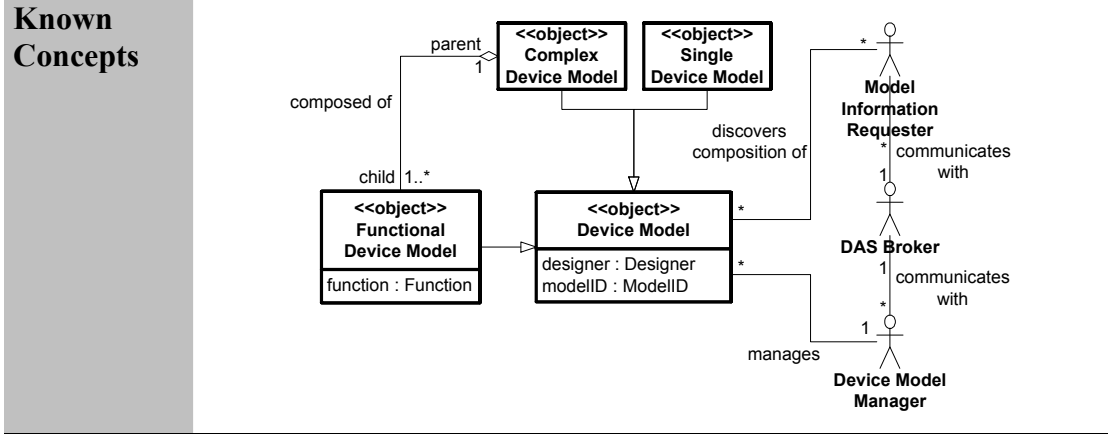
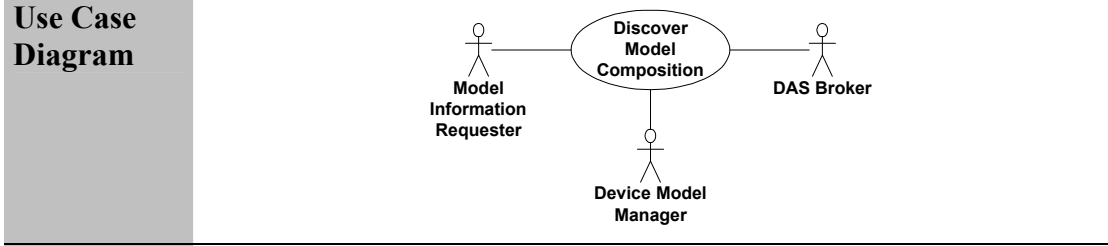
Use Case	Discover Item
Roles	<i>Item Information Requester, Device Item Manager, DAS Broker.</i>
Type	Primary and essential.
Description	An <i>Item Information Requester</i> obtains, by request, the information (serial number, manufacturer, device model, list of measurement addresses, and the measurement points associated to each measurement address) specific to a device item.
Pre-conditions	The <i>Device Item Manager</i> can provide the information specific to a device item.
Post-conditions	An <i>Item Information Requester</i> has discovered the information specific to a device item.



Use Case	Discover Item Composition
Roles	<i>Item Information Requester, Device Item Manager, DAS Broker.</i>
Type	Primary and essential.
Description	An <i>Item Information Requester</i> obtains, by request, the current composition (parent device item, child device items, and functional device model) of a device item.
Pre-conditions	The <i>Device Item Manager</i> can provide the device item composition information.
Post-conditions	An <i>Item Information Requester</i> has discovered the current composition of a device item. The <i>Item Information Requester</i> has obtained a list of device items that compounds a complex device item.
Use Case Diagram	<p>The Use Case Diagram shows three actors: Item Information Requester, Device Item Manager, and DAS Broker. The Item Information Requester is connected to the Discover Item Composition use case. The Device Item Manager is also connected to the Discover Item Composition use case. The DAS Broker is connected to the Discover Item Composition use case.</p>
Known Concepts	<p>The Known Concepts Diagram illustrates the relationships between various objects and actors. The Item Information Requester (actor) communicates with the DAS Broker (actor) and the Device Item Manager (actor). The DAS Broker communicates with the Device Item Manager. The Device Item Manager manages the Device Item (object). The Device Item is composed of Single Device Item (object) and Complex Device Item (object). The Device Item is composed of Device Address (object) and Functional Device Model (object). The Device Address is installed in the Functional Device Model. The Device Item has attributes: manufacturer: Manufacturer, serialNumber: SerialNumber. The Functional Device Model has a function: Function.</p>
Example Scenario	<p>The Example Scenario Sequence Diagram shows the interaction between the Item Information Requester, DAS Broker, and Device Item Manager. The Item Information Requester sends a request to the DAS Broker for getParentItem (deviceItem). The DAS Broker sends a request to the Device Item Manager for getParentItem (deviceItem). The Device Item Manager returns a response to the DAS Broker for getParentItem_Response (parentDeviceItem). The DAS Broker returns a response to the Item Information Requester for getParentItem_Response (parentDeviceItem). The Item Information Requester sends a request to the DAS Broker for getChildItemList (deviceItem). The DAS Broker sends a request to the Device Item Manager for getChildItemList (deviceItem). The Device Item Manager returns a response to the DAS Broker for getChildItemList_Response (chidDeviceItemList). The DAS Broker returns a response to the Item Information Requester for getChildItemList_Response (chidDeviceItemList). The Item Information Requester sends a request to the DAS Broker for getFunctionalModel (deviceItem). The DAS Broker sends a request to the Device Item Manager for getFunctionalModel (deviceItem). The Device Item Manager returns a response to the DAS Broker for getFunctionalModel_Response (functionalDeviceModel). The DAS Broker returns a response to the Item Information Requester for getFunctionalModel_Response (functionalDeviceModel).</p>

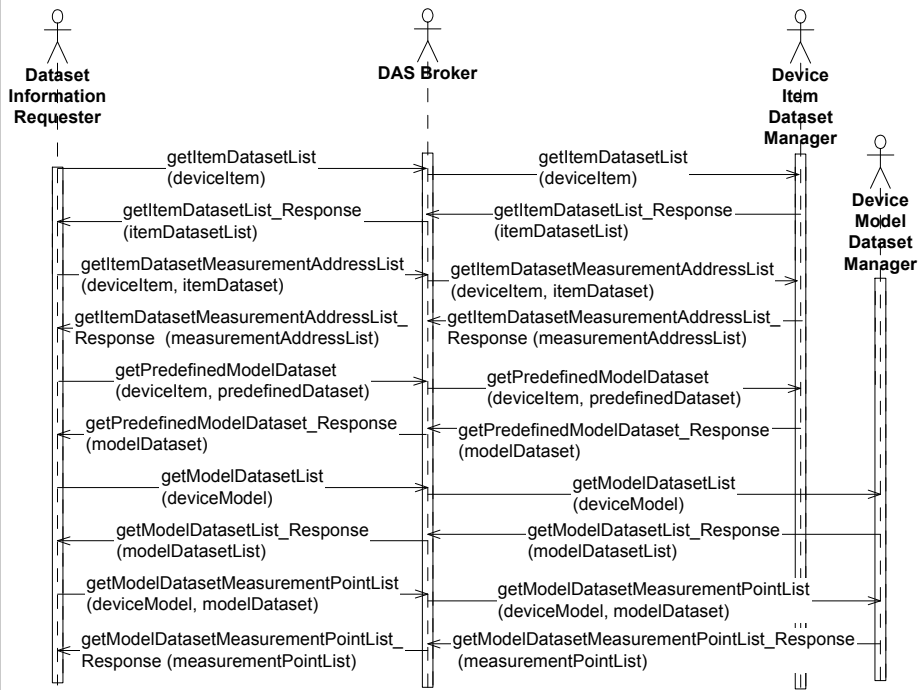
Use Case	Discover Model
Roles	<i>Model Information Requester, Device Model Manager, DAS Broker.</i>
Type	Primary and essential.
Description	A <i>Model Information Requester</i> obtains, by request, the information (modelID, designer, measurement points, measurement type and phenomenon type associated with a measurement point, and the phenomenon list of a phenomenon type) of a device model.
Pre-conditions	The <i>Device Model Manager</i> can provide the information specific to a device model.
Post-conditions	A <i>Model Information Requester</i> has discovered the device model and its associated information.
Use Case Diagram	<p>The Use Case Diagram shows three actors: Model Information Requester, Device Model Manager, and DAS Broker. The Model Information Requester is connected to the Discover Model use case. The Device Model Manager is also connected to the Discover Model use case. The DAS Broker is connected to the Discover Model use case.</p>
Known Concepts	<p>The Known Concepts Diagram illustrates the relationships between various objects. The Model Information Requester (actor) discovers multiple Device Model objects. The Device Model Manager (actor) manages multiple Device Model objects and communicates with the DAS Broker (actor). The DAS Broker (actor) communicates with the Device Model Manager (actor). The Device Model object has a designer (Designer) and a modelID (ModelID). The Device Model object has multiple Measurement Point objects. The Measurement Point object has a Measurement Type object. The Measurement Point object has a Phenomenon Type object. The Measurement Type object has a sampleRange (Range), physicalRange (Range), mapping (MappingPolicy), and measurementUnits (Unit). The Phenomenon Type object has a physicalNormalRange (Range) and phenomenonTypeUnits (Unit). The Phenomenon object has a qualitativeRange (Range).</p>
Example Scenario	<p>The Sequence Diagram shows the interaction between the Model Information Requester, DAS Broker, and Device Model Manager. The Model Information Requester sends a request to the DAS Broker to get the modelID of a device model. The DAS Broker sends a request to the Device Model Manager to get the modelID of a device model. The Device Model Manager returns the modelID to the DAS Broker. The DAS Broker returns the modelID to the Model Information Requester. The Model Information Requester sends a request to the DAS Broker to get the designer of a device model. The DAS Broker sends a request to the Device Model Manager to get the designer of a device model. The Device Model Manager returns the designer to the DAS Broker. The DAS Broker returns the designer to the Model Information Requester. The Model Information Requester sends a request to the DAS Broker to get the measurement point list of a device model. The DAS Broker sends a request to the Device Model Manager to get the measurement point list of a device model. The Device Model Manager returns the measurement point list to the DAS Broker. The DAS Broker returns the measurement point list to the Model Information Requester. The Model Information Requester sends a request to the DAS Broker to get the phenomenon type of a device model and measurement point. The DAS Broker sends a request to the Device Model Manager to get the phenomenon type of a device model and measurement point. The Device Model Manager returns the phenomenon type to the DAS Broker. The DAS Broker returns the phenomenon type to the Model Information Requester. The Model Information Requester sends a request to the DAS Broker to get the phenomenon list of a device model and phenomenon type. The DAS Broker sends a request to the Device Model Manager to get the phenomenon list of a device model and phenomenon type. The Device Model Manager returns the phenomenon list to the DAS Broker. The DAS Broker returns the phenomenon list to the Model Information Requester. The Model Information Requester sends a request to the DAS Broker to get the measurement type of a device model and measurement point. The DAS Broker sends a request to the Device Model Manager to get the measurement type of a device model and measurement point. The Device Model Manager returns the measurement type to the DAS Broker. The DAS Broker returns the measurement type to the Model Information Requester.</p>

Use Case	Discover Model Composition
Roles	<i>Model Information Requester, Device Model Manager, DAS Broker.</i>
Type	Primary and essential.
Description	A <i>Model Information Requester</i> obtains, by request, the composition of an instance of <i>Device Model</i> .
Pre-conditions	The <i>Device Model Manager</i> can provide the device model composition information.
Post-conditions	A <i>Model Information Requester</i> has discovered the composition of a device model. The <i>Model Information Requester</i> has obtained a list of device models that compounds a complex device model.

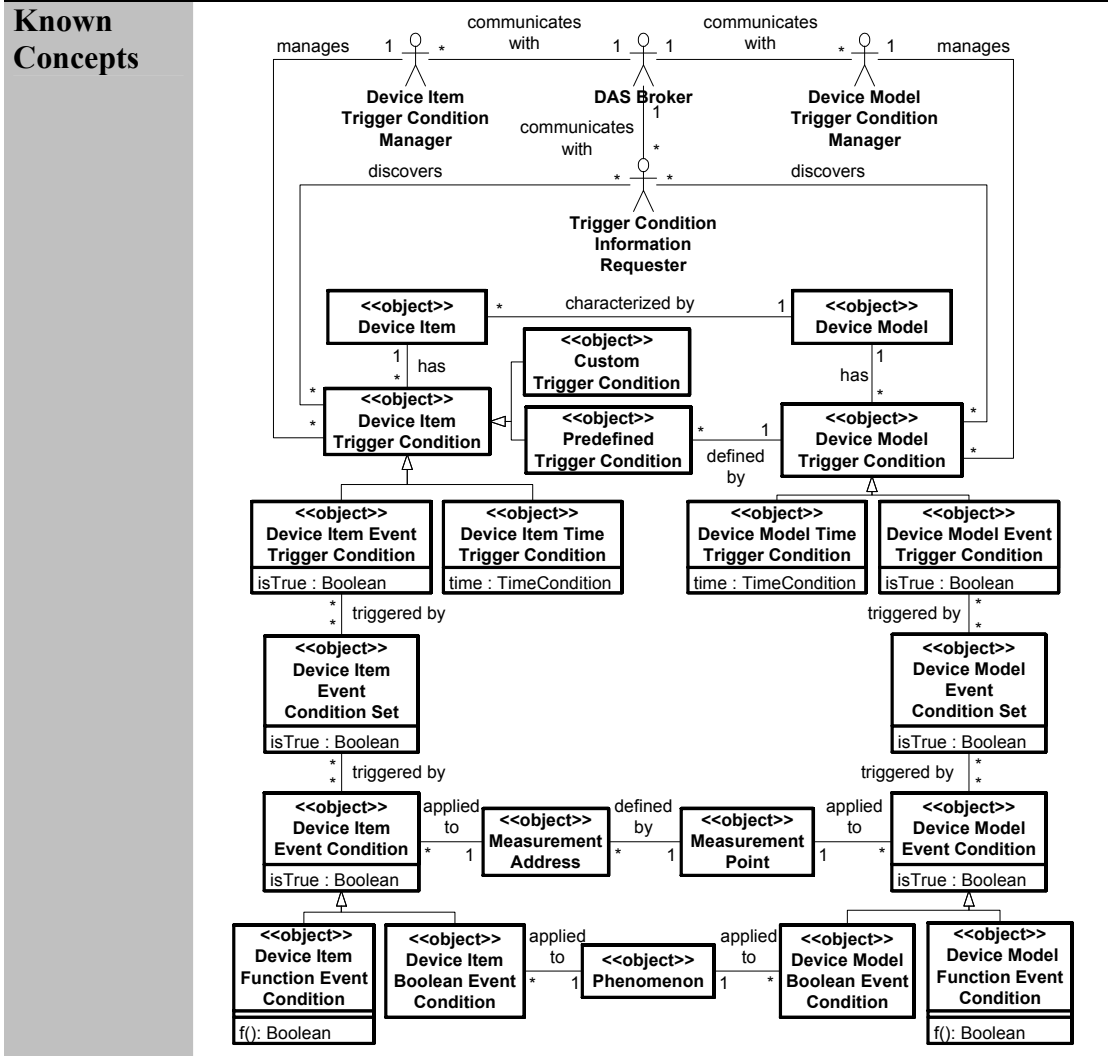
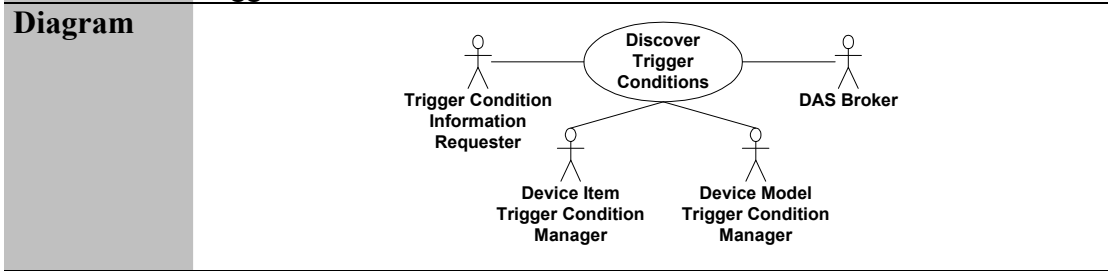


Use Case	Discover Datasets
Roles	<i>Dataset Information Requester, Device Item Dataset Manager, Device Model Dataset Manager, DAS Broker</i>
Type	Primary and essential
Description	A <i>Dataset Information Requester</i> obtains, by request, the datasets defined on a device item.
Pre-conditions	The <i>Device Item Dataset Manager</i> can provide the datasets defined in a device item. The <i>Device Model Dataset Manager</i> can provide the datasets predefined in a device model.
Post-conditions	A <i>Dataset Information Requester</i> has discovered the datasets of a device item.
Diagram	<p>The diagram shows a central use case 'Discover Datasets' connected to four actors: 'Dataset Information Requester', 'DAS Broker', 'Device Item Dataset Manager', and 'Device Model Dataset Manager'. The 'Dataset Information Requester' and 'DAS Broker' are connected to the use case, while 'Device Item Dataset Manager' and 'Device Model Dataset Manager' are connected to the use case via lines that branch from the use case.</p>
Known Concepts	<p>The class diagram illustrates the relationships between various objects and actors. Actors include 'Dataset Information Requester', 'DAS Broker', 'Device Item Dataset Manager', and 'Device Model Dataset Manager'. Objects include 'Device Item', 'Device Model', 'Measurement Address', 'Measurement Point', 'Custom Dataset', 'Predefined Dataset', 'Device Item Dataset', and 'Device Model Dataset'. Relationships are as follows: 'Device Item' (1) has 'Measurement Address' (*); 'Device Model' (1) has 'Measurement Point' (*); 'Device Item Dataset' (*) groups 'Device Item' (1) and 'Device Model Dataset' (*); 'Device Model Dataset' (*) groups 'Device Model' (1) and 'Device Item Dataset' (*); 'Device Item Dataset' (*) manages 'Device Item Dataset' (*); 'Device Model Dataset' (*) manages 'Device Model Dataset' (*); 'Dataset Information Requester' (*) communicates with 'DAS Broker' (1); 'DAS Broker' (1) communicates with 'Device Item Dataset Manager' (*); 'DAS Broker' (1) communicates with 'Device Model Dataset Manager' (*); 'Dataset Information Requester' (*) discovers 'Device Item Dataset' (*); 'Dataset Information Requester' (*) discovers 'Device Model Dataset' (*); 'Device Item Dataset' (*) is defined by 'Custom Dataset' (*) and 'Predefined Dataset' (*); 'Device Model Dataset' (*) is defined by 'Measurement Point' (*); 'Device Item' (*) characterizes 'Device Model' (1); 'Measurement Address' (*) is defined by 'Measurement Point' (*).</p>

Example Scenario



Use Case	Discover Trigger Conditions
Roles	<i>Trigger Condition Information Requester, Device Item Trigger Condition Manager, Device Model Trigger Condition Manager, DAS Broker.</i>
Type	Primary and essential.
Description	A <i>Trigger Condition Information Requester</i> obtains, by request, the trigger conditions defined on a device item.
Pre-conditions	The <i>Device Item Trigger Condition Manager</i> can provide the trigger conditions defined in a device item. The <i>Device Model Trigger Condition Manager</i> can provide the trigger conditions predefined in a device model.
Post-conditions	A <i>Trigger Condition Information Requester</i> has discovered the trigger conditions of a device item.



