PIMRep

Platform Independent Model Repository for a service oriented infrastructure

Hans Stubberud

Cand Scient Thesis

submitted in partial fulfillment for the Cand Scient degree in Informatics at

the Department of Informatics,
Faculty of Mathematics and Natural Sciences,
University of Oslo

January 2004
This master thesis is submitted in partial fulfillment for the Cand. Scient. degree in Informatics at the Department of Informatics, University of Oslo (UiO). The work on this thesis was done at SINTEF, Institute of Telecom and Informatics, Department of Distributed Information Systems. I would like to thank SINTEF for letting me write this thesis.

I would especially like to thank my advisor Dr. Arne-Jørgen Berre for his guidance and help throughout the whole process. Furthermore, I would like to thank family, friends and fellow students for being supportive all this time.
Abstract

This thesis is concerned with finding a repository solution for software engineering in a service oriented infrastructure. As a context we describe two problem examples that need some kind of software repository.

Our solution is called PIMRep – Platform Independent Model Repository for a service oriented infrastructure. The PIMRep Architecture consists of two system parts; PIMRep External and PIMRep Internal.

In this thesis we give thorough descriptions of the information model and services specification of both PIMRep External and PIMRep Internal. We also apply them to a case study as a proof-of-concept.

PIMRep External is an application for sharing software documents over the Internet. It allows you to register contact information, it has a sophisticated model for access control, it allows for subscription to event notifications, it allows for classification according to all kinds of schemes, it stores PIM\(^1\)'s, PSM\(^2\)'s and realization documents, it allows for free text searches and browsing by category, and it can distribute these queries to other instances of PIMRep External.

PIMRep Internal allows for access to elements within models. It lets you create, read, update and delete UML modeling elements. It also lets you browse and search for such elements. PIMRep Internal implements a fine grained access control to modeling elements, and it also allows for versioning.

---

\(^1\) Platform Independent Model
\(^2\) Platform Specific Model
Table of contents

1. INTRODUCTION ........................................................................................................ 1
   1.1 Definitions................................................................................................................ .. 5
   1.2 Structure of this thesis.............................................................................................. 7
   1.3 Summary ................................................................................................................... 8

2. PROBLEM EXAMPLES ............................................................................................ 9
   2.1 Infrastructure of Norwegian e-commerce.............................................................. 9
      2.1.1 The use of registry and repository in an open infrastructure in electronic
           collaboration................................................................. 10
          2.1.1.1 Scenarios for using the registry/repository ........................................... 10
          2.1.1.2 Requirements to the contents of the registry/repository ...................... 12
          2.1.2 Summary ................................................................................................. 15
       2.1.2 ACE-GIS.................................................................................................................. 15
          2.2.1 The Modusa gas dispersion Web service. ......................................................... 16
                   2.2.1.1 Limitations ................................................................................................. 19
          2.2.2 Platform Independent Model (PIM) of the Modusa Web service..................... 19
          2.2.3 Platform Specific Model (PSM) of the Modusa Web Service........................................... 21
          2.2.4 Summary ........................................................................................................... 23

3. REQUIREMENTS ..................................................................................................... 24
   3.1 External repository................................................................................................. 25
   3.2 Internal repository.................................................................................................. 27
   3.3 Summary ................................................................................................................. 28

