A Role-based Use Case Model for Remote Data Acquisition Systems*

Txomin Nieva, Alain Wegmann

Institute for computer Communications and Applications (ICA),
Communication Systems Department (DSC),
Swiss Federal Institute of Technology (EPFL),
CH-1015 Lausanne, Switzerland

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 paper we propose a role-based use case model of a generic DAS. This model gives DAS developers an abstraction of the generic functionalities of DASs; it enables them to compare existing 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. We have found that a role-based use case model of a generic system has many advantages. We propose patterns and techniques that are useful for the development of role-based use case models of generic systems.

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

1. Introduction

In the last few years, companies from many business areas have become increasingly interested in maintenance and asset management. Maintenance improves the reliability and availability of equipment and therefore the quality of service (QoS), which managers have found provides substantial benefits. According to Wireman (1994), maintenance management however makes up anywhere from 15 to 40% of total product cost. 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 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, 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. To apply condition monitoring, the access to utility data source is essential. Remote monitoring systems have been developed in many business areas such as

---

* Technical Report N° DSC/2001/031

building (e.g. Olken et al., 1998), power engineering (e.g. Itschner et al., 1998), and transportation systems (e.g. Fabri et al., 1999), 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. 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* data, converts *knowledge-level* and *operational-level* data to more useful system information and distributes this information to the user. There are many standards for DASs such as OLE for Process and Control (OPC) (OPC Foundation, 1997), Interchangeable Virtual Instrument (IVI) (IVI Foundation, 1997), and Open Data Acquisition Standard (ODAS) (ODAA, 1998), among others. Additionally, the Object Management Group (OMG) has recently issued a Data Acquisition from Industrial Systems (DAIS) Request For Proposal (RFP) (OMG, 1999a). 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. 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.

In this paper we propose a role-based use case model of a generic DAS. This model gives DAS developers an abstraction of the generic functionalities of DASs; it enables them to compare existing 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. We have found that a role-based use case model of a generic system has many advantages. We propose patterns and techniques that are useful for the development of role-based use case models of generic systems. Additionally, our role-based use case model of a generic DAS provides a case study of role-based use case modeling of generic systems that demonstrates, by means of an industrial example, the advantages of role-based use case modeling for the specification of generic systems.

This paper is organized as follows: In section 2, we explain the methodology we used to obtain a generic DAS role-based use case model. In section 3, we present the generic DAS role-based use case model. In section 4, we discuss key issues about the development of the generic DAS role-based use case model. In section 5, we explain the applications of a role-based use case model of a generic system. Finally, in section 6, we draw conclusions from the actual work.

2. Methodology

In Nieva and Wegmann (2001), we proposed a conceptual model of a generic DAS. This model gives DAS developers an abstraction of DASs; it enables them to compare existing 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. 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. Rumbaugh et al. (1999) defined the terms *actor* and *use case* as:

"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 and Scott (1997) gave a simpler definition of these terms as:

"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.

The role-based use case model presented in this paper is the result of an iterative process consisting of the object-oriented analysis, specification, implementation and deployment of a DAS for railway equipment (Fabri et al. ,1999 and Nieva, 1999). During this process, we used our own variation, which puts emphasis on role modeling, of the Catalysis development process, described by D'Souza and Wills (1999), based on the Unified Modeling Language (UML) (OMG, 1999b).

3. A Generic DAS Role-based Use Case Model: the 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 and elementary roles are fully specified by Nieva (2001).

In Figure 1, 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. 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-level* 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*.

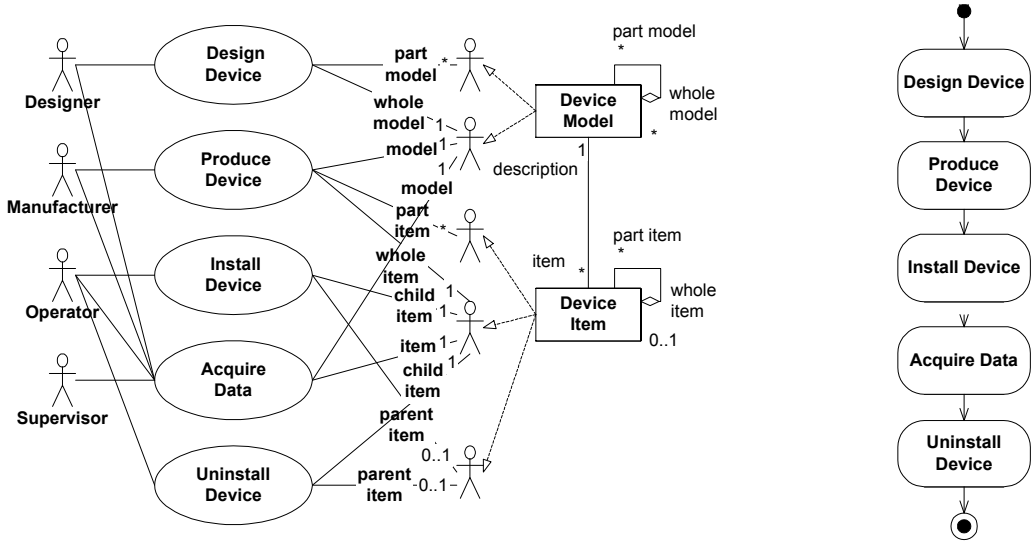


Figure 1 Device Lifecycle

In this paper we focus on the use case corresponding to the acquisition of data (*Acquire Data*). In Figure 2, 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 (OMG, 1999a) 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 section 4); and the *Administrator*, which takes care of the administration of the system.