Example Scenario

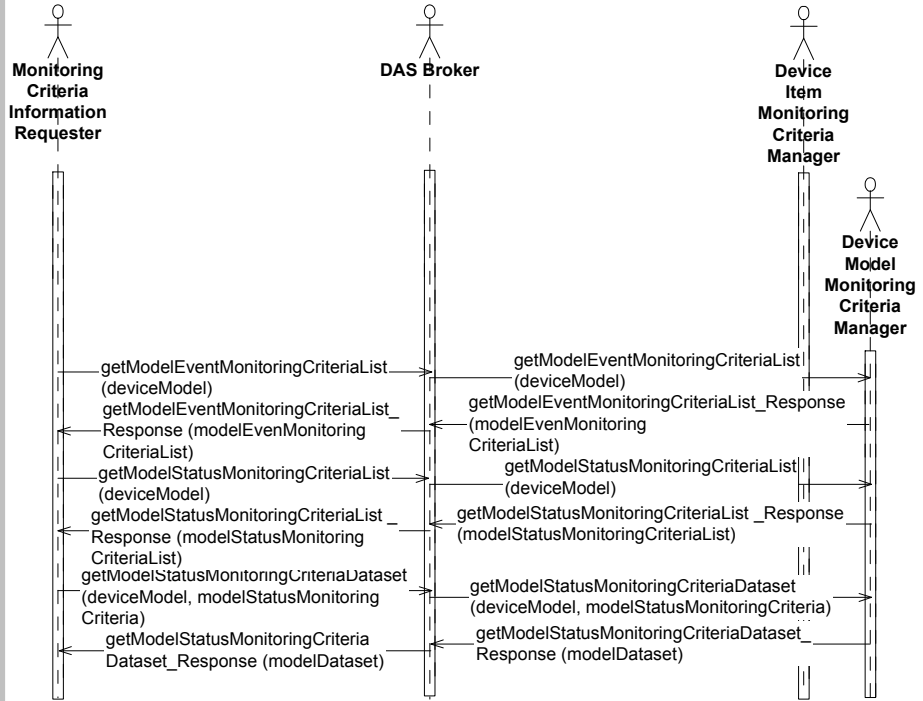


Use Case	Discover Monitoring Criteria
Roles	<i>Monitoring Criteria Information Requester, Custom Monitoring Criteria Manager, Predefined Monitoring Criteria Manager, DAS Broker.</i>
Type	Primary and essential.
Description	A <i>Monitoring Criteria Information Requester</i> obtains, by request, the monitoring criteria defined on a device item. A <i>Monitoring Criteria Information Requester</i> not being the creator of such monitoring criteria can discover only monitoring criteria defined as <i>public</i> . Monitoring criteria predefined on the corresponding device model are considered <i>public</i> and therefore they are accessible for any <i>Monitoring Criteria Information Requester</i> .
Pre-conditions	The <i>Device Item Monitoring Criteria Manager</i> can provide the monitoring criteria defined in a device item. The <i>Device Model Monitoring Criteria Manager</i> can provide the monitoring criteria predefined in a device model.
Post-conditions	A <i>Monitoring Criteria Information Requester</i> has discovered the monitoring criteria of a device item.
Diagram	
Known Concepts	

Example Scenario (1/2)



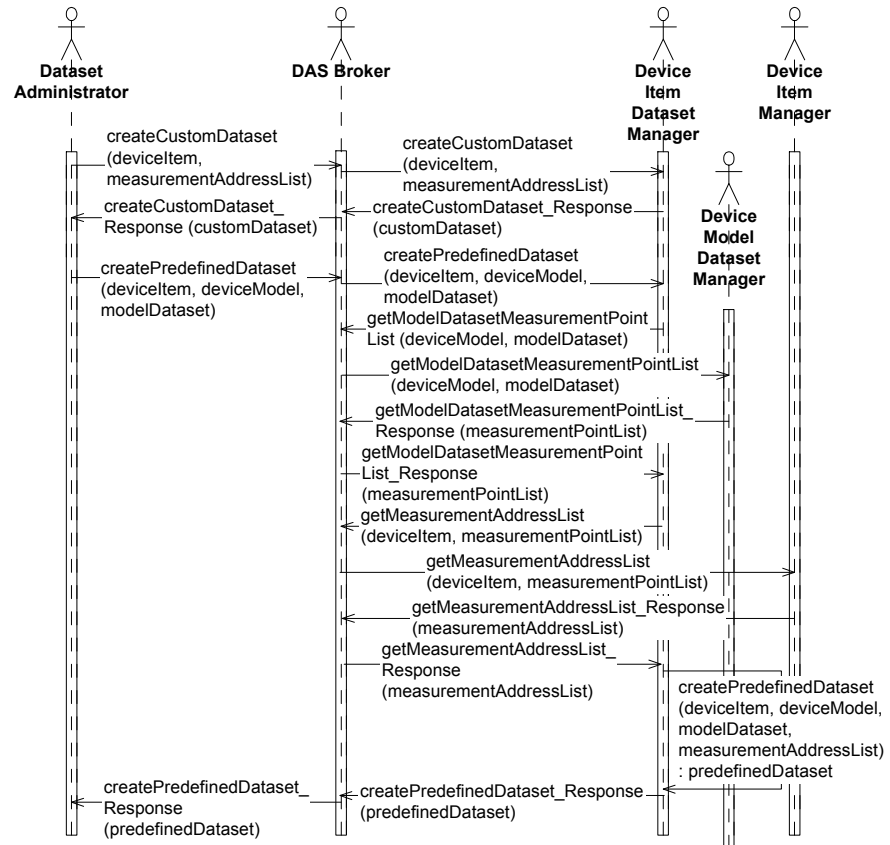
Example Scenario (2/2)



Use Case	Define Data Access
Roles	<i>Dataset Administrator, Device Item Dataset Manager, Device Model Dataset Manager, Trigger Condition Administrator, Device Item Trigger Condition Manager, Device Model Trigger Condition Manager, Monitoring Criteria Administrator, Device Item Monitoring Criteria Manager, Device Model Monitoring Criteria Manager, Monitoring Criteria Subscription Administrator, Device Item Monitoring Criteria Subscription Manager, DAS Broker.</i>
Type	Primary and essential.
Description	<i>Dataset Administrators</i> administer datasets for a device item. <i>Trigger Condition Administrators</i> administer trigger conditions for a device item. <i>Monitoring Criteria Administrators</i> administer monitoring criteria for a device item. <i>Monitoring Criteria Subscription Administrators</i> administer monitoring criteria. We used the term <i>administer</i> as a generic term to refer to <i>create, modify</i> and <i>remove</i> .
Pre-conditions	The device items have been previously discovered.
Post-conditions	Datasets, trigger conditions, monitoring criteria and monitoring criteria subscriptions have been administered.
Diagram	<p>The diagram shows a central use case 'Define Data Access' connected to 13 actors. The actors are: Dataset Administrator, Trigger Condition Administrator, Monitoring Criteria Administrator, Monitoring Criteria Subscription Administrator, Device Item Manager, DAS Broker, Device Item Dataset Manager, Device Item Trigger Condition Manager, Device Item Monitoring Criteria Manager, Device Item Monitoring Criteria Subscription Manager, Device Model Dataset Manager, Device Model Trigger Condition Manager, and Device Model Monitoring Criteria Manager.</p>

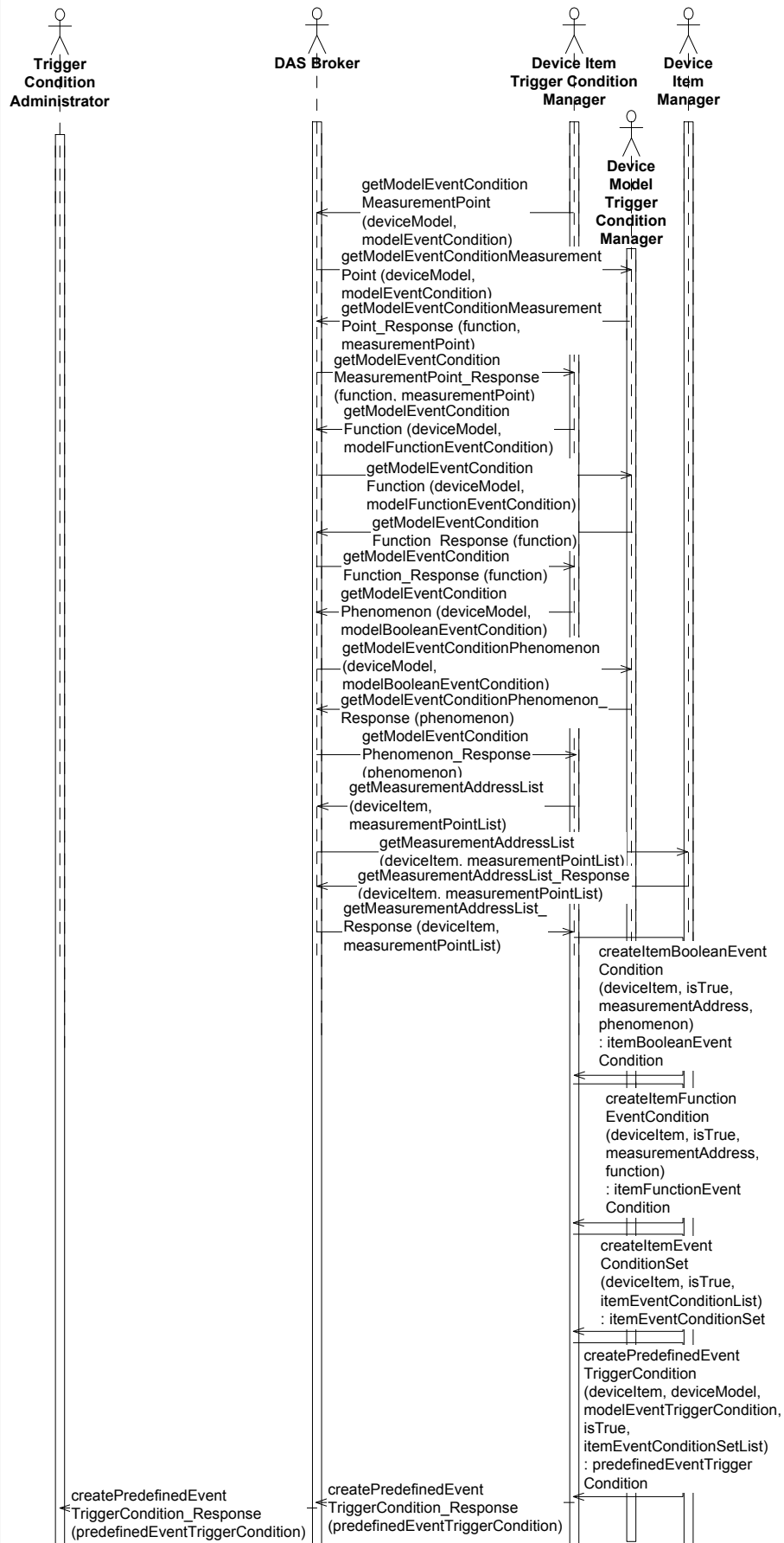
Use Case	Administer Dataset
Roles	<i>Dataset Administrator, Device Item Dataset Manager, Device Model Dataset Manager, Device Item Manager, DAS Broker.</i>
Type	Primary and essential.
Description	A <i>Dataset Administrator</i> administers (creates, modifies or removes) a dataset of a device item. A dataset consists of a set of measurement points to observe. <i>Dataset Administrators</i> can define an entirely new dataset for a device item (custom datasets), or define a dataset for a device item from a dataset predefined on a device model (predefined datasets). All <i>Datasets</i> are public.
Pre-conditions	The <i>Dataset Administrator</i> has discovered a device item and its corresponding device model. The <i>Device Model Dataset Manager</i> can provide the datasets predefined in the device model, and the <i>Device Item Dataset Manager</i> can provide the datasets already defined in a device item.
Post-conditions	A dataset of a device item has been administered.
Diagram	<p>The diagram shows a central use case 'Administer Dataset' connected to two actors: 'Dataset Administrator' and 'DAS Broker'. Below this, three actors are shown: 'Device Item Dataset Manager', 'Device Model Dataset Manager', and 'Device Item Manager'. Lines connect 'Dataset Administrator' to 'Device Item Dataset Manager' and 'Device Model Dataset Manager', and 'DAS Broker' to 'Device Item Manager'.</p>
Known Concepts	<p>The class diagram illustrates the relationships between various objects and actors. Objects include: <ul style="list-style-type: none"> Device Item (multiplicity *) Device Model (multiplicity 1) Measurement Address (multiplicity *) Measurement Point (multiplicity *) Custom Dataset (multiplicity *) Predefined Dataset (multiplicity *) Device Item Dataset (multiplicity *) Device Model Dataset (multiplicity *) Actors include: <ul style="list-style-type: none"> Dataset Administrator (multiplicity 1) Device Item Dataset Manager (multiplicity 1) DAS Broker (multiplicity 1) Device Model Dataset Manager (multiplicity 1) Device Item Manager (multiplicity 1) Relationships: <ul style="list-style-type: none"> Device Item characterizes Device Model (1 to *). Device Item has Measurement Address (1 to *). Measurement Address is defined by Measurement Point (* to 1). Measurement Point has Device Model (1 to *). Device Model has Device Model Dataset (1 to *). Device Model Dataset is defined by Predefined Dataset (* to 1). Device Item Dataset is defined by Custom Dataset (* to *). Device Item Dataset groups Measurement Address (* to *). Device Model Dataset groups Measurement Point (* to *). Dataset Administrator administers Device Item Dataset (1 to *). Dataset Administrator communicates with DAS Broker (1 to *). Device Item Dataset Manager manages Device Item Dataset (1 to *). Device Item Dataset Manager communicates with DAS Broker (1 to *). DAS Broker communicates with Device Model Dataset Manager (1 to *). Device Model Dataset Manager manages Device Model Dataset (* to 1). Device Item Manager manages Device Item Dataset (1 to *). Device Item Manager communicates with DAS Broker (1 to *). </p>