4. EXISTING TECHNOLOGIES................................................................................. 29
   4.1 UDDI........................................................................................................................ 30
       4.1.1 UDDI Data Structure......................................................................................... 31
       4.1.2 Functionality and services of UDDI................................................................. 33
          4.1.2.1 Inquiry API functions ................................................................................ 33
          4.1.2.2 Publishing API functions ......................................................................... 34
          4.1.2.3 Subscription API functions ......................................................................... 35
          4.1.2.4 Registry interaction.................................................................................. 35
       4.1.3 Summary ........................................................................................................... 35
   4.2 ebXML Registry/Repository.................................................................................. 36
       4.2.1 Registry Information Model.............................................................................. 36
       4.2.2 Registry Services Specification......................................................................... 38
       4.2.3 Summary ........................................................................................................... 41
   4.3 OGC Registry........................................................................................................... 41
       4.3.1 OGC Registry interfaces .................................................................................. 42
          4.3.1.1 WRSQuery interface ................................................................................ 42
          4.3.1.2 WRSTransaction interface ....................................................................... 43
11. PIMREP INTERNAL APPLIED TO MODUSA GAS DISPERSION WEB SERVICE .......................................................... 100
   11.1 Summary ........................................................................................................ 102
12. EVALUATION OF PIMREP INTERNAL .................................................................. 103
13. CONCLUSIONS AND FUTURE WORK ................................................................. 105
   13.1 Contributions .................................................................................................. 105
   13.2 Future work ................................................................................................... 106
A. MOF ..................................................................................................................... 107
   A.1 Common Superclasses ...................................................................................... 107
   A.2 Containment Hierarchy .................................................................................. 108
   A.3 Types ............................................................................................................... 109
      A.3.1 Class ............................................................................................................. 110
      A.3.2 Association ................................................................................................. 110
      A.3.3 Datatypes ................................................................................................... 111
   A.4 Features .......................................................................................................... 112
   A.5 Tags ................................................................................................................. 112
REFERENCES ........................................................................................................... 114
Figure 1-1 Uploading model.................................................................................................. 2
Figure 1-2 Downloading model............................................................................................. 2
Figure 1-3 Finding model...................................................................................................... 3
Figure 1-4 Viewing graphical rendering of model ............................................................... 4
Figure 1-5 Viewing generated code....................................................................................... 5
Figure 2-1 Typical use of an ebXML registry/repository.................................................... 11
Figure 2-2 Infrastructure project architecture...................................................................... 12
Figure 2-3 The Modusa Web service is a compound service.............................................. 17
Figure 2-4 The order in which the Modusa Web service invokes the other services........ 18
Figure 2-5 An example client.............................................................................................. 19
Figure 2-6 Modusa PIM ...................................................................................................... 21
Figure 2-7 Modusa PSM ...................................................................................................... 22
Figure 2-8 Modusa PSM ...................................................................................................... 23
Figure 4-1 The five data types that make up a UDDI registration ...................................... 31
Figure 4-2 Overview of the ebXML Registry information model ....................................... 37
Figure 4-3 Inheritance view of the classes inheriting from RegistryObject......................... 38
Figure 4-4 The typical lifecycle of a RegistryObject............................................................ 39
Figure 4-5 The Submit Objects Protocol ........................................................................... 39
Figure 4-6 A client submitting an ad hoc query ............................................................... 40
Figure 4-7 Content validation.............................................................................................. 40
Figure 4-8 Content cataloging............................................................................................. 41
Figure 4-9 OGC Registry service ....................................................................................... 42
Figure 4-10 WRSQuery ..................................................................................................... 42
Figure 4-11 WRSTransaction ............................................................................................ 43
Figure 4-12 The MDA logo............................................................................................... 44
Figure 4-13 An overview of the MOF model...................................................................... 45
Figure 4-14 SimpleTrader meta-model ............................................................................ 46
Figure 4-15 Generating a MOF Metadata Repository ......................................................... 48
Figure 4-16 Workbench GUI............................................................................................. 50
Figure 4-17 SimpleRDB model ....................................................................................... 50
Figure 4-18 The SimpleRDB Metadata Server ................................................................. 52
Figure 5-1 UDDI server hosted at e-blana ......................................................................... 55
Figure 5-2 Searching for CompanyA ............................................................................... 57
Figure 5-3 Retrieving item ............................................................................................... 58
Figure 5-4 Downloading file ............................................................................................ 59
Figure 5-5 askthespider.com ........................................................................................... 61
Figure 6-1 Illustration of communication between servers and clients......................... 66
Figure 6-2 Client side infrastructure ............................................................................... 67
Figure 7-1 Views of the PIMRep External information model .......................................... 69
Figure 7-2 Core elements view of the PIMRep information model ................................... 71
Figure 7-3 Association between NAICS1997 and NAICS2001 ..................................... 73
List of tables