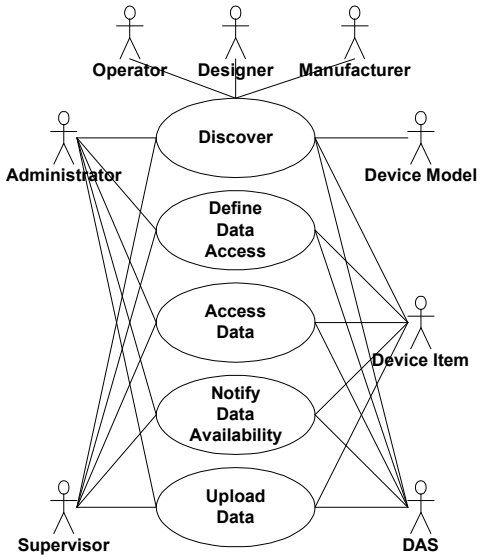


Figure 2 Data Acquisition Use Cases

We use an activity diagram, shown in Figure 3, to specify the sequence in which these use cases are carried out:

(a) *Discover device items*, *device models*, their *composition* and *knowledge-level* 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 data access 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*.

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. 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.

3.1 Discover

This use case implements the "discovery of remote system and device schema" functional requirement, which is defined by the OMG in the DAIS RFP (OMG, 1999a) 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."

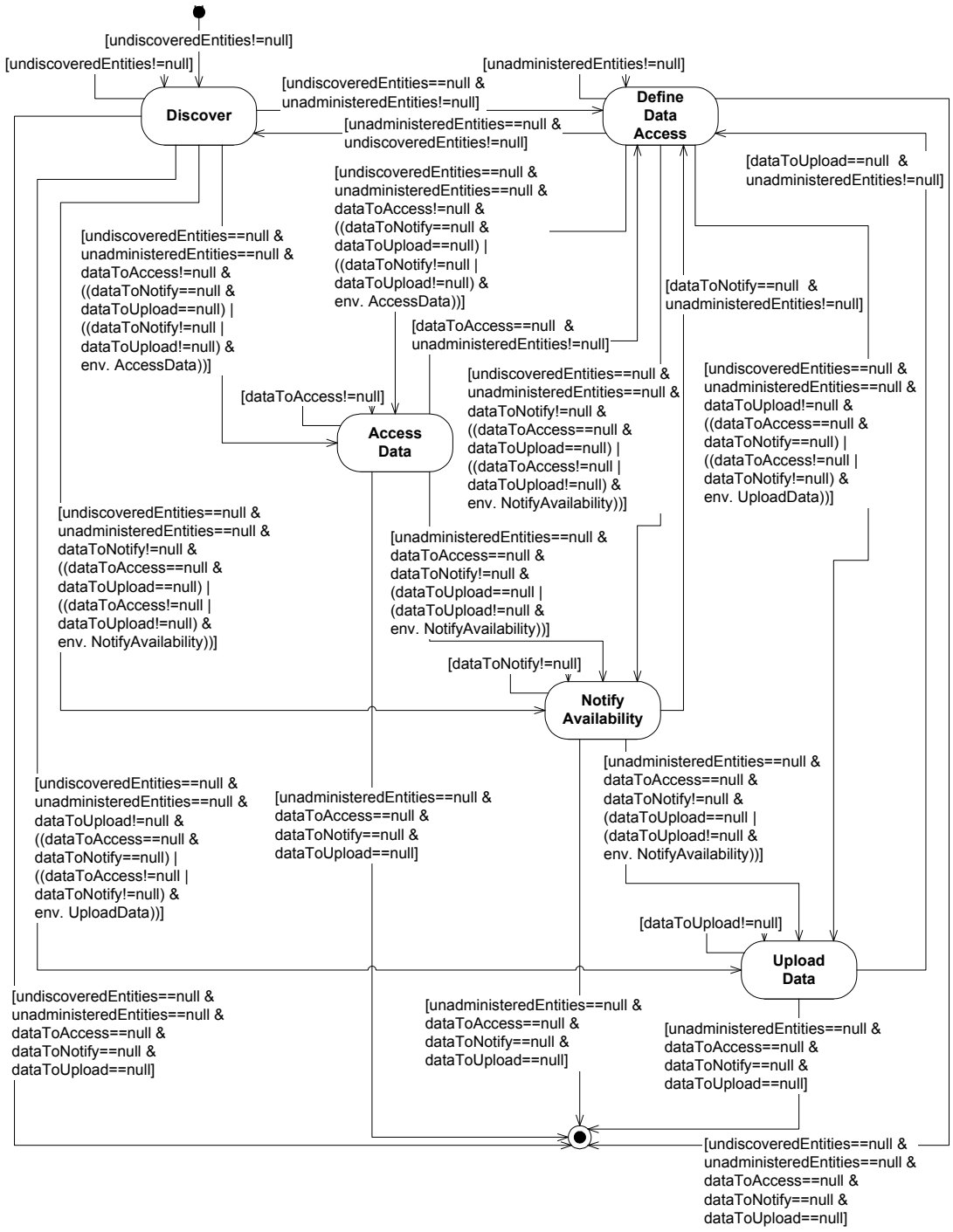


Figure 3 Data Acquisition Activity Diagram

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 4:

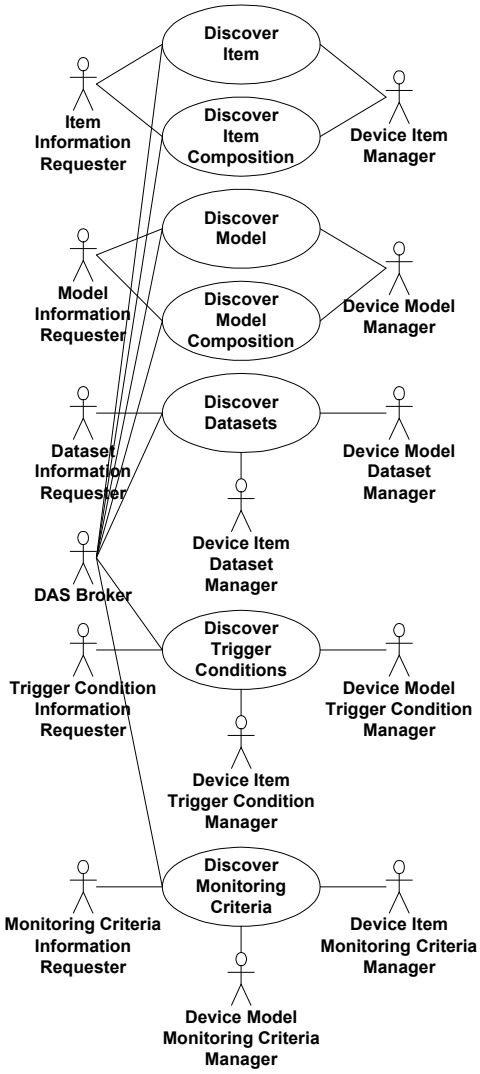


Figure 4 Discover Use Case

(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 a *device item*.

(iii) *Discover Model*. A *Model Information Requester* obtains, by request, the *device model* that characterizes a set of *device items*. 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 a *device model*.

(v) *Discover Datasets*. A *Dataset Information Requester* obtains, by request, the *datasets* defined on a *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 use case model more independent from decisions that should be taken later, in the design phase of a particular DAS. 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. We use an activity diagram, shown in Figure 5, 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.

3.2 Define Data Access

This use case implements the "defining data access request" functional requirement, which is defined by the OMG in the DAIS RFP (OMG, 1999a) 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*.

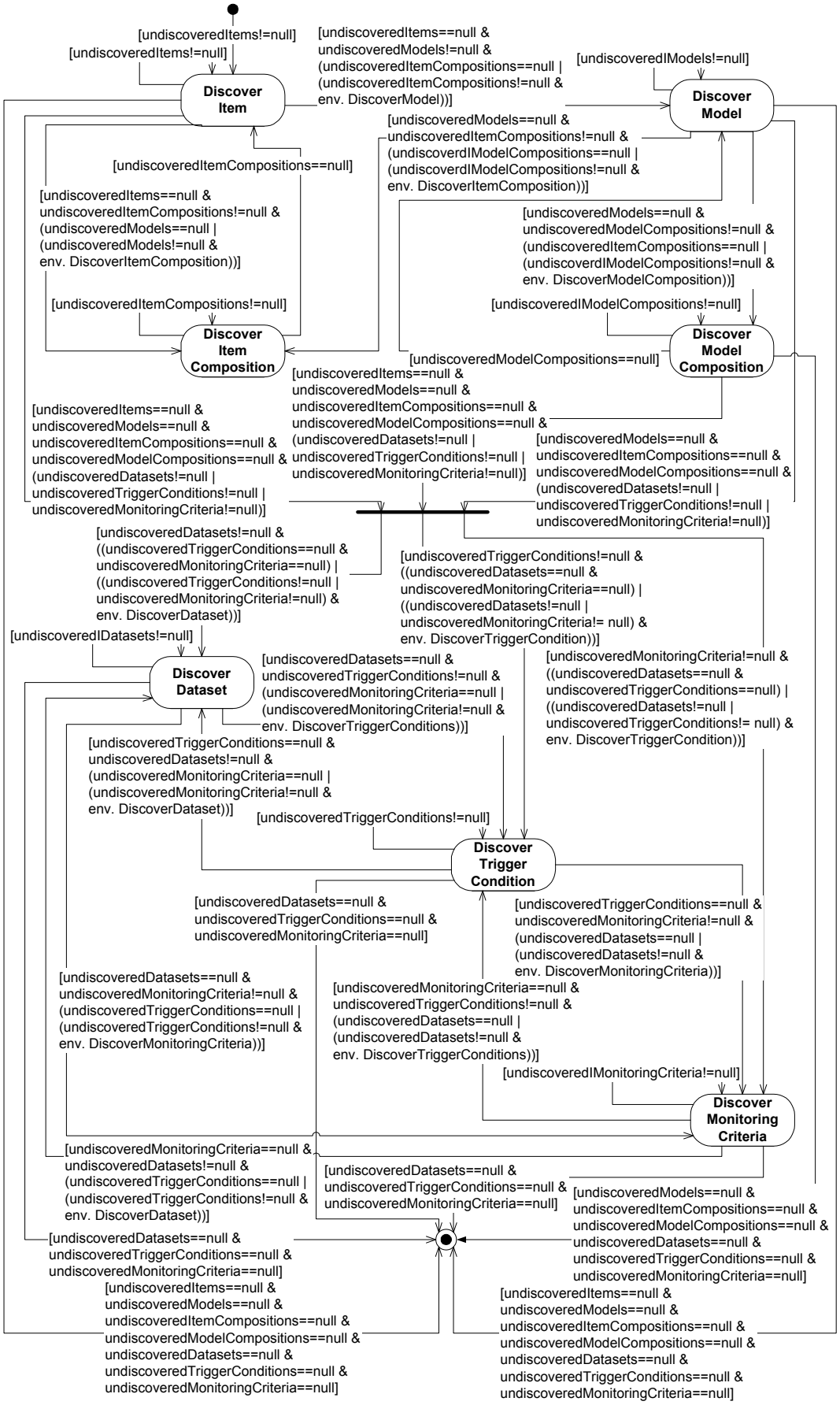


Figure 5 Discover Activity Diagram

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 a *time-based* condition to trigger the recording of the *observations*. Eventually, a *status monitoring criteria* can be associated with an *event trigger condition* instead of a *time trigger 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 (1999) discusses in detail these two paradigms, applied to web-based management. Defining data access must allow to 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 6:

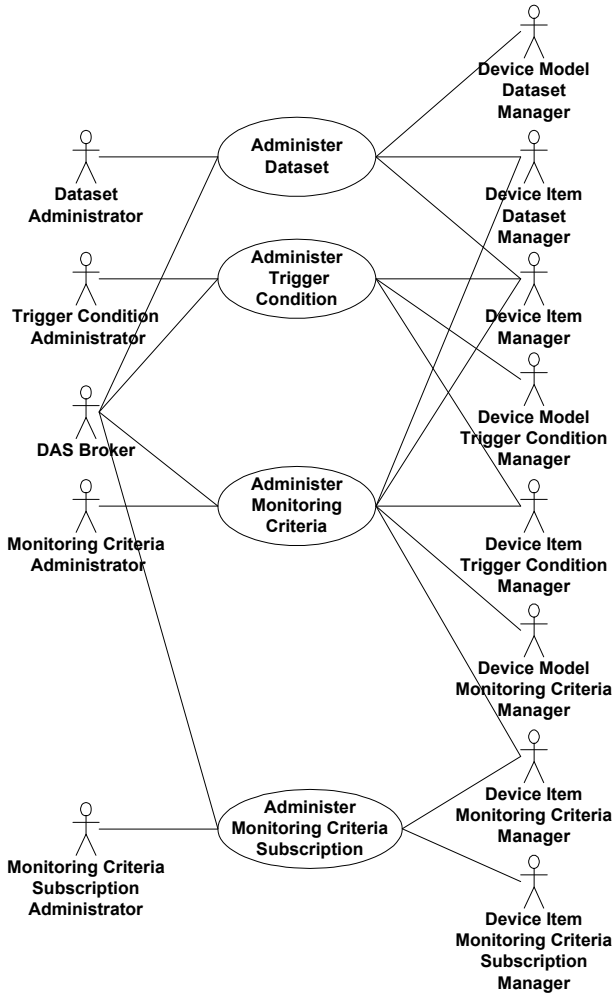


Figure 6 Define Data Access Use Case

(i) *Administer Datasets*. A *Dataset Administrator* administers (creates, modifies or removes) a *dataset* 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 *custom datasets* and *predefined 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 *custom trigger conditions* and *predefined 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 such *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 use case model 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 7, 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.

3.3 Access Data

This use case implements the "data access/retrieval" functional requirement, which is defined by the OMG in the DAIS RFP (OMG, 1999a) 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."

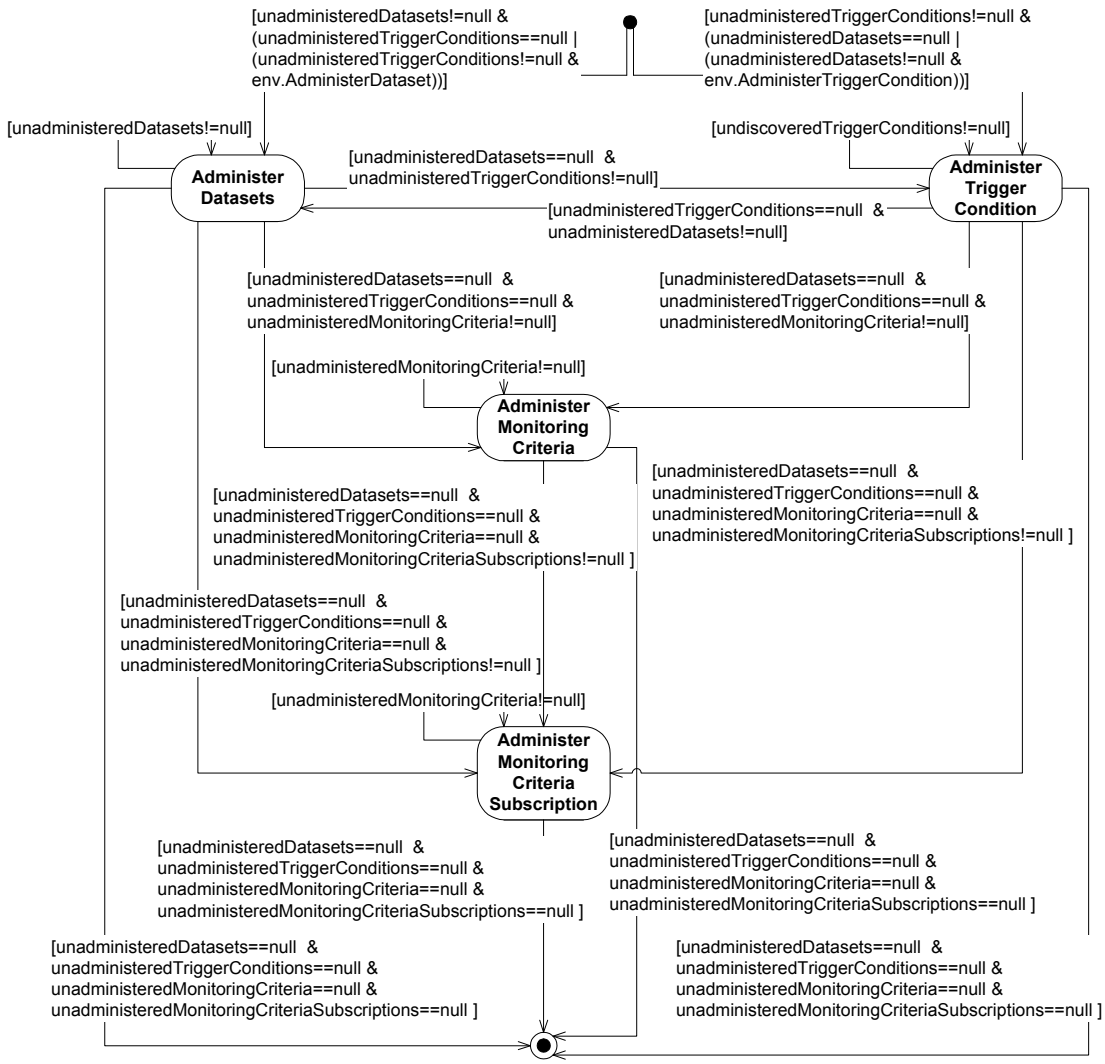


Figure 7 Define Data Access Activity Diagram

Data accessing allows *supervisors* to obtain, by request, the *current value* (a *quantitative measurement* or *qualitative measurement*) of a *measurement point,* or the *current monitoring reports* corresponding to certain *monitoring criteria* (e.g. the *current values* of a *dataset*). We refined the *Access Data* use case into the following role-based use cases, shown in Figure 8:

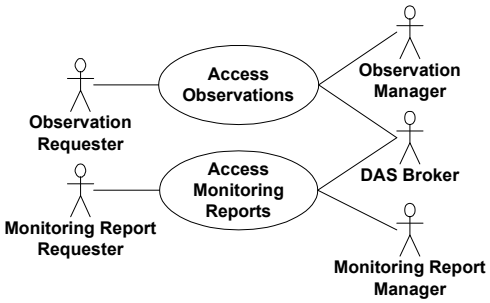


Figure 8 Access Data Use Case

(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 criteria, the monitoring reports of a certain criteria within a specific interval of time).

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 use case model 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 9, to specify the sequence in which these use cases may be carried out:

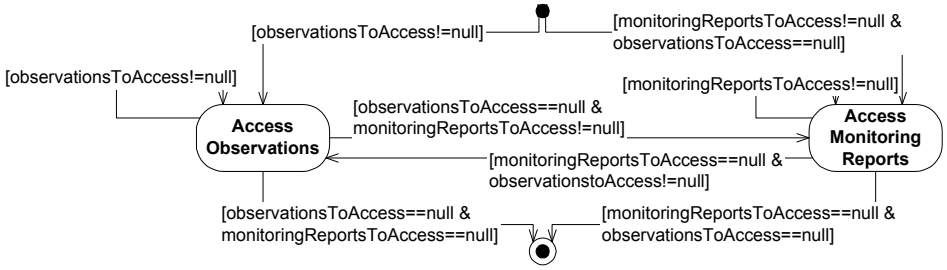


Figure 9 Access Data Activity Diagram

(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.

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 the DAIS RFP (OMG, 1999a) 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 are 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 10:

(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 use case model more independent from decisions that should be taken later, in the design phase of a particular DAS.

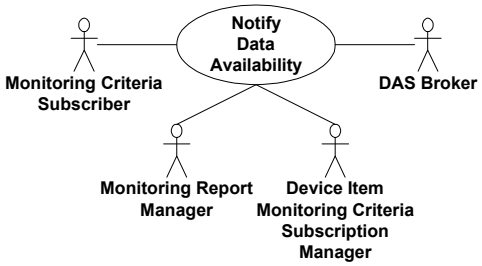


Figure 10 Notify Data Availability Use Case

3.5 Upload Data

This use case implements the "event driven data upload" functional requirement, which is defined by the OMG in the DAIS RFP (OMG, 1999a) 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 11:

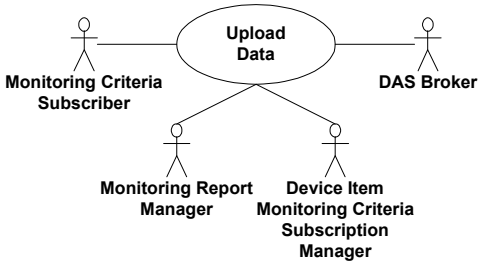


Figure 11 Upload Data Use Case

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

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 use case model more independent from decisions that should be taken later, in the design phase of a particular DAS.

4. A Generic DAS Conceptual Model: Discussion

In this section we discuss key issues about the development of our generic DAS role-based use case model. 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.

4.1 Representation of the System

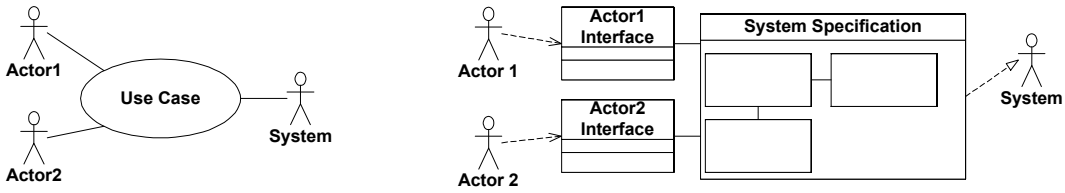
People use different approaches to represent the system in the use cases. These approaches are mainly:

(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 12.



(a) Use Case with the System

(b) System Specification

Figure 12 Representation of the System in the Use Case

4.2 Elementary Roles vs. Actors

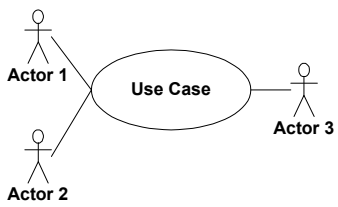
We used actors to represent not only outside users but also the system itself. Additionally, actors can play several elementary roles in use cases. Therefore, an actor can be seen as a composition of elementary roles, each of them playing a single function in a use case. We used elementary roles rather 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 13. 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.