Example Scenario



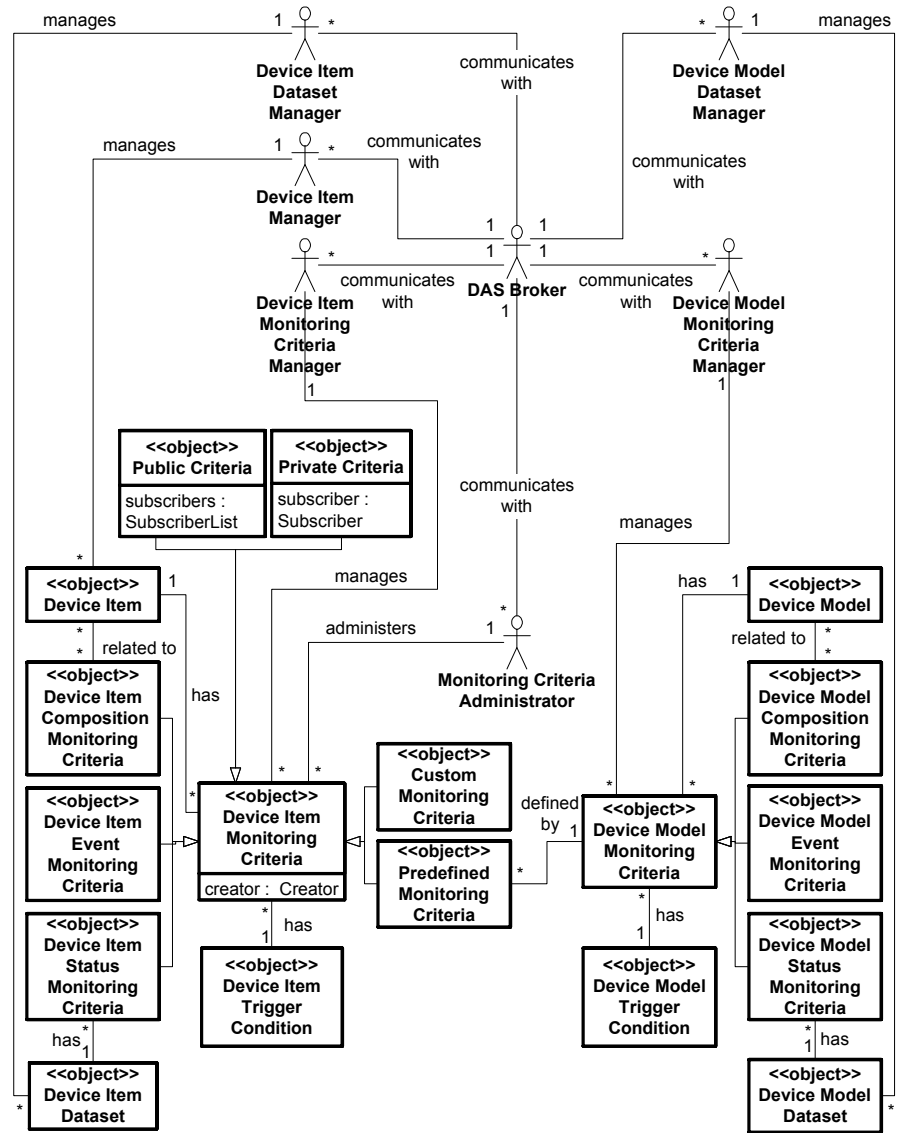
Use Case	Administer Trigger Condition
Roles	<i>Trigger Condition Administrator, Device Item Trigger Condition Manager, Device Model Trigger Condition Manager, Device Item Manager, DAS Broker.</i>
Type	Primary and essential
Description	A <i>Trigger Condition Administrator</i> administers (creates, modifies or removes) a trigger condition of a device item. A trigger condition can be based on time or based on an event. A <i>Trigger Condition Administrator</i> can define an entirely new trigger condition for a device item (custom trigger condition), or define a trigger condition from a trigger condition predefined on a device model (predefined trigger condition). All <i>Trigger Conditions</i> are public.
Pre-conditions	The <i>Trigger Condition Administrator</i> has discovered a device item and its corresponding device model. The <i>Device Model Trigger Condition Manager</i> can provide the trigger conditions predefined in the device model, and the <i>Device Item Trigger Condition Manager</i> can provide the datasets already defined in a device item.
Post-conditions	A trigger condition of a device item has been administered.
Diagram	<pre> graph TD UC((Administer Trigger Condition)) TCA[Trigger Condition Administrator] --- UC DASB[DAS Broker] --- UC UC --- DITCM[Device Item Trigger Condition Manager] UC --- DMTCM[Device Model Trigger Condition Manager] UC --- DIM[Device Item Manager] </pre> <p>The diagram is a UML Use Case Diagram. At the top center is an oval representing the use case 'Administer Trigger Condition'. To its left is an actor 'Trigger Condition Administrator' connected to the use case by a horizontal line. To its right is an actor 'DAS Broker' also connected by a horizontal line. Below the use case, three lines branch out to three actors: 'Device Item Trigger Condition Manager' on the left, 'Device Model Trigger Condition Manager' in the center, and 'Device Item Manager' on the right.</p>

Example Scenario (2/2)

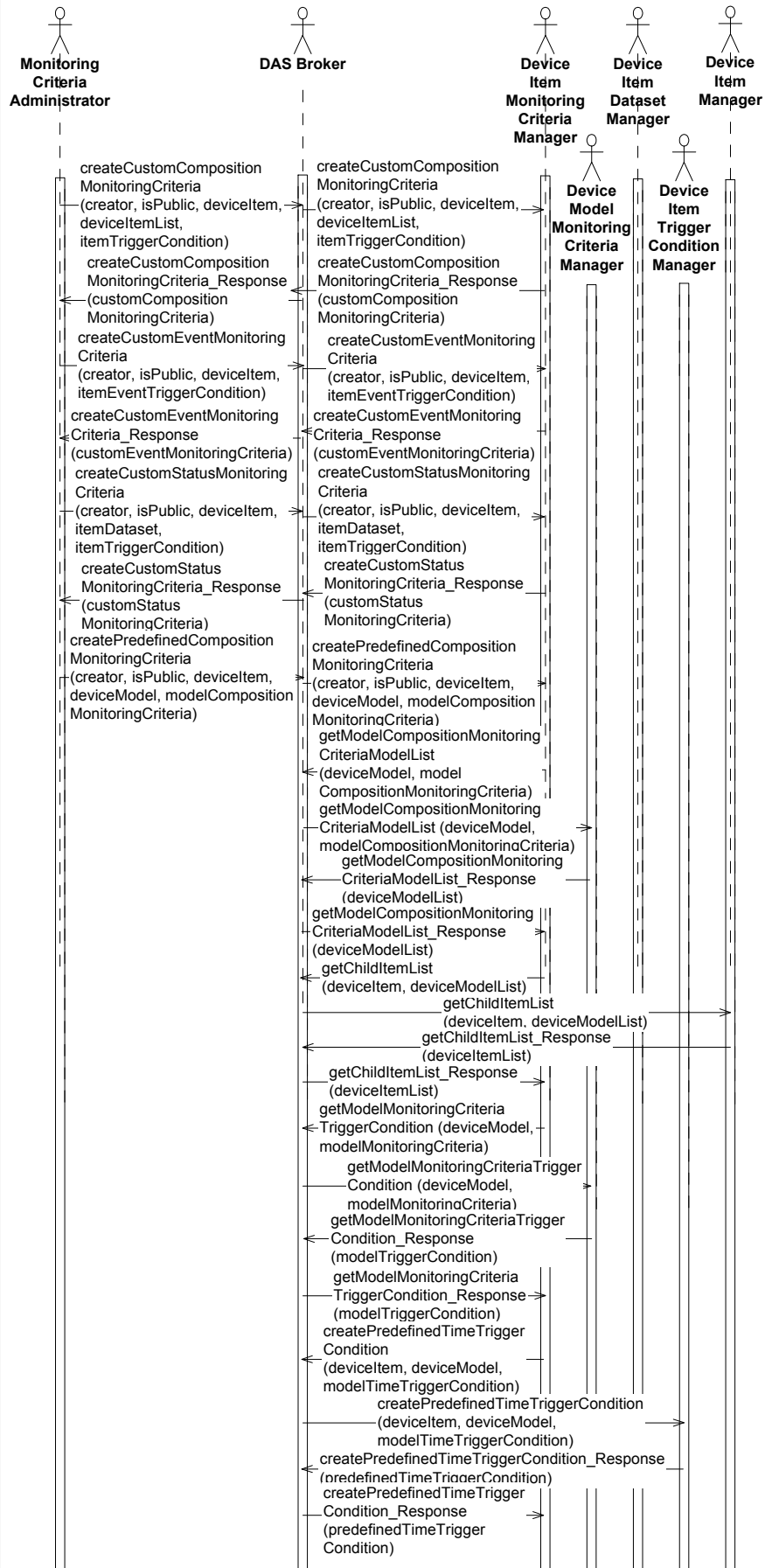


Use Case	Administer Monitoring Criteria
Roles	<i>Monitoring Criteria Administrator, Device Item Monitoring Criteria Manager, Device Model Monitoring Criteria Manager, Device Item Dataset Manager, Device Item Trigger Condition Manager, Device Item Manager, DAS Broker.</i>
Type	Primary and essential
Description	A <i>Monitoring Criteria Administrator</i> administers (creates, modifies or removes) monitoring criteria of a device item. Monitoring criteria allow for the recording of the status of a system at a specific time, the occurrence of an event or the recording of the change on the composition of a device item. <i>Monitoring Criteria Administrators</i> can define entirely new monitoring criteria for a device item, such monitoring criteria being called <i>custom</i> , or define monitoring criteria from monitoring criteria predefined on a device model, such monitoring criteria being called <i>predefined</i> .
Pre-conditions	The <i>Monitoring Criteria Administrator</i> has discovered a device item. The datasets and the trigger conditions required have been previously defined.
Post-conditions	Monitoring criteria of a device item have been administered.
Diagram	<pre> graph TD subgraph Actors D1M[Device Item Manager] D1DM[Device Item Dataset Manager] D1TM[Device Item Trigger Condition Manager] MCA[Monitoring Criteria Administrator] D1MM[Device Item Monitoring Criteria Manager] D2MM[Device Model Monitoring Criteria Manager] DASB[DAS Broker] end subgraph UseCases AMC((Administer Monitoring Criteria)) end D1M --- AMC D1DM --- AMC D1TM --- AMC MCA --- AMC DASB --- AMC D1MM --- AMC D2MM --- AMC </pre> <p>The diagram is a UML Use Case Diagram for the 'Administer Monitoring Criteria' use case. The central use case is represented by an oval labeled 'Administer Monitoring Criteria'. It is connected to seven actor roles, each represented by a stick figure: 'Device Item Manager', 'Device Item Dataset Manager', 'Device Item Trigger Condition Manager', 'Monitoring Criteria Administrator', 'DAS Broker', 'Device Item Monitoring Criteria Manager', and 'Device Model Monitoring Criteria Manager'. All connections are simple lines, indicating that each actor has a role in the use case.</p>

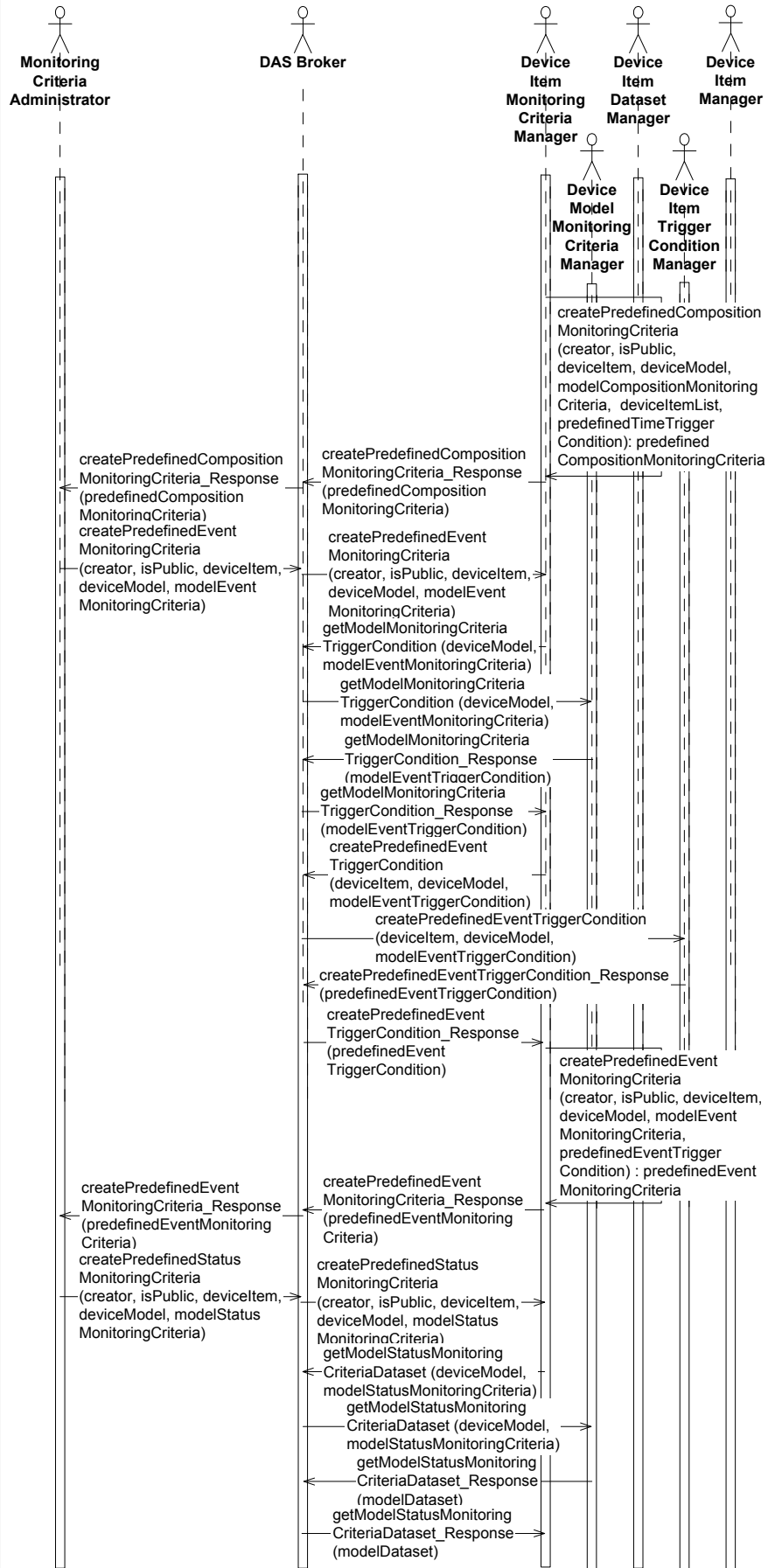
Known Concepts



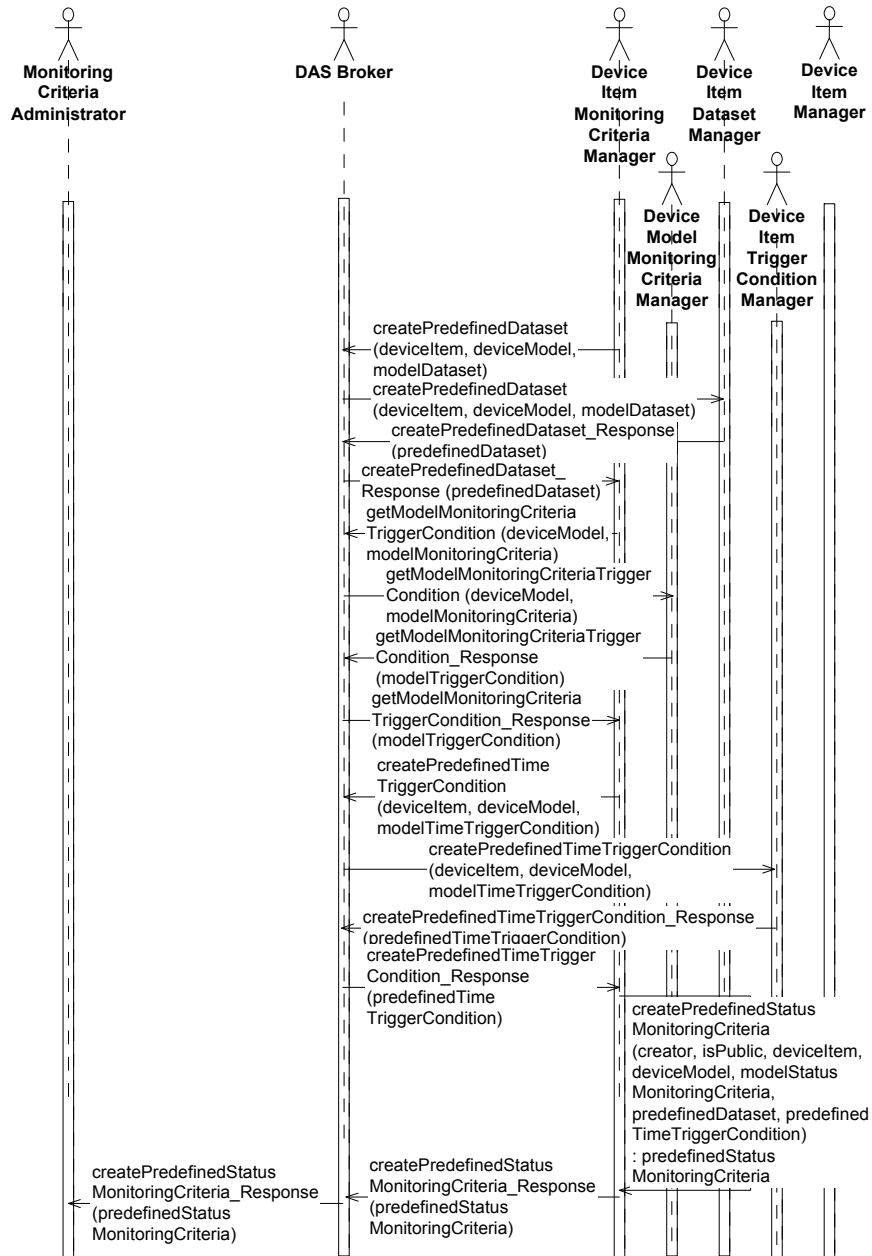
Example Scenario (1/3)



Example Scenario (2/3)



Example Scenario (3/3)

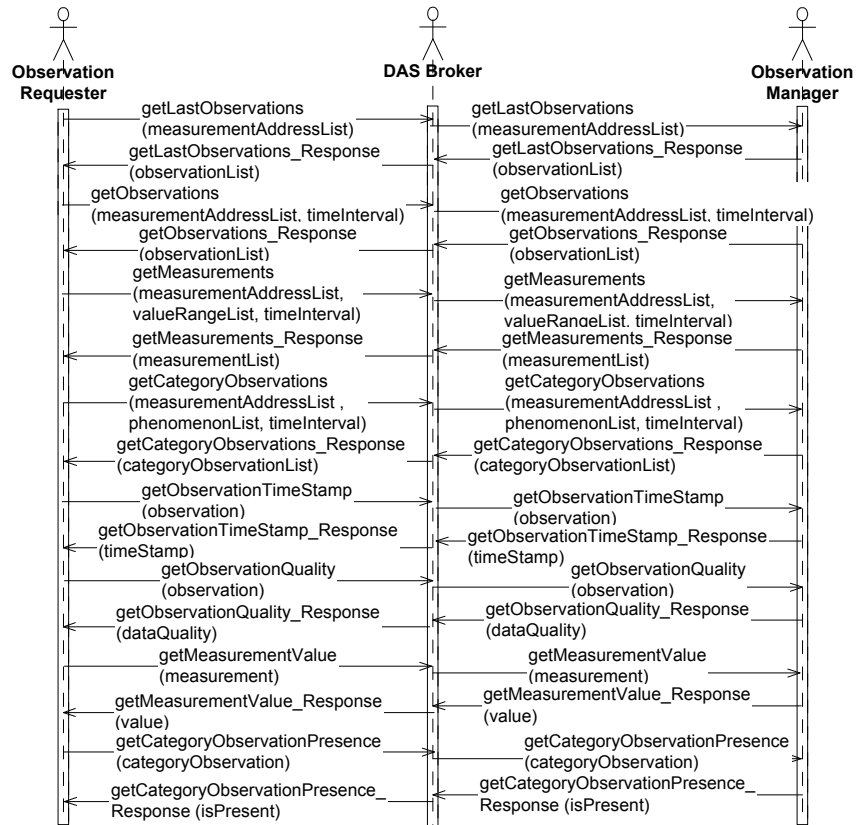


Use Case	Administer Monitoring Criteria Subscription
Roles	<i>Monitoring Criteria Subscription Administrator, Device Item Monitoring Criteria Manager, Device Item Monitoring Criteria Subscription Manager, DAS Broker.</i>
Type	Primary and essential
Description	A <i>Monitoring Criteria Subscription Administrator</i> administers (creates, modifies or removes) subscriptions of interest on certain monitoring criteria. The subscriber can chose between being automatically uploaded with monitoring reports corresponding to such monitoring criteria when available, or receiving a notification of the availability of monitoring reports corresponding to subscribed monitoring criteria.
Pre-conditions	The monitoring criteria of a device item have been previously defined. The <i>Monitoring Criteria Subscription Administrator</i> has discovered the device item and its associated monitoring criteria.
Post-conditions	The <i>Monitoring Criteria Subscription Administrator</i> has administered certain monitoring criteria.
Diagram	
Known Concepts	