Table 3-1 Requirements for the external repository .......................................................... 26
Table 3-2 Requirements for the internal repository .......................................................... 28
Table 4-1 Inquiry API functions ....................................................................................... 34
Table 4-2 Publishing API functions ................................................................................ 35
Table 4-3 Subscription API functions ............................................................................. 35
Table 4-4 WRSQuery ....................................................................................................... 43
Table 4-5 WRSTransaction .............................................................................................. 44
Table 5-1 Evaluation of UDDI ....................................................................................... 56
Table 5-2 Evaluation of ebXML Registry/Repository ....................................................... 59
Table 5-3 Evaluation of dMOF ....................................................................................... 63
Table 5-4 Evaluation of the Reference Implementation of JMI ........................................ 63
Table 6-1 Legend for figure 6-1 ...................................................................................... 66
Table 7-1 Description of the classes of the Core package .............................................. 73
Table 7-2 Description of the classes of the AuditTrail package .................................... 76
Table 7-3 Method summary of LifeCycleManager ......................................................... 81
Table 7-4 Method summary of QueryManager ............................................................... 82
Table 7-5 Method summary of LinkManager ................................................................. 83
Table 9-1 Evaluation of PIMRep External .................................................................... 94
Table 10-1 Explanation of functions of IEasy ................................................................. 96
Table 12-1 Evaluation of PIMRep Internal ................................................................. 103
1. Introduction

The title of this thesis is PIMRep – Platform Independent Model Repository for a service oriented infrastructure. It is an application or specification, or at least an outline for such, for storing and retrieving software models. Why do we want to store and retrieve software models? We believe in something called Model Driven Architecture (MDA), which is an initiative by the object Management Group (OMG). MDA suggests that models should be the heart and soul of software engineering. A model is no longer something you sketch down on a piece of paper five minutes before starting to code on a major software project. The model should be in sync with the code at all times during the project, and it is to be expressed in the Unified Modeling Language (UML).

MDA talks about two kinds of models, Platform Independent Models (PIMs) and Platform Specific Models (PSMs). PIMs have no modeling elements that are specific to any platform, such as J2EE, CORBA or Web services. PSMs, on the other hand, include elements that make seamless transformation between model and code possible. The idea behind this division, is that when modeling a business, you do not want to concern yourself with technological issues specific to one platform. Not at that point. Technology platforms come and go as time passes by. The effort invested in modeling a platform independent solution is done only once and then it is kept in a PIM forever. It can, however, be mapped to as many PSMs as you will need.

Also, once a mapping to a certain platform is made, you can also transform PSMs back to PIMs. That is, you can reverse engineer a PSM to discover the underlying PIM. Then you can redeploy this PIM to a new platform. Bottom line, you do not have to remodel the business or rediscover the wheel for that matter, every time technology changes.

Obviously, one needs some way to store and retrieve all this models. That is where PIMRep comes in. The most obvious place for PIMRep is in a service oriented infrastructure, which is also suggested in the title. For example, if a business wants to commit e-commerce with some of its partners, it would typically need to expose the external interfaces of its software systems. If this is done in a model driven way, both PIMs and PSMs of the interfaces would be made available. The PSMs would be used by another company for generating code for accessing the systems of the business in question. At the same time the publicly available PIMs are mapped to the platform of the other companies’ legacy systems. In this way integration between enterprises is made seamlessly.

In the following we will present a few use cases that we initially had in mind for PIMRep. We will see that some of them are supported by PIMRep, while some of them need further work. We will also introduce a separation of PIMRep into two applications: PIMRep Internal and PIMRep External.