4.3 System Behavior Modeling

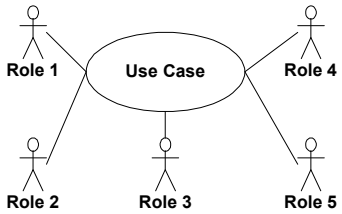
A use case model of a generic system must allow the specification of different system behaviors depending on specific execution constraints and types of control flow. In this section we give some techniques to specify the interactions across use cases and the scenarios of use cases in such a way that the use case model 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 (OMG, 1999b) 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). A use case model of a generic system must be able to specify

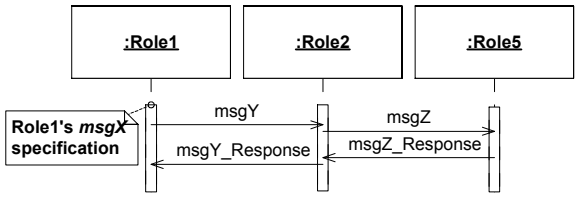
many particular systems with different interactions across use cases depending on particular execution constraints. To achieve this, we propose to:



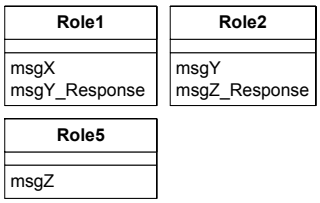
(a) Use Cases with Actors



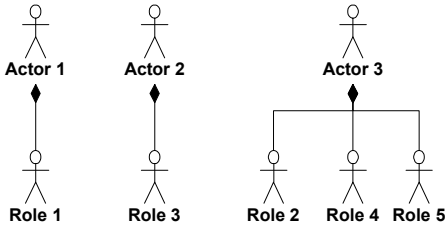
(b) Use Cases with Roles



(c) Scenario Specification with Roles



(d) Roles Interfaces



(e) Mapping of Roles into Actors

Figure 13 Elementary Roles vs. Actors

(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 14, 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.

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 (OMG, 1999b) 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. A use case model 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`.

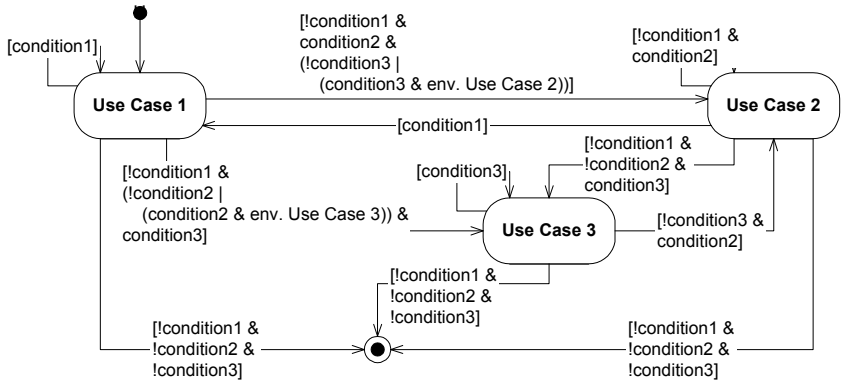


Figure 14 Example of Modeling of the Interactions across Use Cases

Example: as shown in Figure 15, 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.

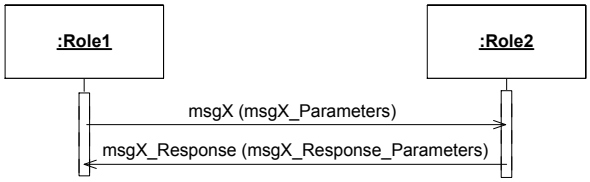


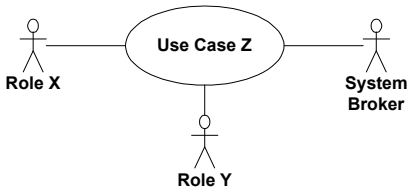
Figure 15 Example of Modeling of the Scenarios of Use Cases

4.4 Broker Pattern

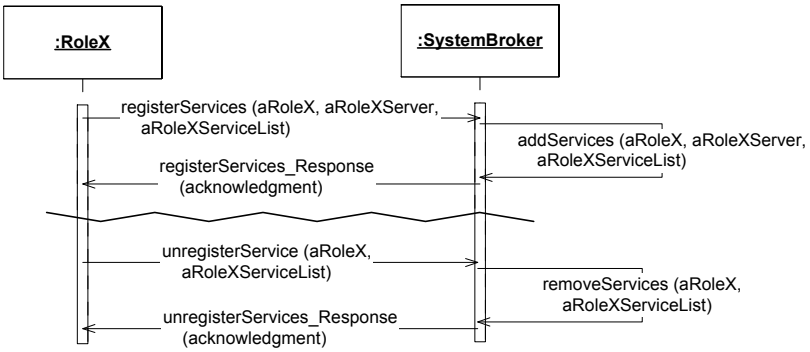
The *Broker* pattern is defined by Buschmann et al. (1996):

"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".

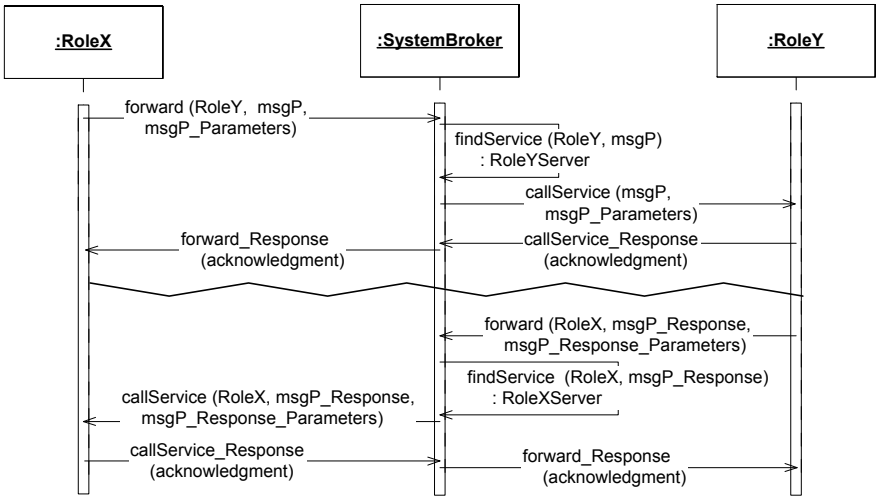
We made a wide use of this pattern in our specification of a generic DAS. Due to the genericness of our specification we cannot predict if a role will be implemented internally by a component of the system, or an outside user or system. The *System Broker* allows us to decouple roles allowing us to remain 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 16a. Roles register on the *System Broker* to handle certain messages, as shown in Figure 16b. 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 16c we show an example of communication 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 16d.