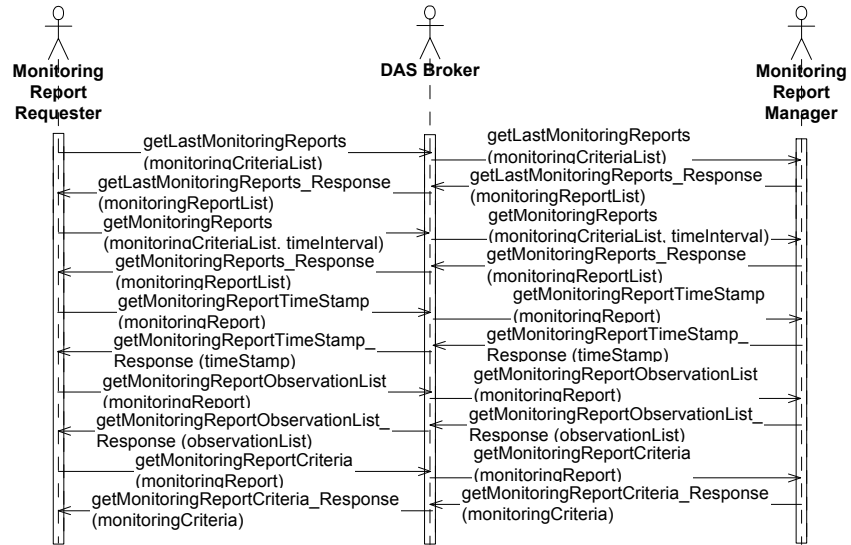
Use Case	Access Data
Roles	<i>Observation Requester, Monitoring Report Requester, Observation Manager, Monitoring Report Manager, DAS Broker.</i>
Type	Primary and essential.
Description	An <i>Observation Requester</i> obtains, by request, observations corresponding to the values of one or more measurement points. A <i>Monitoring Report Requester</i> obtains, by request, monitoring reports taken on a device item.
Pre-conditions	The <i>Observation Requester</i> has discovered the device item. The <i>Monitoring Report Requester</i> has discovered the device item.
Post-conditions	The <i>Observation Requester</i> has obtained the requested observations. The <i>Monitoring Report Requester</i> has obtained the requested monitoring reports.
Diagram	<pre> graph TD OR[Observation Requester] --- AD((Access Data)) MR[Monitoring Report Requester] --- AD DB[DAS Broker] --- AD OMR[Observation Monitoring Report Manager] --- AD MRM[Monitoring Report Manager] --- AD </pre>

Use Case	Access Observations
Roles	<i>Observation Requester, Observation Manager, DAS Broker.</i>
Type	Primary and essential
Description	An <i>Observation Requester</i> obtains, by request, observations corresponding to the values of one or more measurement points. The system may offer <i>Observation Requesters</i> ways to specify filters to access specific observations of measurement points (e.g. the last observations, the observations within a specific interval of time, the observations that exceed certain values).
Pre-conditions	The <i>Observation Requester</i> has discovered the device item.
Post-conditions	The <i>Observation Requester</i> has obtained the requested observations.
Diagram	<pre> graph TD OR[Observation Requester] --- AO((Access Observations)) OM[Observation Manager] --- AO DB[DAS Broker] --- AO </pre>
Known Concepts	<pre> classDiagram class DeviceItem["<<object>> Device Item"] class DeviceModel["<<object>> Device Model"] class MeasurementAddress["<<object>> Measurement Address"] class MeasurementPoint["<<object>> Measurement Point"] class Observation["<<object>> Observation"] class Measurement["<<object>> Measurement"] class CategoryObservation["<<object>> Category Observation"] class Phenomenon["<<object>> Phenomenon"] class PhenomenonType["<<object>> Phenomenon Type"] DeviceItem "*" -- "1" DeviceModel : characterized by DeviceItem "1" -- "*" MeasurementAddress : has MeasurementAddress "*" -- "1" MeasurementPoint : defined by MeasurementAddress "1" -- "*" Observation : taken at Observation "*" -- "*" Measurement : Observation "*" -- "*" CategoryObservation : Observation "*" -- "*" Phenomenon : Observation "*" -- "*" PhenomenonType : Observation "*" -- "1" ObservationManager : manages ObservationRequester "*" -- "1" DASBroker : communicates with ObservationRequester "*" -- "1" ObservationManager : communicates with Measurement "value : Number" CategoryObservation "isPresent: Boolean" </pre>

Example Scenario

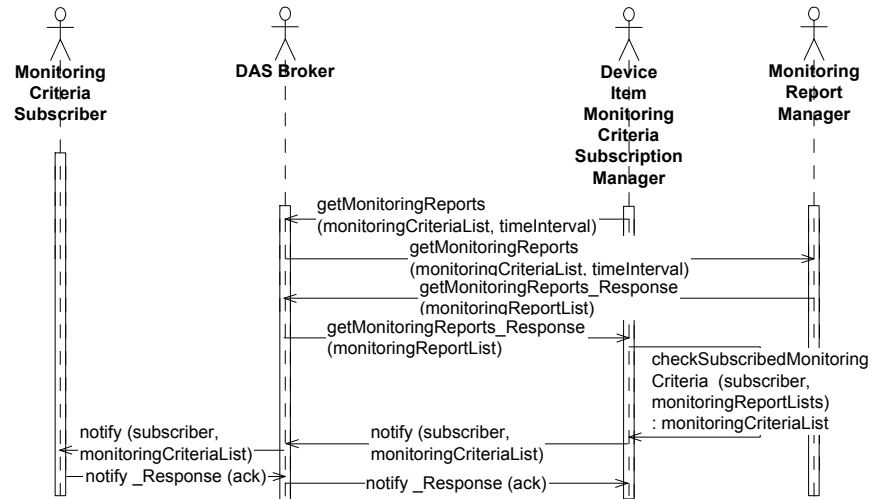


Example Scenario



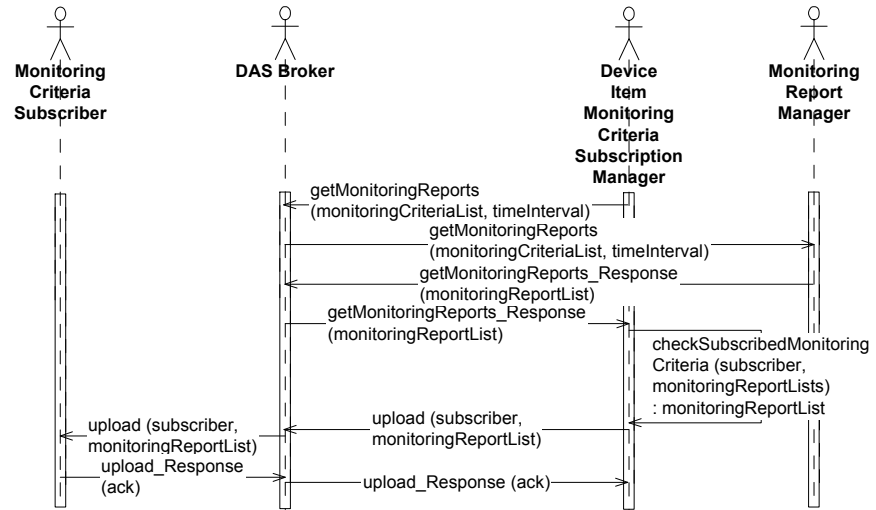
Use Case	Notify Data Availability
Roles	<i>Monitoring Criteria Subscriber, Monitoring Report Manager, Device Item Monitoring Criteria Subscription Manager, DAS Broker.</i>
Type	Primary and essential
Description	A <i>Monitoring Criteria Subscriber</i> receives a notification that new data is available. Data are monitoring reports corresponding to monitoring criteria subscribed by the <i>Monitoring Criteria Subscriber</i> . The notification process is based on the <i>push</i> model.
Pre-conditions	A monitoring report corresponding to certain monitoring criteria subscribed by a <i>Monitoring Criteria Subscriber</i> has been recorded.
Post-conditions	The <i>Monitoring Criteria Subscriber</i> has received a notification of availability of data.
Diagram	<p>The diagram shows a central use case 'Notify Data Availability' connected to four actors: 'Monitoring Criteria Subscriber', 'DAS Broker', 'Monitoring Report Manager', and 'Device Item Monitoring Criteria Subscription Manager'.</p>
Known Concepts	<p>The class diagram illustrates the relationships between various objects. Key classes include: <ul style="list-style-type: none"> Public Criteria (attributes: subscribers : SubscriberList) Private Criteria (attribute: subscriber : Subscriber) Device Item (1 instance) Device Item Composition Monitoring Criteria (related to Device Item) Device Item Event Monitoring Criteria (related to Device Item) Device Item Status Monitoring Criteria (related to Device Item) Device Item Monitoring Criteria (attribute: creator : Creator) Device Item Trigger Condition (related to Device Item Monitoring Criteria) DAS Broker (1 instance) Device Item Monitoring Criteria Subscription Manager (1 instance) Monitoring Criteria Subscriber (1 instance) Monitoring Report Manager (1 instance) Monitoring Report (attribute: time : TimeStamp) Relationships include: <ul style="list-style-type: none"> Public/Private Criteria have subscribers. Device Item Monitoring Criteria has Device Item Trigger Conditions. Device Item Monitoring Criteria is characterized by Monitoring Reports. DAS Broker communicates with Monitoring Criteria Subscriber and Monitoring Report Manager. Monitoring Criteria Subscriber manages subscriptions to Device Item Monitoring Criteria Subscription Manager. Monitoring Report Manager is notified of Monitoring Reports. </p>

Example Scenario



Use Case	Upload Data
Roles	<i>Monitoring Criteria Subscriber, Monitoring Report Manager, Device Item Monitoring Criteria Subscription Manager, DAS Broker.</i>
Type	Primary and essential
Description	<i>Monitoring Criteria Subscribers</i> receive monitoring reports corresponding to monitoring criteria subscribed by them.
Pre-conditions	A monitoring report corresponding to certain monitoring criteria subscribed by a <i>Monitoring Criteria Subscriber</i> has been recorded.
Post-conditions	The <i>Monitoring Criteria Subscriber</i> has received the monitoring report.
Diagram	<p>The diagram shows a central use case 'Upload Data' connected to four actors: 'Monitoring Criteria Subscriber', 'DAS Broker', 'Monitoring Report Manager', and 'Device Item Monitoring Criteria Subscription Manager'.</p>
Known Concepts	<p>The class diagram illustrates the following objects and their relationships:</p> <ul style="list-style-type: none"> Public Criteria (attributes: subscribers : SubscriberList) and Private Criteria (attribute: subscriber : Subscriber) are associated with Device Item Monitoring Criteria via a 'has' relationship. Device Item Monitoring Criteria (attribute: creator : Creator) has a 'has' relationship with Device Item Trigger Condition. Device Item Monitoring Criteria is characterized by Monitoring Report (attribute: time : TimeStamp). Device Item Monitoring Criteria has a 'related to' relationship with Device Item Composition Monitoring Criteria and Device Item Status Monitoring Criteria. Device Item Dataset has a 'has' relationship with Device Item Status Monitoring Criteria. Device Item Monitoring Criteria Subscription Manager (1) manages subscriptions to Device Item Monitoring Criteria (*). Device Item Monitoring Criteria Subscription Manager (1) is uploaded with Monitoring Report (*). Monitoring Report Manager (1) manages Monitoring Report (*). DAS Broker (1) communicates with Device Item Monitoring Criteria Subscription Manager (*), Monitoring Criteria Subscriber (*), and Monitoring Report Manager (*).

Example Scenario



Appendix C DAS Elementary Roles

In this appendix we present the elementary roles that take part in a data acquisition system:

Error! No table of contents entries found.

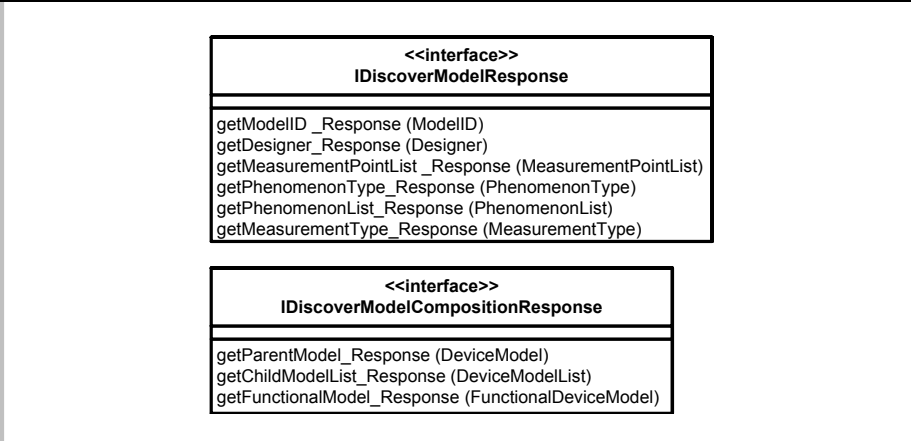
Role	Item Information Requester
Description	An <i>Item Information Requester</i> obtains, by request, the information (serial number, manufacturer, device model, list of measurement addresses, and the measurement points associated to each measurement address) specific to device items and their current composition (parent device items, child device items, and functional device models).
Policies	The device item composition may be discovered only if the device item is a complex device item.
Interfaces	<div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> <p style="text-align: center;"><<interface>> IDiscoverItemResponse</p> <hr/> <p>getSerialNumber_Response (SerialNumber) getManufacturer_Response (Manufacturer) getModel_Response (DeviceModel) getMeasurementAddressList_Response (MeasurementAddressList) getMeasurementPoint_Response (MeasurementPoint)</p> </div> <div style="border: 1px solid black; padding: 5px;"> <p style="text-align: center;"><<interface>> IDiscoverItemCompositionResponse</p> <hr/> <p>getParentItem_Response (DeviceItem) getChildItemList_Response (DeviceItemList) getFunctionalModel_Response (FunctionalDeviceModel)</p> </div>
Known Concepts	<pre> classDiagram class Myself class DeviceItem { manufacturer : Manufacturer serialNumber : SerialNumber } class SingleDeviceItem class ComplexDeviceItem class DeviceAddress class FunctionalDeviceModel class DeviceModel class MeasurementPoint Myself --> DeviceItem : discovers Myself --> DeviceItem : discovers composition of DeviceItem < -- SingleDeviceItem DeviceItem < -- ComplexDeviceItem DeviceItem --> DeviceAddress : composed of DeviceItem --> DeviceAddress : 0..1 parent DeviceItem --> DeviceAddress : 1..* child DeviceItem --> DeviceModel : characterized by DeviceItem --> MeasurementPoint : has DeviceAddress --> FunctionalDeviceModel : installed in DeviceAddress --> MeasurementPoint : defined by </pre>

Role **Model Information Requester**

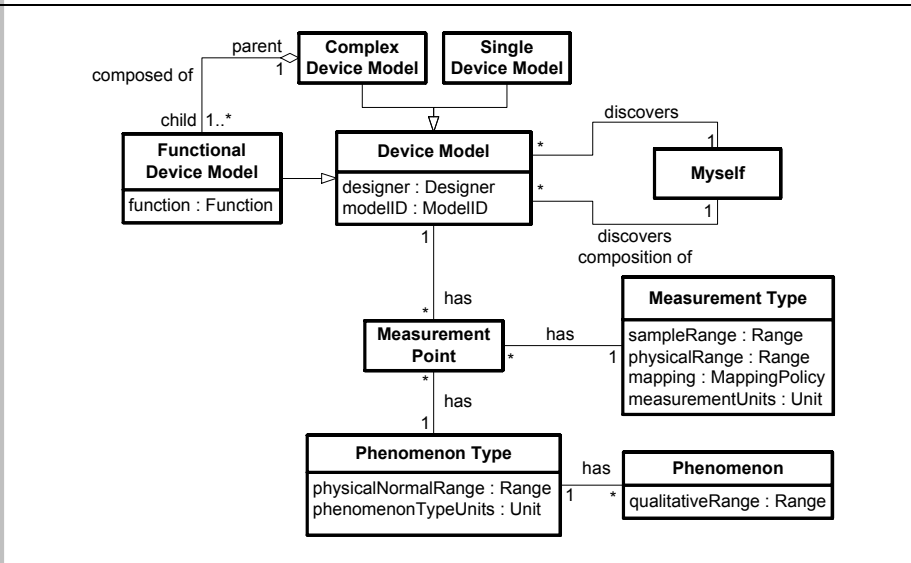
Description A *Model Information Requester* obtains, by request, the information (modelID, designer, measurement points, measurement type and phenomenon type associated with a measurement point, and phenomenon list of a phenomenon type) specific to device models and their composition information.

Policies The device model composition may be discovered only if the device model is a complex device model.