Figure 1-1 shows a use case where a developer is uploading a model to the repository. The model is in the format of UML, because this is the standard way of presenting software models. Also, all models that I have developed in this thesis to describe PIMRep are
written in UML according to the UML specification [1] and Martin Fowler’s book *UML Distilled* [2]. Notice also that the UML model is uploaded as an XML Metadata Interchange (XMI) file. XMI is the standard non proprietary way of serializing UML models. It is actually a mapping from the Meta Object Facility (MOF) to XML. For more information on XMI please see [3].

The name of the actor in figure 1-1 is Developer. That is because the files being uploaded to PIMRep are software documents, and thus the users will typically be software developers.

![Figure 1-1 Uploading model](image)

**Figure 1-1 Uploading model**

The use case in figure 1-2 is the opposite of that in figure 1-1. Figure 1-2 shows a developer downloading a UML model.

![Figure 1-2 Downloading model](image)

**Figure 1-2 Downloading model**

But how is the developer going to find the model to download? Figure 1-3 depicts the use case of finding a UML model. The *Find UML model* use case has two sub-use cases that include *Browse for UML model* and *Search for UML model*. Browsing means that you look for a model in certain categories. These categories can for instance be within the domains of geography or business function. Searching requires the user to type in the title, description or some other property of a model.
The use cases presented this far are typical use cases of any kind of registry. Therefore we have investigated three existing registry specifications and implementations. These include the Universal Description, Discovery and Integration (UDDI) registry, the ebXML Registry/Repository and the Open GIS\(^3\) Consortium (OGC) registry. The latter two of these we found to be a suitable basis for our own registry/repository, PIMRep External which is specified in this thesis.

First starting to work on this thesis we were quite ambitious as to what functionality to include in our repository. Eventually we made the discovery that the repository should actually be divided into two separate units, PIMRep External and PIMRep Internal. In the following we will present two potential use cases for PIMRep Internal.

Figure 1-4 shows the use case *View graphical rendering of model* which includes use case *Generate graphical UML model form XMI file* executed by a component *GraphicalRendererEngine*. As this is something that can be accomplished by mostly any UML modeling tool we get the feeling that this functionality should not be included in the external repository.

---

\(^3\) Geographic Information Systems
Figure 1-4 Viewing graphical rendering of model

Figure 1-5 depicts the use case *View generated code* which includes use case *Generate code* executed by component *CodeGeneratorEngine*. Use case *Generate code* has two sub-use cases which include *Generate Java from XMI* and *Generate IDL from XMI*. 
For realizing the latter two use cases one requires a repository with a unique access to the models it is hosting. Other components should be able to query the repository about every little detail in the model to be able to generate code from it or to make a graphical rendering of it. While the external repository is for searching and browsing for models, the internal repository is for accessing the elements within the models.

For the internal repository we want to use Meta Object Facility (MOF) compliant technology, because that is an integral part of MDA which is ultimately the framework for software engineering that we think is the most sensible. We have tried out two different implementations of MOF. These are dMOF and the Reference Implementation of Java Metadata Interface (JMI). They will be explained later on.

### 1.1 Definitions

In this section we shall define some terms that will be used throughout this thesis.

**Platform Independent Model (PIM).**

According to the MDA Guide [4], a PIM describes a system, but does not show details of its use of its platform.
We quote from [4]:

*A PIM might consist of enterprise, information and computational ODP viewpoint specifications. (The structure of this information model might be quite different from the structure of an information viewpoint model in a computation independent model of the same system.)*

The focus of this thesis is on a service oriented infrastructure. So, as far as we are concerned, the functionality of a system that is most important to express in a PIM, is the interface that it exposes to other systems. That includes methods with their input and output parameters.

**Service oriented infrastructure.**

This thesis is first and foremost meant to apply to a service oriented infrastructure. By service oriented infrastructure, we mean an infrastructure where systems publicly expose their interfaces and functionality, so that other systems can use them. For example, in the domain of e-commerce, making new business agreements could potentially be accomplished automatically in a service oriented infrastructure.

With the Internet and the emerging Web services technology, which makes it possible for applications to expose their methods over the HTTP protocol, the concept of a service oriented infrastructure is becoming more and more relevant.