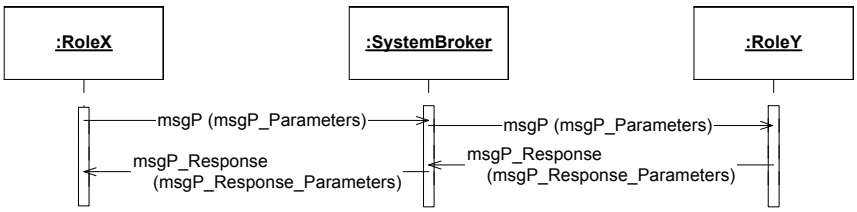
(a) Example of Representation of the System Broker in the Use Cases



(b) Example of (Un)Registration of Services in the System Broker



(c) Example of Comm. between Roles using the System Broker



(d) Example of Simplified Comm. between Roles using the System Broker

Figure 16 The Broker Pattern

4.5 Administrator-Manager Pattern

The so-called *Administrator-Manager* use case pattern is shown in Figure 17. 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. Some examples of the utilization of this pattern can be found in the refined use cases, specified by Nieva (2001), of the *Define Data Access* use case.

Figure 17 Administrator-Manager Pattern

4.6 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 Larman (1997). By one way, use cases are classified as primary, secondary or optional: primary use cases represent major common processes, secondary use cases represent minor processes and optional use cases represent processes that many not be tackled. By other way, use cases are classified as essential or real: essential use cases are expressed in an ideal form that is implementation independent, whereas real use cases describe the process in terms of a specific design. In this thesis we focused only on primary and essential 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 object 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.

4.7 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 *Myself*. 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.

5. Application and Validation

In this chapter we explain the applications of a role-based use case model of a generic system. The most direct application of a role-based use case model is for the writing of a RFP for a new standard. Another potential application of a role-based use case model is for the evaluation of existing systems, standards or RFP responses. We illustrate this by using our generic DAS role-based use case model to compare the different DAS standards. Finally, a role-based use case model can be used 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 using our generic DAS role-based use case model.

5.1 Issuing/Replying a RFP

This is probably the most direct application of a role-based use case model of a generic system. *RFP Issuers* may use a role-based use case model of a generic system as a guide to specify the functionalities that such a standard must deal with, creating and writing down a specific RFP. *RFP Repliers* may use a role-based use case model of a generic system to easier understand and analyze the requirements of this RFP. Additionally, *RFP Repliers* can use a role-based use case model of a generic system to describe their proposal of standard in request to this RFP. In this way a role-based use case model of a generic system acts as an efficient communication mechanism between *RFP Issuers* and *RFP Repliers*, facilitating the creation, writing, understanding and replying of a RFP. A role-based use case model of a generic system can be seen as an actor that actively collaborates in all these actions, as shown in Figure 18.

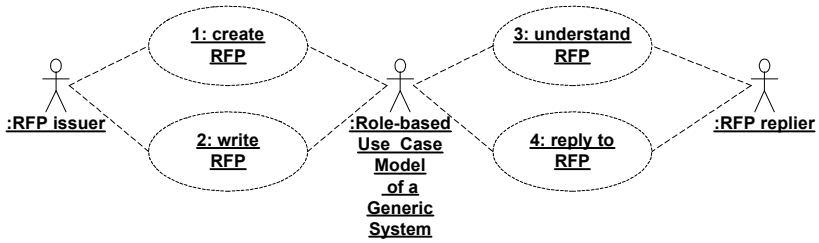


Figure 18 Write New RFP

5.2 Evaluation of Existing Systems or Proposals

Another potential application of a role-based use case model of a generic system is the evaluation of existing systems or standards. Developers may use a role-based use case model of a generic system to check the functionalities that these systems or standards support. In this section, as an example, we use the role-based use case model of a generic DAS that we developed to compare the DAS standards: OPC, IVI and ODAS. The results of these comparisons are shown in Table 1.

OPC. OPC specifies all the functionalities defined in the generic DAS specification except *Discover Model*, as OPC does not support the notion of *model*.

IVI. IVI specifies all the functionalities defined in the role-based use case model of a generic DAS.

ODAS. ODAS specifies all the functionalities defined in the generic DAS specification except *Discover Model*, as ODAS does not support the notion of *model*.

Table 1 Generic DAS Functionality Comparison

Generic DAS Functionality	OPC (v2)	IVI	ODAS
Discover			
Discover Composition	✓	✓	✓
Discover Model	✗	✓	✗
Discover Datasets	✓	✓	✓
Discover Trigger Conditions	✓	✓	✓
Discover Monitoring Criteria	✓	✓	✓
Define Data Access			
Administer Dataset	✓	✓	✓
Administer Trigger Condition	✓	✓	✓
Administer Monitoring Criteria	✓	✓	✓
Administer Monitoring Criteria Subscription	✓	✓	✓
Access Data			
Access Observations	✓	✓	✓
Access Monitoring Reports	✓	✓	✓
Notify Data Availability			
Notify Data Availability	✓	✓	✓
Upload Data			
Upload Data	✓	✓	✓

5.3 Design of a New System

A role-based use case model of a generic system 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 role-based use case model of a generic system can also be used as a starting point for the design of a particular system. As a result, the use of a role-based use case model of a generic system 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.

Development of a DAS for Railway Equipment. The main objective of this development was to validate our role-based use case model of a generic DAS 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 role-based use case model of a generic system. 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 role-based use case model of a generic DAS. We first analyzed our generic DAS conceptual model to obtain a DAS conceptual model specialized in the context of railway equipment. 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 knowledge-level* data 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 19, we present the resulting high-level system use cases. 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.