Interfaces

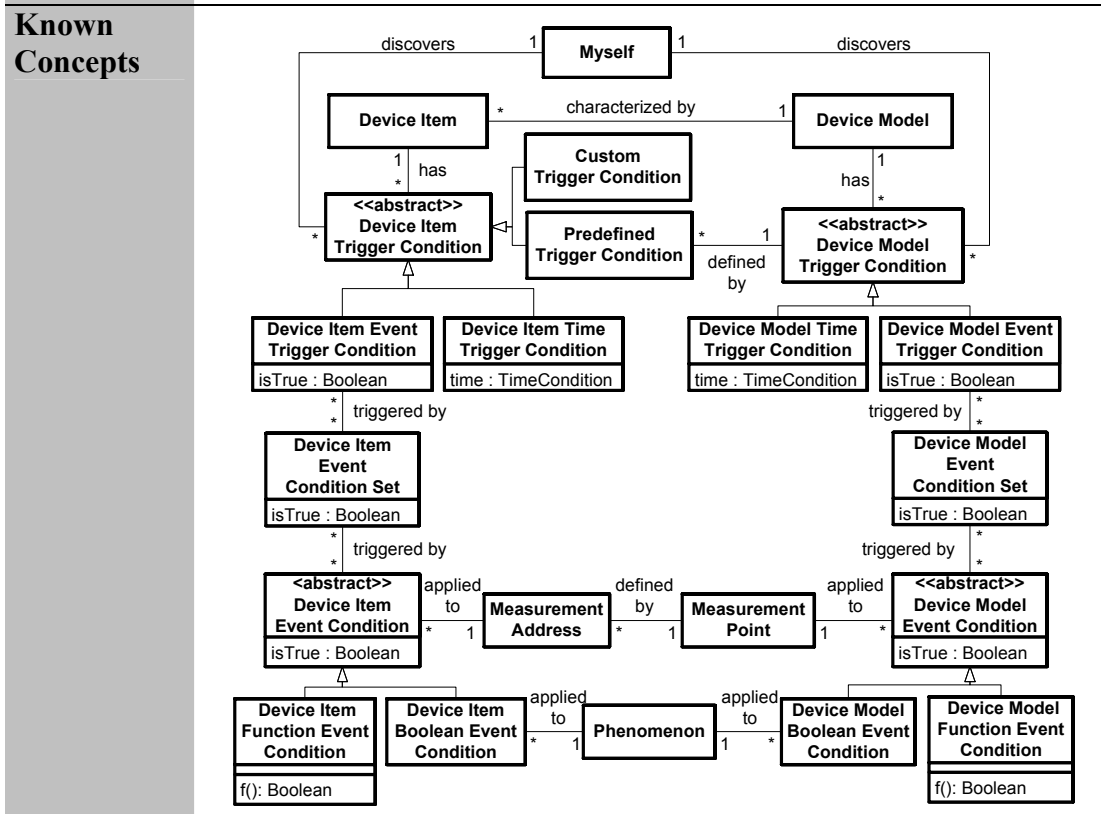
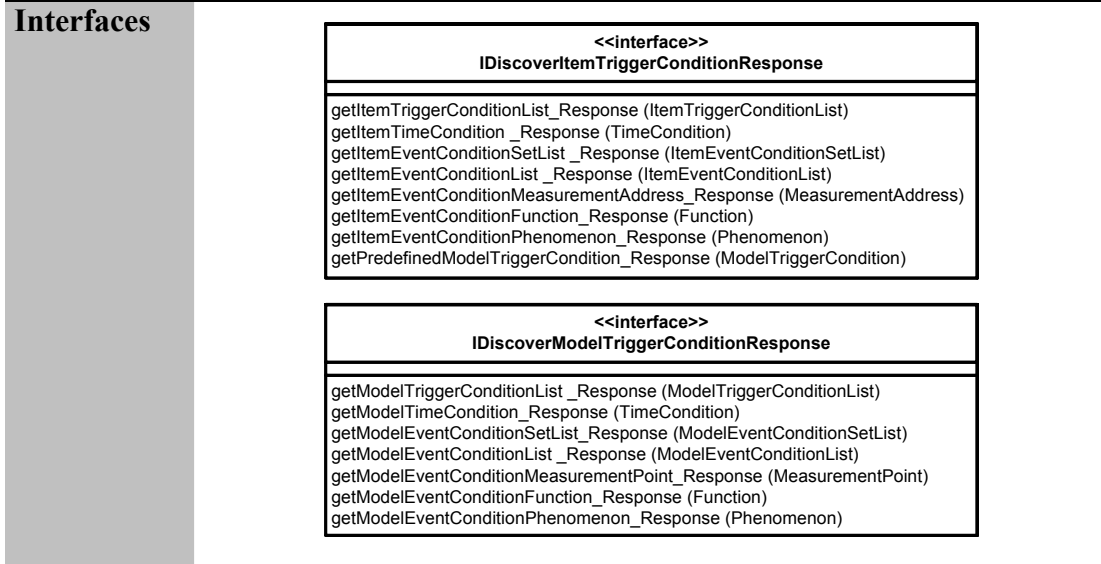


Known Concepts



Role	Dataset Information Requester
Description	A <i>Dataset Information Requester</i> obtains, by request, the datasets defined on device items and device models.
Policies	All device model datasets and device item datasets are public. Therefore, a <i>Dataset Information Requester</i> may discover all of them.
Interfaces	<div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> <p style="text-align: center;"><<interface>> IDiscoverItemDatasetResponse</p> <hr/> <p>getItemDatasetList_Response (ItemDatasetList) getItemDatasetMeasurementAddressList_Response (MeasurementAddressList) getPredefinedModelDataset_Response (ModelDataset)</p> </div> <div style="border: 1px solid black; padding: 5px;"> <p style="text-align: center;"><<interface>> IDiscoverModelDatasetResponse</p> <hr/> <p>getModelDatasetList_Response (ModelDatasetList) getModelDatasetMeasurementPointList_Response (MeasurementPointList)</p> </div>
Known Concepts	<pre> classDiagram class DeviceItem class DeviceModel class MeasurementAddress class MeasurementPoint class CustomDataset class PredefinedDataset class DeviceItemDataset["<<abstract>> Device Item Dataset"] class DeviceModelDataset class Myself DeviceItem "*" -- "1" DeviceModel : characterizes by DeviceItem "1" -- "*" MeasurementAddress : has DeviceItem "1" -- "*" MeasurementAddress : has MeasurementAddress "*" -- "1" MeasurementPoint : defined by MeasurementPoint "*" -- "1" DeviceModel : has MeasurementPoint "*" -- "*" DeviceModelDataset : groups DeviceModel "1" -- "*" DeviceModelDataset : has DeviceModelDataset "*" -- "*" DeviceItemDataset : groups DeviceItemDataset "*" -- "1" Myself : discovers PredefinedDataset "*" -- "1" DeviceModelDataset : defined by Myself "1" -- "*" DeviceModelDataset : discovers DeviceItemDataset < -- CustomDataset DeviceItemDataset < -- PredefinedDataset </pre>

Role	Trigger Condition Information Requester
Description	A <i>Trigger Condition Information Requester</i> obtains, by request, the trigger conditions defined on device items and device models.
Policies	All device model trigger conditions and device item trigger conditions are public. Therefore, a <i>Trigger Condition Information Requester</i> may discover all of them.



Role	Device Item Manager
Description	A <i>Device Item Manager</i> manages device items. It provides the information (serial number, manufacturer, device model, list of measurement addresses, and the measurement points associated to each measurement address) specific to a device item.
Policies	A <i>Device Item Manager</i> provides the composition information of only complex device items.
Interfaces	<div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> <p style="text-align: center;"><<interface>> IDiscoverItem</p> <hr/> <p>getSerialNumber (DeviceItem) getManufacturer (DeviceItem) getModel (DeviceItem) getMeasurementAddressList (DeviceItem) getMeasurementPoint (DeviceItem, MeasurementAddress) getMeasurementAddressList (DeviceItem, MeasurementPointList)</p> </div> <div style="border: 1px solid black; padding: 5px;"> <p style="text-align: center;"><<interface>> IDiscoverItemComposition</p> <hr/> <p>getParentItem (DeviceItem) getChildItemList (DeviceItem) getFunctionalModel (DeviceItem) getChildItemList (DeviceItem, DeviceModelList)</p> </div>
Known Concepts	<pre> classDiagram class Myself class DeviceItem { manufacturer : Manufacturer serialNumber : SerialNumber } class SingleDeviceItem class ComplexDeviceItem class DeviceAddress class MeasurementAddress class FunctionalDeviceModel class DeviceModel Myself "1" -- "*" DeviceItem : manages DeviceItem < -- SingleDeviceItem DeviceItem < -- ComplexDeviceItem DeviceItem "1..*" -- "0..1" DeviceAddress : composed of DeviceItem "1..*" -- "1" DeviceModel : characterized by DeviceItem "1" -- "*" MeasurementAddress : has DeviceAddress "*" -- "1" FunctionalDeviceModel : installed in MeasurementAddress "*" -- "1" DeviceModel : defined by </pre>

Role	Device Model Manager
Description	A <i>Device Model Manager</i> manages device models. It provides the information (modelID, designer, measurement points, measurement type and phenomenon type associated with a measurement point, and the phenomenon list of a phenomenon type) specific to a device model and its composition information.
Policies	A <i>Device Model Manager</i> provides the composition information of only complex device models.
Interfaces	<div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> <p align="center"><<interface>> IDiscoverModel</p> <hr/> <p>getModelID (DeviceModel) getDesigner (DeviceModel) getMeasurementPointList (DeviceModel) getPhenomenonType (DeviceModel, MeasurementPoint) getPhenomenonList (DeviceModel, PhenomenonType) getMeasurementType (DeviceModel, MeasurementPoint)</p> </div> <div style="border: 1px solid black; padding: 5px;"> <p align="center"><<interface>> IDiscoverModelComposition</p> <hr/> <p>getParentModel (DeviceModel) getChildModelList (DeviceModel) getFunctionalModel (DeviceModel)</p> </div>
Known Concepts	<pre> classDiagram class DeviceModel { designer : Designer modelID : ModelID } class ComplexDeviceModel class SingleDeviceModel class FunctionalDeviceModel { function : Function } class MeasurementPoint { } class PhenomenonType { physicalNormalRange : Range phenomenonTypeUnits : Unit } class MeasurementType { sampleRange : Range physicalRange : Range mapping : MappingPolicy measurementUnits : Unit } class Phenomenon { qualitativeRange : Range } class Myself DeviceModel < -- ComplexDeviceModel DeviceModel < -- SingleDeviceModel DeviceModel < -- FunctionalDeviceModel DeviceModel "1" -- "*" MeasurementPoint : has DeviceModel "1" -- "*" MeasurementType : has DeviceModel "1" -- "*" PhenomenonType : has DeviceModel "*" -- "1" Myself : manages MeasurementPoint "1" -- "*" PhenomenonType : has MeasurementPoint "*" -- "1" MeasurementType : has PhenomenonType "1" -- "*" Phenomenon : has DeviceModel "1" -- "1..*" FunctionalDeviceModel : composed of FunctionalDeviceModel "1..*" -- "1" DeviceModel : child </pre>

Role **Device Item Dataset Manager**

Description A *Device Item Dataset Manager* manages datasets defined on device items. It provides the information corresponding to these datasets.

Policies All device item datasets are *public*.

Interfaces

```

<<interface>>
IDiscoverItemDataset

getItemDatasetList (DeviceItem)
getItemDatasetMeasurementAddressList (DeviceItem, ItemDataset)
getItemPredefinedModelDataset (DeviceItem, PredefinedDataset)
    
```

```

<<interface>>
ICreateCustomDataset

createCustomDataset (DeviceItem, MeasurementAddressList)
    
```

```

<<interface>>
ICreatePredefinedDataset

createPredefinedDataset (DeviceItem, DeviceModel, ModelDataset)
createPredefinedDataset (DeviceItem, DeviceModel, ModelDataset, MeasurementAddressList)
: PredefinedDataset
    
```

```

<<interface>>
IDiscoverModelDatasetResponse

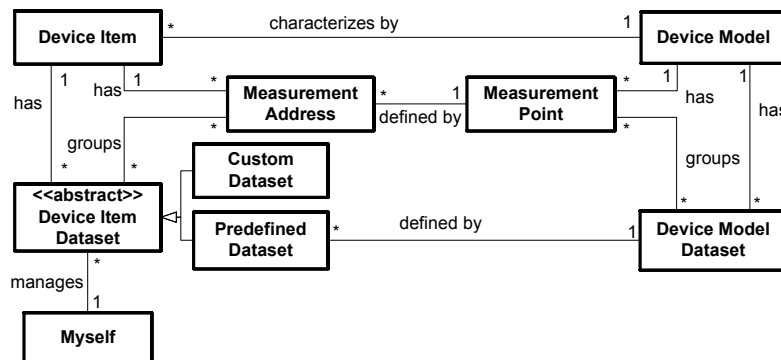
getModelDatasetList_Response (ModelDatasetList)
getModelDatasetMeasurementPointList_Response (MeasurementPointList)
    
```

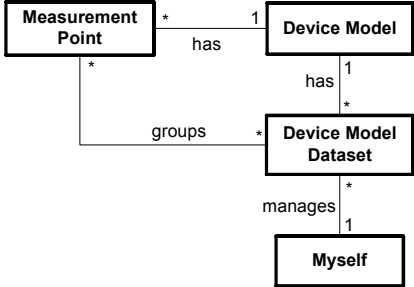
```

<<interface>>
IDiscoverItemResponse

getSerialNumber_Response (SerialNumber)
getManufacturer_Response (Manufacturer)
getModel_Response (DeviceModel)
getMeasurementAddressList_Response (MeasurementAddressList)
getMeasurementPoint_Response (MeasurementPoint)
    
```

Known Concepts



Role	Device Model Dataset Manager
Description	A <i>Device Model Dataset Manager</i> manages datasets defined on device models. It provides the information corresponding to these datasets.
Policies	All device model datasets are <i>public</i> .
Interfaces	<div style="border: 1px solid black; padding: 5px; margin: 10px auto; width: fit-content;"> <pre> <<interface>> IDiscoverModelDataset getModelDatasetList (DeviceModel) getModelDatasetMeasurementPointList (DeviceModel, ModelDataset) </pre> </div>
Known Concepts	 <pre> classDiagram class MeasurementPoint class DeviceModel class DeviceModelDataset class Myself MeasurementPoint "*" -- "1" DeviceModel : has DeviceModel "1" -- "*" DeviceModelDataset : has DeviceModelDataset "*" -- "*" MeasurementPoint : groups Myself "1" -- "*" DeviceModelDataset : manages </pre>

Role	Device Item Trigger Condition Manager
-------------	--

Description	A <i>Device Item Trigger Condition Manager</i> manages trigger conditions defined on device items. It provides the information corresponding to these trigger conditions.
--------------------	---

Policies	All device item trigger conditions are <i>public</i> .
-----------------	--

Interfaces	
-------------------	--

```

<<interface>>
IDiscoverItemTriggerCondition

getItemTriggerConditionList (DeviceItem)
getItemTimeCondition (DeviceItem, ItemTimeTriggerCondition)
getItemEventConditionSetList (DeviceItem, ItemEventTriggerCondition)
getItemEventConditionList (DeviceItem, ItemEventConditionSet)
getItemEventConditionMeasurementAddress (DeviceItem, ItemEventCondition)
getItemEventConditionFunction (DeviceItem, ItemFunctionEventCondition)
getItemEventConditionPhenomenon (DeviceItem, ItemBooleanEventCondition)
getPredefinedModelTriggerCondition (DeviceItem, PredefinedTriggerCondition)

```

```

<<interface>>
ICreateCustomTriggerCondition

createCustomTimeTriggerCondition (DeviceItem, TimeCondition)
createCustomEventTriggerCondition (DeviceItem, Boolean, ItemEventConditionSetList)
createItemBooleanEventCondition (DeviceItem, Boolean, MeasurementAddress, Phenomenon)
createItemEventConditionSet (DeviceItem, Boolean, ItemEventConditionList)
createItemFunctionEventCondition (DeviceItem, Boolean, MeasurementAddress, Function)
createItemBooleanEventCondition (DeviceItem, Boolean, MeasurementAddress, Phenomenon)
    : ItemBooleanEventCondition
createItemFunctionEventCondition (DeviceItem, Boolean, MeasurementAddress, Function)
    : ItemFunctionEventCondition
createItemEventConditionSet (DeviceItem, Boolean, ItemEventConditionList)
    : ItemEventConditionSet

```

```

<<interface>>
ICreatePredefinedTriggerCondition

createPredefinedTimeTriggerCondition (DeviceItem, DeviceModel, ModelTimeTriggerCondition)
createPredefinedTimeTriggerCondition (DeviceItem, DeviceModel, ModelTimeTriggerCondition,
    TimeCondition) : PredefinedTimeTriggerCondition
createPredefinedEventTriggerCondition (DeviceItem, DeviceModel, ModelEventTriggerCondition)
createPredefinedEventTriggerCondition (DeviceItem, DeviceModel, ModelEventTriggerCondition,
    Boolean, ItemEventConditionSetList) : PredefinedEventTriggerCondition

```

```

<<interface>>
IDiscoverModelTriggerConditionResponse

getModelTriggerConditionList_Response (ModelTriggerConditionList)
getModelTimeCondition_Response (TimeCondition)
getModelEventConditionSetList_Response (ModelEventConditionSetList)
getModelEventConditionList_Response (ModelEventConditionList)
getModelEventConditionMeasurementPoint_Response (MeasurementPoint)
getModelEventConditionFunction_Response (Function)
getModelEventConditionPhenomenon_Response (Phenomenon)

```

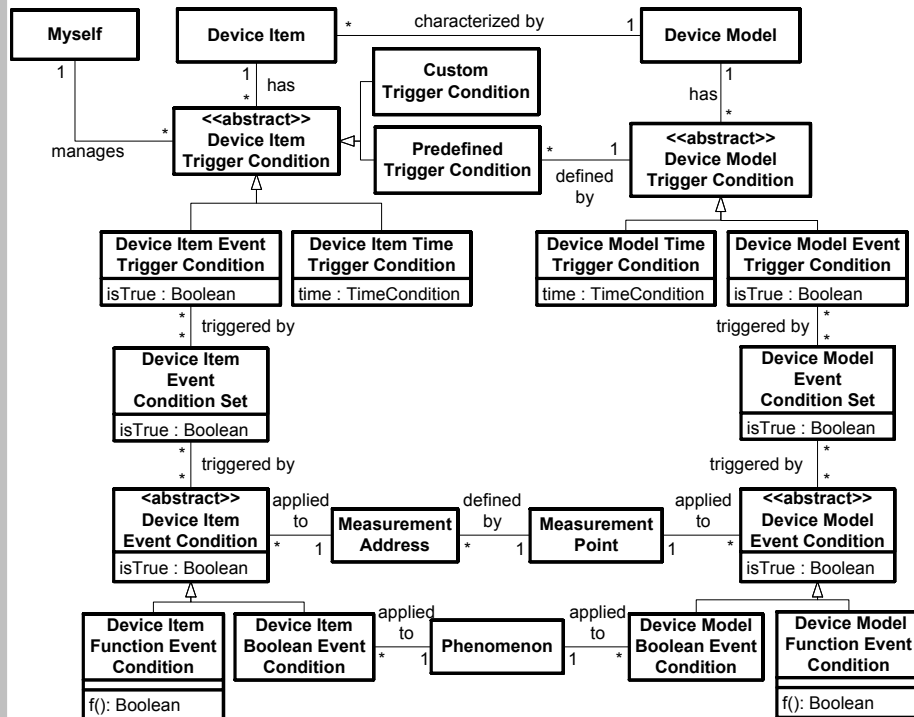
```

<<interface>>
IDiscoverItemResponse

getSerialNumber_Response (SerialNumber)
getManufacturer_Response (Manufacturer)
getModel_Response (DeviceModel)
getMeasurementAddressList_Response (MeasurementAddressList)
getMeasurementPoint_Response (MeasurementPoint)

```

Known Concepts



Role **Device Model Trigger Condition Manager**

Description A *Device Model Trigger Condition Manager* manages trigger conditions defined on device models. It provides the information corresponding to these trigger conditions.

Policies All device model trigger conditions are *public*.

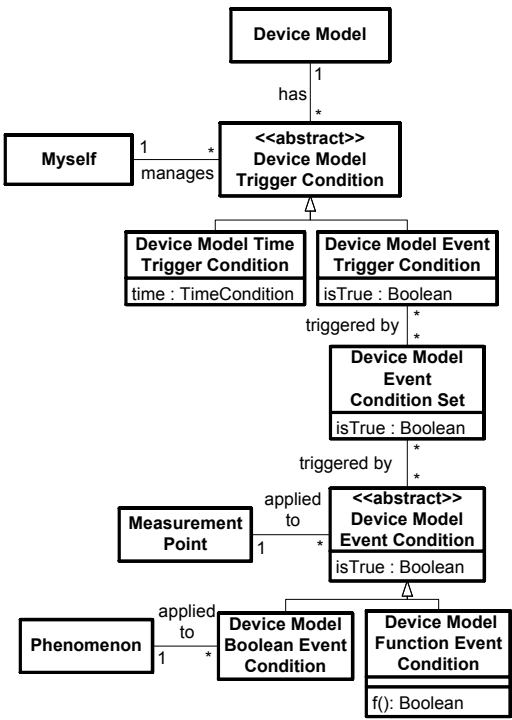
Interfaces

```

<<interface>>
IDiscoverModelTriggerCondition

getModelTriggerConditionList (DeviceModel)
getModelTimeCondition (DeviceModel, ModelTimeTriggerCondition)
getModelEventConditionSetList (DeviceModel, ModelEventTriggerCondition)
getModelEventConditionList (DeviceModel, ModelEventConditionSet)
getModelEventConditionMeasurementPoint (DeviceModel, ModelEventCondition)
getModelEventConditionFunction (DeviceModel, ModelFunctionEventCondition)
getModelEventConditionPhenomenon (DeviceModel, ModelBooleanEventCondition)
    
```

Known Concepts



Role	Device Item Monitoring Criteria Manager
Description	A <i>Device Item Monitoring Criteria Manager</i> manages monitoring criteria defined on device items. It provides the information corresponding to these monitoring criteria.
Policies	<i>Device Item Monitoring Criteria Managers</i> provide device item monitoring criteria defined as <i>public</i> to anyone, and device item monitoring criteria defined as <i>private</i> only to the creators of such monitoring criteria.
Interfaces	<div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> <p style="text-align: center;"><<interface>> IDiscoverItemMonitoringCriteria</p> <hr/> <p>getItemMonitoringCriteriaList (DeviceItem) getItemPublicMonitoringCriteriaList (DeviceItem) getItemPrivateMonitoringCriteriaList (DeviceItem, MonitoringCriteriaCreatorList) getItemMonitoringCriteriaCreator (DeviceItem, ItemMonitoringCriteria) getItemMonitoringCriteriaSubscriberList (DeviceItem, ItemMonitoringCriteria) getItemMonitoringCriteriaTriggerCondition (DeviceItem, ItemMonitoringCriteria) getItemCompositionMonitoringCriteriaList (DeviceItem) getItemCompositionMonitoringCriteriaItem (DeviceItem, ItemCompositionMonitoringCriteria) getItemEventMonitoringCriteriaList (DeviceItem) getItemStatusMonitoringCriteriaList (DeviceItem) getItemStatusMonitoringCriteriaDataset (DeviceItem, ItemStatusMonitoringCriteria) getPredefinedModelMonitoringCriteria (DeviceItem, PredefinedMonitoringCriteria)</p> </div> <div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> <p style="text-align: center;"><<interface>> ICreateCustomMonitoringCriteria</p> <hr/> <p>createCustomCompositionMonitoringCriteria (MonitoringCriteriaCreator, AccessType, DeviceItem, DeviceItemList, ItemTriggerCondition) createCustomEventMonitoringCriteria (MonitoringCriteriaCreator, AccessType, DeviceItem, ItemEventTriggerCondition) createCustomStatusMonitoringCriteria (MonitoringCriteriaCreator, AccessType, DeviceItem, ItemDataset, ItemTriggerCondition)</p> </div> <div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> <p style="text-align: center;"><<interface>> ICreatePredefinedMonitoringCriteria</p> <hr/> <p>createPredefinedCompositionMonitoringCriteria (MonitoringCriteriaCreator, AccessType, DeviceItem, DeviceModel, ModelCompositionMonitoringCriteria) createPredefinedEventMonitoringCriteria (MonitoringCriteriaCreator, AccessType, DeviceItem, DeviceModel, ModelEventMonitoringCriteria) createPredefinedStatusMonitoringCriteria (MonitoringCriteriaCreator, AccessType, DeviceItem, DeviceModel, ModelStatusMonitoringCriteria)</p> </div> <div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> <p style="text-align: center;"><<interface>> IDiscoverModelMonitoringCriteriaResponse</p> <hr/> <p>getModelMonitoringCriteriaList_Response (ModelMonitoringCriteriaList) getModelMonitoringCriteriaTriggerCondition_Response (ModelTriggerCondition) getModelCompositionMonitoringCriteriaList_Response (ModelCompositionMonitoringCriteriaList) getModelCompositionMonitoringCriteriaModelList_Response (DeviceModelList) getModelEventMonitoringCriteriaList_Response (ModelEventMonitoringCriteriaList) getModelStatusMonitoringCriteriaList_Response (ModelStatusMonitoringCriteriaList) getModelStatusMonitoringCriteriaDataset_Response (ModelDataset)</p> </div> <div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> <p style="text-align: center;"><<interface>> IDiscoverItemCompositionResponse</p> <hr/> <p>getParentItem_Response (DeviceItem) getChildItemList_Response (DeviceItemList) getFunctionalModel_Response (FunctionalDeviceModel)</p> </div> <div style="border: 1px solid black; padding: 5px;"> <p style="text-align: center;"><<interface>> ICreatePredefinedTriggerConditionResponse</p> <hr/> <p>createPredefinedTimeTriggerCondition_Response (PredefinedTimeTriggerCondition) createPredefinedEventTriggerCondition_Response (PredefinedEventTriggerCondition)</p> </div>


```

<<interface>>
ICreatePredefinedDatasetResponse
-----
createPredefinedDataset_Response (PredefinedDataset)

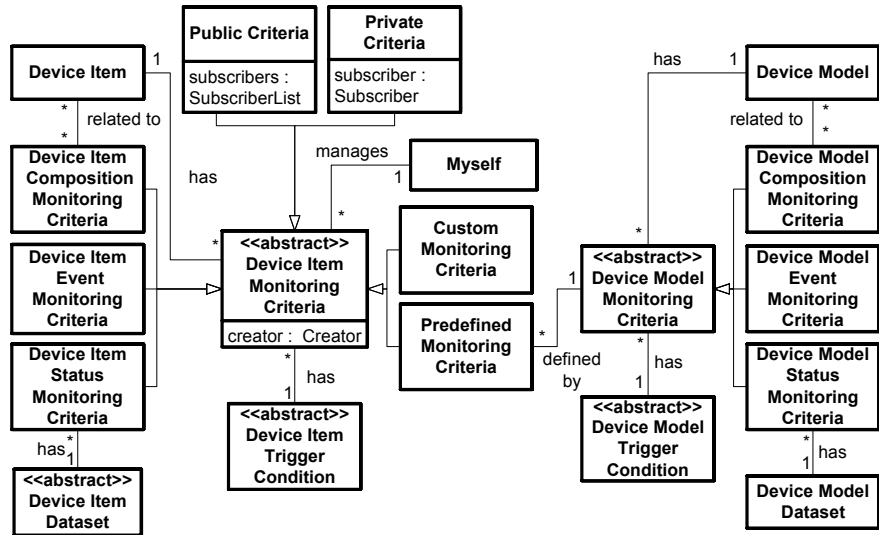
```

```

<<interface>>
IManageMonitoringCriteriaSubscriber
-----
addSubscriber (MonitorinCriteriaSubscriber, DeviceItem, ItemMonitoringCriteria)
removeSubscriber (MonitorinCriteriaSubscriber, DeviceItem, ItemMonitoringCriteria)

```

Known Concepts

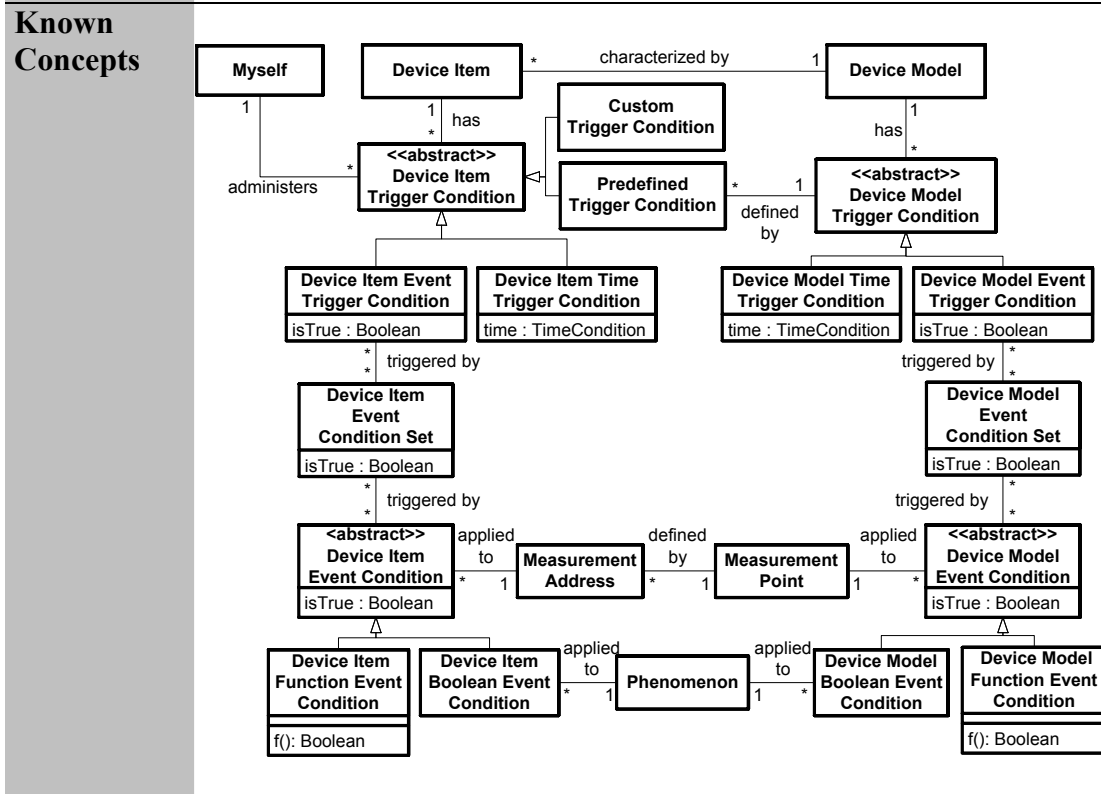
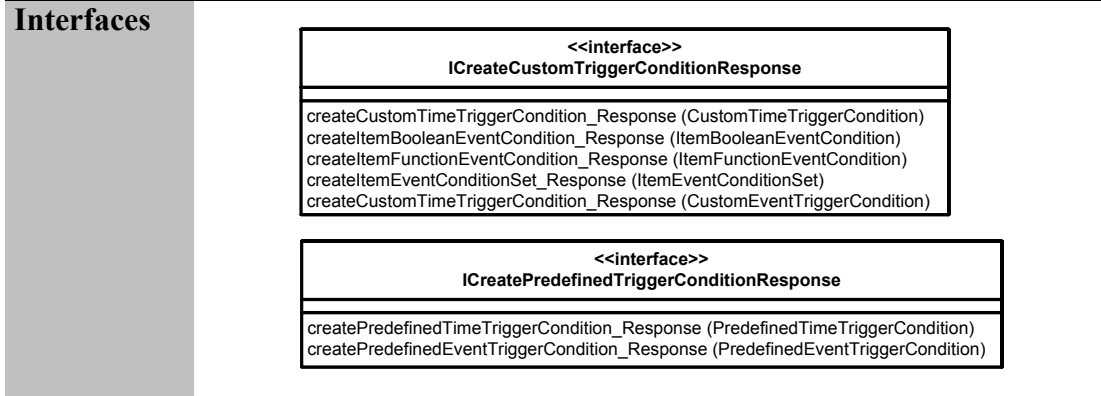


Role	Device Model Monitoring Criteria Manager
Description	A <i>Device Model Monitoring Criteria Manager</i> manages monitoring criteria defined on device models. It provides the information corresponding to these monitoring criteria.
Policies	All device model monitoring criteria are <i>public</i> .
Interfaces	<div style="border: 1px solid black; padding: 5px;"> <p style="text-align: center;"><<interface>> IDiscoverModelMonitoringCriteria</p> <hr/> <p>getModelMonitoringCriteriaList (DeviceModel) getModelMonitoringCriteriaTriggerCondition (DeviceModel, ModelMonitoringCriteria) getModelCompositionMonitoringCriteriaList (DeviceModel) getModelCompositionMonitoringCriteriaModelList (DeviceModel, ModelCompositionMonitoringCriteria) getModelEventMonitoringCriteriaList (DeviceModel) getModelStatusMonitoringCriteriaList (DeviceModel) getModelStatusMonitoringCriteriaDataset (DeviceModel, ModelStatusMonitoringCriteria)</p> </div>
Known Concepts	<pre> classDiagram class Myself class DeviceModelMonitoringCriteria["<<abstract>> Device Model Monitoring Criteria"] class DeviceModel class DeviceModelCompositionMonitoringCriteria class DeviceModelEventMonitoringCriteria class DeviceModelStatusMonitoringCriteria class DeviceModelTriggerCondition["<<abstract>> Device Model Trigger Condition"] class DeviceModelDataset Myself "1" -- "*" DeviceModelMonitoringCriteria : manages DeviceModelMonitoringCriteria "*" -- "1" DeviceModel : has DeviceModelMonitoringCriteria "*" -- "*" DeviceModelCompositionMonitoringCriteria : related to DeviceModelMonitoringCriteria "*" -- "*" DeviceModelEventMonitoringCriteria DeviceModelMonitoringCriteria "*" -- "*" DeviceModelStatusMonitoringCriteria DeviceModelMonitoringCriteria "*" -- "1" DeviceModelTriggerCondition : has DeviceModelMonitoringCriteria "*" -- "*" DeviceModelDataset : has </pre>

Role	Dataset Administrator
Description	A <i>Dataset Administrator</i> administers (creates, modifies or removes) datasets of device items. A <i>Dataset Administrator</i> can define an entirely new dataset for a device item (<i>custom</i> dataset), or define a dataset for a device item from a dataset predefined on a device model (<i>predefined</i> dataset).
Policies	All device item datasets are <i>public</i>
Interfaces	<div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> <p style="text-align: center;"><<interface>> ICreateCustomDatasetResponse</p> <hr/> <p>createCustomDataset_Response (CustomDataset)</p> </div> <div style="border: 1px solid black; padding: 5px;"> <p style="text-align: center;"><<interface>> ICreatePredefinedDatasetResponse</p> <hr/> <p>createPredefinedDataset_Response (PredefinedDataset)</p> </div>
Known Concepts	<pre> classDiagram class DeviceItem class DeviceModel class MeasurementAddress class MeasurementPoint class CustomDataset class PredefinedDataset class DeviceItemDataset["<<abstract>> Device Item Dataset"] class DeviceModelDataset class Myself DeviceItem "*" -- "1" DeviceModel : characterizes by DeviceItem "1" -- "*" MeasurementAddress : has DeviceItem "1" -- "*" MeasurementAddress : has MeasurementAddress "*" -- "1" MeasurementPoint : defined by MeasurementPoint "*" -- "1" DeviceModel : has MeasurementPoint "*" -- "*" DeviceModelDataset : groups DeviceItemDataset < -- CustomDataset DeviceItemDataset < -- PredefinedDataset DeviceModelDataset "*" -- "1" PredefinedDataset : defined by Myself "*" -- "1" DeviceItemDataset : administers </pre>

Role	Trigger Condition Administrator
Description	A <i>Trigger Condition Administrator</i> administers (creates, modifies or removes) trigger conditions of device items. A trigger condition can be based on time or based on an event. A <i>Trigger Condition Administrator</i> can define an entirely new trigger condition for a device item (<i>custom</i> trigger condition), or define a trigger condition from a trigger condition predefined on a device model (<i>predefined</i> trigger condition).