The role-based use case model of a generic DAS had a significant role in the design of the DAS railway equipment. We used the role-based use case model of a generic DAS to clearly specify the functionalities that the DAS for railway equipment should implement. We used the elementary roles to specify some of the components of the system. As a result, we had an implementation where a significant amount of the designed classes represent elementary roles specified in the role-based use case model of a generic DAS. The development of this particular DAS demonstrated, by means of an industrial example, the usefulness of a role-based use case model of a generic system for the development of a particular system.

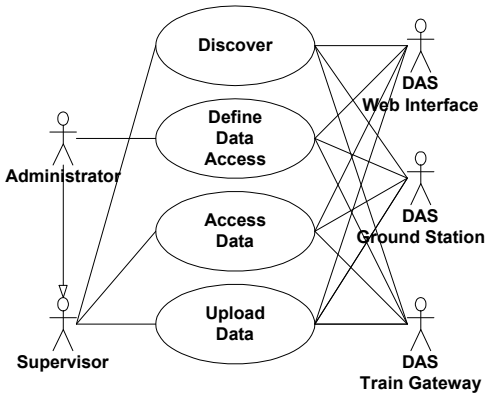


Figure 19 Railway Equipment DAS System Use Case Model

6. Conclusions

In this paper, we described a role-based use case model of a generic DAS. This model gives DAS developers an abstraction of the generic functionalities of DASs; it enables them to compare existing 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. We have found that a role-based use case model of a generic system has many advantages. We propose patterns and techniques that are useful for the development of role-based use case models of generic systems.

The most direct application of a role-based use case model of a generic system is for the writing of a RFP for a new standard. Another potential application of a role-based use case model of a generic system is for the evaluation of existing systems, standards or RFP responses. We illustrated this by using our role-based use case model of a generic DAS to compare the different DAS standards. Finally, a role-based use case model of a generic system 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 role-based use case model of a generic DAS. This development demonstrates, by means of an industrial example, the usefulness of a role-based use case model of a generic system for the development of a particular system.

The role-based use case model of a generic DAS specifies the behavior of a generic DAS. This role-based use case model of a generic DAS complements the generic DAS conceptual model described by Nieva and Wegmann (2001). Together, these models provide a complete specification of a generic DAS. 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.

References

[Buschmann et al., 1996] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., and Stal, M., "*Pattern - Oriented Software Architecture: A System of Patterns*", Wiley, 1996.

[D'Souza and Wills, 1999] D'Souza, D. F. and Wills, A. C., "*Objects, Components, and Frameworks with UML - The Catalysis Approach*", Addison-Wesley, 1999.

[Fabri et al., 1999] Fabri, A., Nieva, T., and Umiliacchi, P., "*Use of the Internet for Remote Train Monitoring and Control: the ROSIN Project*" presented at Rail Technology '99, London, UK, 1999, http://icawww.epfl.ch/nieva/thesis/Conferences/RailTech99/article/RailTech99.PDF.

[Fowler and Scott, 1997] Fowler, M. and Scott, K., "*UML Distilled: Applying the Standard Object Modeling Language*", Addison Wesley Longman, 1997.

[Itschner et al., 1998] Itschner, R., Pommerell, C., and Rutishauser, M., "*GLASS: Remote Monitoring of Embedded Systems in Power Engineering*" in IEEE Internet Computing, vol 2, 1998.

[IVI Foundation, 1997] IVI Foundation, "*Interchangeable Virtual Instruments Standard*", 1997, http://www.ivifoundation.org/.

[Larman, 1997] Larman, C., "*Applying UML and Patterns*", Prentice Hall, 1997.

[Martin-Flatin, 1999] Martin-Flatin, J. P., "*Push vs. Pull in Web-based Network Management*" presented at 6th IFIP/IEEE International Symposium on Integrated Network Management (IM'99), Boston, MA, USA, 1999.

[Nieva, 1999] Nieva, T., "*Automatic Configuration for Remote Diagnosis and Monitoring of Railway Equipment*" presented at IASTED International Conference - Applied Informatics, Innsbruck, Austria, 1999, http://icawww.epfl.ch/nieva/thesis/Conferences/ai99/article/ai99.pdf.

[Nieva and Wegmann , 2001] Nieva, T. and Wegmann, A. "*A Conceptual Model for Remote Data Acquisition Systems*", Technical Report N°030, EPFL, DSC, Lausanne, 2001.

[Nieva, 2001] Nieva, T., "*Remote Data Acquisition of Embedded Systems Using Internet Technologies: a Role-based Generic System Specification*", PhD Thesis N°2388, EPFL, DI, Lausanne, 2001, http://icawww.epfl.ch/nieva/thesis/report/phd.pdf.

[ODAA, 1998] ODAA, "*Open Data Acquisition Standard*", 1998, http://www.opendaq.org/.

[Olken et al., 1998] Olken, F., Jacobsen, H. A., McParland, C., Piette, M. A., and Anderson, M. F., "*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, 1998, http://www.lbl.gov/~olken/rbo/rbo.html.

[OMG, 1999a] OMG, "*Data Acquisition from Industrial Systems (DAIS)*", Request for Proposal (RFP), OMG Document: dtc/99-01-02, January 15, 1999a, http://www.omg.org/techprocess/meetings/schedule/Data_Acquisition_RFP.html.

[OMG, 1999b] OMG, "*Unified Modeling Language Specification Version 1.3*", June, 1999b, http://www.omg.org.

[OPC Foundation, 1997] OPC Foundation, "*OLE for Process and Control Standard*", 1997, http://www.opcfoundation.org.

[Rumbaugh et al., 1999] Rumbaugh, J., Jacobson, I., and Booch, G., "*The Unified Modelling Language Reference Manual*", Addison Wesley, 1999, http://www.rational.com, http://www.omg.org.

[Wireman, 1994] Wireman, T., "*Computerized Maintenance Management Systems*", Industrial Press, Inc, 1994.