Policies All device item trigger conditions are *public*.



Role	Monitoring Criteria Administrator
Description	A <i>Monitoring Criteria Administrator</i> administers (creates, modifies or removes) monitoring criteria of a device item. <i>Monitoring Criteria Administrators</i> can define entirely new monitoring criteria for a device item, such monitoring criteria being called <i>custom</i> , or define monitoring criteria from monitoring criteria predefined on a device model, such monitoring criteria being called <i>predefined</i> . <i>Monitoring Criteria Administrators</i> can define device item monitoring criteria as <i>public</i> , meaning that any supervisor of the system can access monitoring reports of such monitoring criteria, or <i>private</i> , meaning that only the creator of the monitoring criteria is allowed to access monitoring reports corresponding to such monitoring criteria.
Policies	A <i>Monitoring Criteria Administrator</i> can administer all device item monitoring criteria.
Interfaces	<div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> <p style="text-align: center;"><<interface>> ICreateCustomMonitoringCriteriaResponse</p> <hr/> <p>createCustomCompositionMonitoringCriteria_Response (CustomCompositionMonitoringCriteria) createCustomEventMonitoringCriteria_Response (CustomEventMonitoringCriteria) createCustomStatusMonitoringCriteria_Response (CustomStatusMonitoringCriteria)</p> </div> <div style="border: 1px solid black; padding: 5px;"> <p style="text-align: center;"><<interface>> ICreatePredefinedMonitoringCriteriaResponse</p> <hr/> <p>createPredefinedCompositionMonitoringCriteria_Response (PredefinedCompositionMonitoringCriteria) createPredefinedEventMonitoringCriteria_Response (PredefinedEventMonitoringCriteria) createPredefinedStatusMonitoringCriteria_Response (PredefinedStatusMonitoringCriteria)</p> </div>
Known Concepts	<pre> classDiagram class DeviceItem { +DeviceItemCompositionMonitoringCriteria +DeviceItemEventMonitoringCriteria +DeviceItemStatusMonitoringCriteria +DeviceItemDataset } class DeviceModel { +DeviceModelCompositionMonitoringCriteria +DeviceModelEventMonitoringCriteria +DeviceModelStatusMonitoringCriteria +DeviceModelDataset } class PublicCriteria { +subscribers : SubscriberList } class PrivateCriteria { +subscriber : Subscriber } class Myself class CustomMonitoringCriteria class PredefinedMonitoringCriteria class DeviceItemMonitoringCriteria { <<abstract>> +creator : Creator } class DeviceModelMonitoringCriteria { <<abstract>> } class DeviceItemTriggerCondition { <<abstract>> } class DeviceModelTriggerCondition { <<abstract>> } DeviceItem "1" -- "*" DeviceItemMonitoringCriteria DeviceItemMonitoringCriteria < -- DeviceItemCompositionMonitoringCriteria DeviceItemMonitoringCriteria < -- DeviceItemEventMonitoringCriteria DeviceItemMonitoringCriteria < -- DeviceItemStatusMonitoringCriteria DeviceItemMonitoringCriteria < -- DeviceItemTriggerCondition DeviceItemMonitoringCriteria "1" -- "*" PublicCriteria DeviceItemMonitoringCriteria "1" -- "*" PrivateCriteria DeviceItemMonitoringCriteria "1" -- "1" Myself DeviceItemMonitoringCriteria "1" -- "*" CustomMonitoringCriteria DeviceItemMonitoringCriteria "1" -- "*" PredefinedMonitoringCriteria DeviceItemMonitoringCriteria "1" -- "1" DeviceItemTriggerCondition DeviceModel "1" -- "*" DeviceModelMonitoringCriteria DeviceModelMonitoringCriteria < -- DeviceModelCompositionMonitoringCriteria DeviceModelMonitoringCriteria < -- DeviceModelEventMonitoringCriteria DeviceModelMonitoringCriteria < -- DeviceModelStatusMonitoringCriteria DeviceModelMonitoringCriteria < -- DeviceModelTriggerCondition DeviceModelMonitoringCriteria "1" -- "*" DeviceModelTriggerCondition DeviceModelMonitoringCriteria "1" -- "1" DeviceModelTriggerCondition DeviceItem "1" -- "*" DeviceItemDataset DeviceModel "1" -- "*" DeviceModelDataset DeviceItem "1" -- "*" DeviceModel : related to DeviceItemMonitoringCriteria "*" -- "*" DeviceModelMonitoringCriteria : has DeviceItemMonitoringCriteria "*" -- "1" DeviceItemTriggerCondition : has DeviceModelMonitoringCriteria "*" -- "1" DeviceModelTriggerCondition : has DeviceItemMonitoringCriteria "*" -- "1" DeviceModelMonitoringCriteria : defined by </pre>

Role	Monitoring Criteria Subscription Administrator
Description	A <i>Monitoring Criteria Subscription Administrator</i> administers (creates, modifies or removes) subscriptions of interest on certain monitoring criteria.
Policies	Device item monitoring criteria defined as <i>public</i> can be subscribed by anyone, and device item monitoring criteria defined as <i>private</i> can be subscribed only by the creator of such monitoring criteria.
Interfaces	<pre> <<interface>> IAdministerMonitoringCriteriaSubscriptionResponse ----- subscribeMonitoringCriteria_Response (MonitoringCriteriaSubscription) unsubscribeMonitoringCriteria_Response (MonitoringCriteriaUnsubscription) </pre>
Known Concepts	<pre> classDiagram class DeviceItem class Myself class DeviceItemMonitoringCriteria["<<abstract>> Device Item Monitoring Criteria"] { creator : Creator } class PublicCriteria { subscribers : SubscriberList } class PrivateCriteria { subscriber : Subscriber } DeviceItem "1" -- "*" DeviceItemMonitoringCriteria : has Myself "1" -- "*" DeviceItemMonitoringCriteria : administers subscriptions to DeviceItemMonitoringCriteria < -- PublicCriteria DeviceItemMonitoringCriteria < -- PrivateCriteria </pre>

Role	Device Item Monitoring Criteria Subscription Manager
Description	A <i>Device Item Monitoring Criteria Subscription Manager</i> manages subscriptions of interest on certain monitoring criteria.
Policies	Device item monitoring criteria defined as <i>public</i> can be subscribed by anyone, and device item monitoring criteria defined as <i>private</i> can be subscribed only by the creator of such monitoring criteria.
Interfaces	<div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> <p style="text-align: center;"><<interface>> IAdministerMonitoringCriteriaSubscription</p> <hr/> <p>subscribeMonitoringCriteria (MonitoringCriteriaSubscriber, DeviceItem, ItemMonitoringCriteria) unsubscribeMonitoringCriteria (MonitoringCriteriaSubscriber, DeviceItem, ItemMonitoringCriteria)</p> </div> <div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> <p style="text-align: center;"><<interface>> IManageMonitoringCriteriaSubscription</p> <hr/> <p>createMonitoringCriteriaSubscription (MonitoringCriteriaSubscriber, DeviceItem, ItemMonitoringCriteria) : MonitoringCriteriaSubscription removeMonitoringCriteriaSubscription (MonitoringCriteriaSubscriber, DeviceItem, ItemMonitoringCriteria) : MonitoringCriteriaUnsubscription</p> </div> <div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> <p style="text-align: center;"><<interface>> IManageMonitoringCriteriaSubscriberResponse</p> <hr/> <p>addSubscriber_Response (Acknowledgement) removeSubscriber_Response (Acknowledgement)</p> </div> <div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> <p style="text-align: center;"><<interface>> IAccessMonitoringReportResponse</p> <hr/> <p>getLastMonitoringReports_Response (MonitoringReportList) getMonitoringReports_Response (MonitoringReportList) getMonitoringReportTimeStamp_Response (TimeStamp) getMonitoringReportObservationList_Response (ObservationList) getMonitoringReportCriteria_Response (MonitoringCriteria)</p> </div> <div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> <p style="text-align: center;"><<interface>> ICheckMonitoringCriteria</p> <hr/> <p>checkSubscribedMonitoringCriteria (MonitoringCriteriaSubscriber, MonitoringReportLists) : MonitoringCriteriaList</p> </div> <div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> <p style="text-align: center;"><<interface>> INotifyMonitoringReportResponse</p> <hr/> <p>notify_Response (Acknowledgement)</p> </div> <div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> <p style="text-align: center;"><<interface>> IUploadMonitoringReportResponse</p> <hr/> <p>upload_Response (Acknowledgement)</p> </div>
Known Concepts	<pre> classDiagram class Myself class DeviceItem class DeviceItemMonitoringCriteria["<<abstract>> Device Item Monitoring Criteria"] { creator : Creator } class PublicCriteria { subscribers : SubscriberList } class PrivateCriteria { subscriber : Subscriber } Myself "1" -- "*" DeviceItemMonitoringCriteria : manages subscriptions to DeviceItem "1" -- "*" DeviceItemMonitoringCriteria : has DeviceItemMonitoringCriteria < -- PublicCriteria DeviceItemMonitoringCriteria < -- PrivateCriteria </pre>

Role	Observation Requester
Description	An <i>Observation Requester</i> obtains, by request, observations corresponding to the values of one or more measurement points of a device item.
Policies	The system may offer <i>Observation Requesters</i> ways to specify filters to access specific observations of measurement points (e.g. the last observations, the observations within a specific interval of time, the observations that exceed certain values).
Interfaces	<div style="border: 1px solid black; padding: 5px; margin-bottom: 5px;"> <p style="text-align: center;"><<interface>> IAccessObservationResponse</p> <hr/> <p>getLastObservations_Response (ObservationList) getObservations_Response (ObservationList) getObservationTimeStamp_Response (TimeStamp) getObservationQuality_Response (DataQuality)</p> </div> <div style="border: 1px solid black; padding: 5px; margin-bottom: 5px;"> <p style="text-align: center;"><<interface>> IAccessMeasurementResponse</p> <hr/> <p>getMeasurements_Response (MeasurementList) getMeasurementValue_Response (Value)</p> </div> <div style="border: 1px solid black; padding: 5px;"> <p style="text-align: center;"><<interface>> IAccessCategoryObservationResponse</p> <hr/> <p>getCategoryObservations_Response (CategoryObservationList) getCategoryObservationPresence_Response (Boolean)</p> </div>
Known Concepts	<pre> classDiagram class DeviceItem class DeviceModel class MeasurementAddress class MeasurementPoint class Measurement class CategoryObservation class Phenomenon class PhenomenonType class Observation class Myself DeviceItem "*" -- "1" DeviceModel : characterized by DeviceItem "1" -- "*" MeasurementAddress : has MeasurementAddress "*" -- "1" MeasurementPoint : defined by MeasurementAddress "1" -- "*" Observation : taken at Observation < -- CategoryObservation Observation "time : TimeStamp" Observation "quality : DataQualifier" Observation "*" -- "1" Phenomenon : has Phenomenon "*" -- "1" PhenomenonType : has Myself "1" -- "*" Observation : accesses </pre>

Role	Monitoring Report Requester
Description	A <i>Monitoring Report Requester</i> obtains, by request, monitoring reports taken on a device item.
Policies	The system may offer <i>Monitoring Report Requesters</i> ways to specify filters to access specific monitoring reports. (e.g. the last monitoring report of a certain monitoring criteria, the monitoring reports of a certain monitoring criteria within a specific interval of time).
Interfaces	<div style="border: 1px solid black; padding: 5px;"> <p style="text-align: center;"><<interface>> IAccessMonitoringReportResponse</p> <hr/> <p>getLastMonitoringReports_Response (MonitoringReportList) getMonitoringReports_Response (MonitoringReportList) getMonitoringReportTimeStamp_Response (TimeStamp) getMonitoringReportObservationList_Response (ObservationList) getMonitoringReportCriteria_Response (MonitoringCriteria)</p> </div>
Known Concepts	<pre> classDiagram class DeviceItem { +DeviceItemCompositionMonitoringCriteria +DeviceItemEventMonitoringCriteria +DeviceItemStatusMonitoringCriteria } class DeviceItemMonitoringCriteria { <<abstract>> } class DeviceItemTriggerCondition { <<abstract>> } class PublicCriteria { +subscribers : SubscriberList } class PrivateCriteria { +subscriber : Subscriber } class MonitoringCriteria { +creator : Creator } class MonitoringReport { +time : TimeStamp } class Observation { <<abstract>> +time : TimeStamp +quality : DataQualifier } class Myself DeviceItem "1" -- "*" MonitoringCriteria : has DeviceItem "1" -- "*" Observation : has DeviceItem "1" -- "*" Observation : related to DeviceItemMonitoringCriteria < -- DeviceItemEventMonitoringCriteria DeviceItemMonitoringCriteria < -- DeviceItemStatusMonitoringCriteria MonitoringCriteria < -- PublicCriteria MonitoringCriteria < -- PrivateCriteria MonitoringCriteria "1" -- "*" Observation : characterized by MonitoringCriteria "1" -- "*" DeviceItemTriggerCondition : has Observation "1" -- "*" Myself : accesses </pre>

Role	Observation Manager
Description	An <i>Observation Manager</i> manages observations recorded on device items.
Policies	An <i>Observation Manager</i> may offer <i>Observation Requesters</i> ways to specify filters to access specific observations of measurement points (e.g. the last observations, the observations within a specific interval of time, the observations that exceed certain values).
Interfaces	<div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> <p style="text-align: center;"><<interface>> IAccessObservation</p> <hr/> <p>getLastObservations (MeasurementAddressList) getObservations (MeasurementAddressList, TimeInterval) getObservationTimeStamp (Observation) getObservationQuality (Observation)</p> </div> <div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> <p style="text-align: center;"><<interface>> IAccessMeasurement</p> <hr/> <p>getMeasurements (MeasurementAddressList, ValueRangeList, TimeInterval) getMeasurementValue (Measurement)</p> </div> <div style="border: 1px solid black; padding: 5px;"> <p style="text-align: center;"><<interface>> IAccessCategoryObservation</p> <hr/> <p>getCategoryObservations (MeasurementAddressList, PhenomenonList, TimeInterval) getCategoryObservationPresence (CategoryObservation)</p> </div>
Known Concepts	<pre> classDiagram class DeviceItem class DeviceModel class MeasurementAddress class MeasurementPoint class Observation class CategoryObservation class Phenomenon class PhenomenonType class Myself DeviceItem "*" -- "1" DeviceModel : characterized by DeviceItem "1" -- "*" MeasurementAddress : has MeasurementAddress "*" -- "1" MeasurementPoint : defined by MeasurementAddress "1" -- "*" Observation : taken at Observation < -- CategoryObservation Observation "*" -- "1" Phenomenon : has Observation "*" -- "1" Myself : manages MeasurementPoint "*" -- "1" PhenomenonType : has Phenomenon "*" -- "1" PhenomenonType : has </pre> <p>The diagram illustrates the following relationships:</p> <ul style="list-style-type: none"> Device Item (multiplicity *) is characterized by Device Model (multiplicity 1). Device Item (multiplicity 1) has Measurement Address (multiplicity *). Measurement Address (multiplicity *) is defined by Measurement Point (multiplicity 1). Measurement Address (multiplicity 1) is taken at Observation (multiplicity *). Observation is an abstract class with Category Observation as a specialization. Observation (multiplicity *) has Phenomenon (multiplicity 1). Observation (multiplicity *) manages Myself (multiplicity 1). Measurement Point (multiplicity *) has Phenomenon Type (multiplicity 1). Phenomenon (multiplicity *) has Phenomenon Type (multiplicity 1).

Role	Monitoring Criteria Subscriber
Description	A <i>Monitoring Criteria Subscriber</i> subscribes for receiving uploads with monitoring reports, or notifications of availability of monitoring reports, corresponding to certain monitoring criteria. A <i>Monitoring Criteria Subscriber</i> receives uploads with monitoring reports, or notifications of availability of monitoring reports, corresponding to monitoring criteria subscribed previously by this subscriber.
Policies	The upload and notification processes are based on the <i>push</i> model. Device item monitoring criteria defined as <i>public</i> can be subscribed by anyone, and device item monitoring criteria defined as <i>private</i> can be subscribed only by the creator of such monitoring criteria.
Interfaces	<div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> <p style="text-align: center;"><<interface>> INotifyMonitoringReport</p> <hr/> <p>notify (MonitoringCriteriaSubscriber, MonitoringCriteriaList)</p> </div> <div style="border: 1px solid black; padding: 5px;"> <p style="text-align: center;"><<interface>> IUploadMonitoringReport</p> <hr/> <p>upload (MonitoringCriteriaSubscriber, MonitoringReportList)</p> </div>
Known Concepts	<pre> classDiagram class DeviceItem class DeviceItemMonitoringCriteria["<<abstract>> Device Item Monitoring Criteria"] class DeviceItemCompositionMonitoringCriteria class DeviceItemEventMonitoringCriteria class DeviceItemStatusMonitoringCriteria class DeviceItemTriggerCondition["<<abstract>> Device Item Trigger Condition"] class PublicCriteria class PrivateCriteria class MonitoringReport class Observation["<<abstract>> Observation"] class Myself DeviceItem "1" -- "*" DeviceItemMonitoringCriteria : has DeviceItemMonitoringCriteria < -- DeviceItemCompositionMonitoringCriteria DeviceItemMonitoringCriteria < -- DeviceItemEventMonitoringCriteria DeviceItemMonitoringCriteria < -- DeviceItemStatusMonitoringCriteria DeviceItemMonitoringCriteria < -- DeviceItemTriggerCondition DeviceItemMonitoringCriteria "1" -- "*" Observation : characterized by Observation "1" -- "*" MonitoringReport : has MonitoringReport "1" -- "*" Myself : is notified of Myself "1" -- "*" MonitoringReport : is uploaded with PublicCriteria "1" -- "*" DeviceItemMonitoringCriteria : subscribers : SubscriberList PrivateCriteria "1" -- "*" DeviceItemMonitoringCriteria : subscriber : Subscriber </pre>

Bibliography

- [1] F. Olken, H. A. Jacobsen, C. McParland, M. A. Piette, and M. F. Anderson, “*Objects lessons learned from a distributed system for remote building monitoring and operation*” presented at Conference on Object-oriented Programming, Systems, Languages and Applications, Vancouver, Canada, October 18-22, 1998, <http://www.lbl.gov/~olken/rbo/rbo.html>.
- [2] R. Itschner, C. Pommerell, and M. Rutishauser, “*GLASS: Remote Monitoring of Embedded Systems in Power Engineering*” in IEEE Internet Computing, vol 2, 1998.
- [3] A. Fabri, T. Nieva, and P. Umiliacchi, “*Use of the Internet for Remote Train Monitoring and Control: the ROSIN Project*” presented at Rail Technology '99, London, UK, September 7-8, 1999, <http://icawww.epfl.ch/nieva/thesis/Conferences/RailTech99/article/RailTech99.PDF>.
- [4] OPC Foundation, “*OLE for Process and Control Standard*”, 1997, <http://www.opcfoundation.org>.
- [5] IVI Foundation, “*Interchangeable Virtual Instruments Standard*”, 1997, <http://www.ivifoundation.org/>.
- [6] ODAA, “*Open Data Acquisition Standard*”, 1998, <http://www.opendaq.org/>.
- [7] OMG, “*Data Acquisition from Industrial Systems (DAIS)*”, Request for Proposal (RFP), OMG Document: dtc/99-01-02, 1999, http://www.omg.org/techprocess/meetings/schedule/Data_Acquisition_RFP.html.
- [8] T. Nieva, “*Automatic Configuration for Remote Diagnosis and Monitoring of Railway Equipment*” presented at IASTED International Conference - Applied Informatics, Innsbruck, Austria, February 15-18, 1999, <http://icawww.epfl.ch/nieva/thesis/Conferences/ai99/article/ai99.pdf>.
- [9] T. Nieva, A. Fabri, and A. Benammour, “*Jini Technology Applied to Railway Systems*” presented at 2nd International Symposium on Distributed Objects and Applications (DOA'00), Antwerp, Belgium, September 21-23, 2000, <http://icawww.epfl.ch/nieva/thesis/Conferences/DOA00/article/DOA2000.pdf>.

- [10] T. Nieva and A. Wegmann, “*A Conceptual Model for Remote Data Acquisition Systems*” presented at 19th International Conference on Conceptual Modeling (ER'2000), Salt Lake City, Utah, USA, October 9-12, 2000, <http://icawww.epfl.ch/nieva/thesis/Conferences/ER00/Article/ER2000.pdf>.
- [11] IEE, “*The Millennium Problem in Embedded Systems*”, 2000, <http://www.iee.org.uk/2000risk/>.
- [12] T. Wireman, “*Computerized Maintenance Management Systems*”, Industrial Press, Inc, 1994.
- [13] A. Davies, “*Handbook of Condition Monitoring - Techniques and Methodology*”, Kluwer Academic Publishers, 1997.
- [14] Victorian Government, “*Asset Management Series: Principles, Policies and Practices*”, November, 1995, <http://home.vicnet.net.au/~assetman/welcome.htm>.
- [15] J. Ehrlich, A. Zerrouki, and N. Demassieux, “*Distributed Architecture for Data Acquisition: a Generic Model*” presented at IEEE Instrumentation and Measurement Technology Conference - IMTC'97, Ottawa, Canada, May 19-21, 1997.
- [16] Merriam-Webster, “*Webster Dictionary*”, 2000, <http://www.m-w.com/>.
- [17] P. H. Sydenham, “*Handbook of Measurement Science: Vol. 1 - Theoretical Fundamentals*”, John Wiley & Sons, 1982.
- [18] C. Alexander, “*The Timeless Way of Building*”, Oxford University Press, 1979.
- [19] C. Alexander, S. Ishikawa, and M. Silverstein, “*A Pattern Language : Towns, Buildings, Construction*”, Oxford University Press, 1977.
- [20] Hillside Group, “*Patterns Home Page*”, October, 1999, <http://hillside.net/patterns/>.
- [21] B. Appleton, “*Patterns and Software: Essential Concepts and Terminology*”, November 20, 1997, <http://www.enteract.com/~bradapp/docs/patterns-intro.html>.
- [22] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, “*Pattern - Oriented Software Architecture: A System of Patterns*”, Wiley, 1996.
- [23] R. Johnson, “*Frameworks Home Page*”, September, 1997, <http://st-www.cs.uiuc.edu/users/johnson/frameworks.html>.
- [24] M. Shaw and D. Garlan, “*Software architecture : perspectives on an emerging discipline*”, Prentice Hall, 1996.
- [25] J. P. Martin-Flatin, “*Web-based Management of IP Networks and Systems*”,

- PhD Thesis, EPFL, Lausanne, Switzerland, 2000, <http://ica2www.epfl.ch/~jpmf/papers/phd.pdf>.
- [26] ITS, “*Telecommunications: Glossary of Telecommunication Terms*”, Federal Standard, 1037C, Institute for Telecommunication Sciences, 1996, <http://www.its.bldrdoc.gov/fs-1037/>.
- [27] A. Wegmann, “*Object-Oriented Analysis and Design*”, Course Material, EPFL, Lausanne, Switzerland, 2000.
- [28] M. Boman, J. A. Bubenko Jr., P. Johannesson, and B. Wangler, “*Conceptual Modelling*”, Prentice Hall, 1997.
- [29] J. Rumbaugh, I. Jacobson, and G. Booch, “*The Unified Modelling Language Reference Manual*”, Addison Wesley, 1999, <http://www.rational.com>, <http://www.omg.org>.
- [30] M. Fowler and K. Scott, “*UML Distilled: Applying the Standard Object Modeling Language*”, Addison Wesley Longman, 1997.
- [31] G. Booch, “*Object-oriented analysis and design with applications*”, 2nd ed, Benjamin/Cummings Pub. Co., 1994.
- [32] J. Rumbaugh, “*Object-oriented modeling and design*”, Prentice Hall, 1991.
- [33] I. Jacobson, “*Object-oriented software engineering : a use case driven approach*”, Addison-Wesley, 1992.
- [34] OMG, “*UML Resource Page*”, September, 2000, <http://www.omg.org/uml/>.
- [35] ISO/IEC and ITU-T, “*Open Distributed Processing - Part1: Overview*”, Standard 10746-2, Recommendation X.901, 1995.
- [36] G. Genilloud, “*Common Objects in the RM-ODP Viewpoint Languages*” in Computer Standards and Interfaces vol. 19 (7), pp. 361-374, 1998.
- [37] J. Miller, “*Relationship of the UML to the Reference Model of Open Distributed Computing*”, September, 1997, <http://enterprise.shl.com/uml-odp/uml-odp.html>.
- [38] D. F. D'Souza and A. C. Wills, “*Objects, Components, and Frameworks with UML - The Catalysis Approach*”, Addison-Wesley, 1999.
- [39] L. Rising, “*The Pattern Almanac 2000*”, Addison-Wesley, 2000.
- [40] M. Fowler, “*Analysis Patterns: Reusable Object Models*”, Addison-Wesley, 1997, <http://www2.awl.com/cseng/titles/0-201-89542-0/apsupp/index.htm>.
- [41] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, “*Design Patterns - Elements of Reusable Object-Oriented Software*”, Addison-Wesley, 1995.
- [42] J. O. Coplien and D. C. Schmidt, “*Pattern Languages of Program Design*”,

- Addison-Wesley, 1995.
- [43] R. C. Martin, D. Riehle, and F. Buschmann, “*Pattern Languages of Program Design 3*”, Addison-Wesley, 1998.
- [44] N. Harrison, B. Foote, and H. Rohnert, “*Pattern Languages of Program Design 4*”, Addison Wesley, 2000.
- [45] J. M. Vlissides, J. O. Coplien, and N. L. Kerth, “*Pattern Languages of Program Design 2*”, Addison-Wesley, 1996.
- [46] Microsoft, “*The Component Object Model (COM)*”, 2000, <http://www.microsoft.com/com/tech/com.asp>.
- [47] Microsoft, “*Distributed COM (DCOM)*”, 2000, <http://www.microsoft.com/com/tech/dcom.asp>.
- [48] IVI Foundation, “*IVI Measurement and Stimulus Subsystems (IVI-MSS)*”, 2000, <http://www.ivifoundation.org/groups/MSS/>.
- [49] Digital Inc., “*EPFL Supervision - Information for the Superuser*”, Technical Manual, 1996.
- [50] Compaq Inc., “*BASEstar Product Suite - Real-time Factory Floor Data Integration Middleware*”, 2001, <http://www5.compaq.com/products/software/solutions/basestar/index.html>.
- [51] Gensym Inc., “*A Strategic Choice for Improving Business Operations*”, 2000, http://www.gensym.com/expert_operations/products/G2.htm.
- [52] M. P. de Albuquerque and E. Lelievre-Berna, “*Remote Monitoring over the Internet*” in Nuclear Instruments & Methods in Physics Research vol. 412 (1), pp. 140-145, 1998.
- [53] K. Kusunoki, I. Imai, H. Ohtani, T. Nakakawaji, M. Ohshima, and K. Ushijima, “*A CORBA-based Remote Monitoring System for Factory Automation*” presented at First International Symposium on Object-Oriented Real-time Distributed Computing - ISORC'98, Kyoto, Japan, April 20-22, 1998.
- [54] T. Lump, G. Gruhler, and W. Küchlin, “*Virtual Java Devices: Integration of Fieldbus Based Systems in the Internet*” presented at Annual Conference of the IEEE Industrial Electronics Society - IECON'98, Aachen - Germany, August 31 - September 4, 1998.
- [55] H. Kirrmann, “*ROSIN Data Representation and Notation*”, Part of the ROSIN WP4 Deliverable, 1996.
- [56] IEC, “*Train Communication Network - Part5: Train Network Management*”, IEC-61375-5, 1998.
- [57] K. Arnold, B. O'Sullivan, R. W. Scheifler, and J. Waldo, “*The Jini*

- specification*”, Addison-Wesley, 1999.
- [58] Jini User Group, “*Jini Home Page*”, 2000, <http://www.jini.org>.
- [59] Sun Microsystems, “*Jini Connection Technology Homepage*”, 2000, <http://www.sun.com/jini/>.
- [60] Sun Microsystems, “*The Java Tutorial*”, February, 2000, <http://java.sun.com/docs/books/tutorial/index.html>.
- [61] D. Flanagan, “*Java in a Nutshell*”, O'Reilly & Associates Inc., 1996.
- [62] Sun Microsystems, “*Java Remote Method Invocation White Paper*”, 1999, <http://java.sun.com/marketing/collateral/javarmi.html>.
- [63] R. Orfali, D. Harkey, and J. Edwards, “*The Essential Client/Server Survival Guide*”, 2nd ed, John Wiley & Sons Inc., 1996.
- [64] Microsoft, “*Microsoft Developer Network (MSDN) Online*”, 2000, <http://msdn.microsoft.com/>.
- [65] A. Vogel and K. Duddy, “*Java Programming with CORBA*”, John Wiley & Sons Inc., 1997.
- [66] OMG, “*OMG Home Page*”, 2000, <http://www.omg.org/>.
- [67] IEC, “*Electric Railway Equipment - Train Bus - Part 1: Train Communication Network*”, IEC 61375-1, 1999.
- [68] W3C, “*eXtensible Markup Language (XML) Home Page*”, 1997, <http://www.w3.org/XML/>.
- [69] C. F. Goldfarb and P. Prescod, “*The XML Handbook*”, Prentice Hall Inc., 1998.
- [70] N. Bradley, “*The XML Companion*”, Addison-Wesley Longman Limited, 1998.
- [71] J. Hunter and W. Crawford, “*Java Servlet Programming*”, O'Reilly & Associates Inc., 1998.
- [72] M. Mansouri-Samani and M. Sloman, “*Monitoring Distributed Systems*” in *Network and Distributed Systems Management*, Addison-Wesley, 1994.
- [73] A. Carzaniga, G. P. Picco, and G. Vigna, “*Designing Distributed Applications with Mobile Code Paradigms*” presented at 19th International Conference on Software Engineering (ICSE'97), Boston, Massachusetts, USA, 1997.
- [74] J. P. Martin-Flatin, “*Push vs. Pull in Web-based Network Management*” presented at 6th IFIP/IEEE International Symposium on Integrated Network Management (IM'99), Boston, MA, USA, May, 1999.
- [75] R. Motschnig-Pitrik and J. Kaasboll, “*Part-Whole Relationship Categories*

and Their Application in Object-Oriented Analysis” in IEEE Transactions on Knowledge and Data Engineering vol. 11 (5), pp. 779-797, 1999.

- [76] MSDN, “*Microsoft Windows and the Plug and Play Framework Architecture*”, 1994,
http://msdn.microsoft.com/library/backgrnd/html/msdn_pnp.htm.
- [77] UPnP Forum, “*Universal Plug and Play Home Page*”, 2000,
<http://www.upnp.org/>.
- [78] C. Larman, “*Applying UML and Patterns*”, Prentice Hall, 1997.

Curriculum Vitae

Txomin Nieva

Professional Address:

EPFL/DSC-ICA
1015 - Lausanne
Switzerland
E-mail: txomin.nieva@epfl.ch
Web: <http://icawww.epfl.ch/nieva>

Personal Address:

Ch. du Mottey 14
1020 - Renens
Switzerland
E-mail: nieva@ieee.org

Date of birth: 31.01.1972
Place of birth: San Sebastian (Spain)
Citizenship: Spanish

Gender: Male
Civil status: Single

EDUCATION

- 1997 - 2001** PhD on the topic "*Remote Data Acquisition of Embedded Systems using Internet Technologies: a Role-based Generic System Specification*" at the *Institute for computer Communications and Applications (ICA)* at the *Swiss Federal Institute of Technology Lausanne (EPFL), Lausanne (Switzerland)*.
- 1996 -1997** Graduate Thesis titled "*Ladder Diagram Monitoring for the PLC of Fagor 8050 CNC*" completed in the "*R+D Software*" department of *Fagor Automation S.Coop., Mondragon (Spain)*.
- 1994 - 1996** Systems Engineering degree from the *Mondragon University, Mondragon (Spain)*. Major studies on Systems Control, Signal Processing, Computer Science and Electrical Engineering.
- 1991 - 1994** Computer Science Engineering degree from the *Mondragon University, Mondragon (Spain)*.
- 1986 - 1991** Electrotechnis Specialist degree from the *I.P.C.L.Don Bosco technical school, Renteria (Spain)*.

TECHNICAL PUBLICATIONS

- 📖 T. Nieva, A.Wegmann. "*A Conceptual Model for Remote Data Acquisition Systems*". Proceedings of the 19th International Conference on Conceptual Modeling - ER'2000, Salt Lake City, Utah, USA, October 2000.

- 📖 T. Nieva, A.Fabri, A.Benammour. “*Jini Technology Applied to Railway Systems*”. Proceedings of the 2nd International Symposium on Distributed Objects and Applications - DOA'00, Antwerp, Belgium, September 2000.
- 📖 T. Nieva. “*NePESM: New Paradigms for Embedded Systems Management*”. Proceedings of Swiss Priority Programme for Information and Communications Structures – SPP-ICS Closing Conference, Fribourg, Switzerland, March 2000.
- 📖 T.Nieva, A.Fabri, P.Umiliachi. “*The Use of the Internet for Remote Train Monitoring and Control: the ROSIN Project*”. Proceedings of Rail Technology '99, London, UK, September 1999.
- 📖 T.Nieva. “*NePESM: New Paradigms for Embedded Systems Management*”. Informatik/Informatique, Volume 4, August 1999, pp. 38-39.
- 📖 T.Nieva. “*Automatic Configuration For Remote Diagnosis And Monitoring Of Railway Equipments*”. Proceedings of 17th IASTED International Conference - Applied Informatics, Innsbruck, Austria, February 1999, pp. 93-97.

WORK EXPERIENCE

- 1998 - 2000** System-Engineer (at 25%) in the *Information Technologies Dept.* at *ABB Corporate Research Ltd., Baden (Switzerland)*. Specially focused on web-based remote monitoring and diagnosis of embedded devices.
- 1997 – 1998** Stage at *ABB Corporate Research Ltd., Baden (Switzerland)*. Work related to the *RoMain* project (WP4 of *ROSIN* European Project).
- 1997 - 2001** Research-Assistant (at 75%) at the *Institute for Computer Communications and Applications (ICA)* at the *Swiss Federal Institute of Technology Lausanne (EPFL), Lausanne (Switzerland)*.
- 1996 - 1997** Computer-Assistant (at 50%) in the “*R+D Software*” department of *Fagor Automation S.Coop., Mondragon (Spain)*.

SKILLS IN COMPUTER SCIENCE

- Languages:** Java, UML, C/C++, HTML, XML/XSL, WML, VBScript, JavaScript, ASP, etc ...
- OSs:** Windows 95/NT, MS-DOS, UNIX.
- Software:** Symantec Cafe, Ms Office, Ms Visual C++, Ms Visual Interdev, Rational Rose , etc...

LANGUAGES

- Spanish:** Native language.
- Basque:** Fluent.
- French:** Fluent.
- English:** Fluent.
- German:** Beginner.

HOBBIES

Mountain bike, trekking, ski, traveling.