**PIMRep.**

Platform Independent Model Repository (PIMRep) is a generic term for PIMRep External and PIMRep Internal, which are the two applications that are introduced in this thesis. By introducing, prototyping and applying these applications, we also, implicitly, define a methodology for the use of PIMRep.

**Information model.**

The term information model is used several places in this thesis. The meaning of the term information model is somewhat similar to the meaning of the term domain model. It is a model containing the most essential elements of a certain context.

For example, the information model of PIMRep External includes the classes that make up the most important concepts of the application.

**Meta data.**

Meta data is data about data. In the context of PIMRep External, meta data is the information describing a document such as a PIM. This information could include the submitting organization of the PIM, a classification of the PIM according to some classification scheme, or an auditable event related to the PIM, et cetera.

In the context of PIMRep Internal, meta data denotes the models contained in the repository. The set of concepts used to define a model is called the meta model.
1.2 Structure of this thesis

In the following is a short description of each chapter.

Introduction.
This chapter, where we present the purpose of this thesis which is to specify and validate a platform independent model repository. We also outline some initial thoughts concerning this research topic.

Problem examples.
In this chapter we present two problem examples that illustrate the need for a platform independent repository. Problem example number one is a project called Infrastructure of Norwegian e-commerce, in which I became involved in at the end of the project. I was a part of the taskforce concerned with *The use of registry and repository in an open infrastructure in electronic collaboration* [5]. This problem example is not applied to PIMRep, but serves as a background justifying the need for such a repository.

Problem example number two is the ACE-GIS project. This is a research project funded by the European Union with participants from Spain, Portugal, Belgium, Germany, Ireland and Norway. A number of students from SINTEF, including myself, realized a small scenario base upon ACE-GIS. This scenario is later used as a case study for PIMRep.

Requirements.
In this chapter we introduce the requirements for our repository. We also introduce the division of the repository in an external and an internal part. The requirements are summed up in table 3-1 Requirements for the external repository and table 3-2 Requirements for the internal repository.

Existing technologies.
In this chapter we investigate existing technologies whose purpose are somewhat similar to ours.

As candidates for external repositories we introduce UDDI, ebXML Registry/Repository and OGC Registry.

For the internal repository we have chosen to focus on MOF technology, and we introduce dMOF and the Reference Implementation of JMI.

Evaluation of existing technologies.
In this chapter we evaluate the technologies presented in the previous chapter.

UDDI, ebXML Registry/Repository and OGC Registry are evaluated according to the requirements for the external repository.

dMOF and the Reference Implementation of JMI are evaluated according to the requirements for the internal repository.
**PIMRep Architecture.**
In this chapter we present what is to be referred to as the PIMRep Architecture. We introduce a couple of significant figures and explanatory text, clearly defining the tasks of PIMRep External and PIMRep Internal.

**PIMRep External.**
This chapter sorts of outlines a specification of PIMRep External. Both the information model and the services specification are described thoroughly through UML diagrams and explanatory text.

**PIMRep External applied to Modusa gas dispersion Web service.**
In this chapter we apply PIMRep External to the Modusa gas dispersion Web service presented as a student scenario in chapter 2 Problem examples. The application of PIMRep External to the student scenario is documented through UML object diagrams and tentative screenshots.

**Evaluation of PIMRep External.**
This chapter provides an evaluation of PIMRep External according to table 3-1 Requirements for the external repository.

**PIMRep Internal.**
This chapter outlines a specification of PIMRep Internal. We describe how the core of the application can be generated by a tool.

**PIMRep Internal applied to Modusa gas dispersion Web service.**
In this chapter we apply PIMRep Internal to the Modusa gas dispersion Web service. We show how a tool using PIMRep Internal can be used to develop the platform independent model of the Modusa gas dispersion Web service.

**Evaluation of PIMRep Internal.**
This chapter provides an evaluation of PIMRep Internal according to table 3-2 Requirements for the internal repository.

**Conclusions and future work.**
This chapter provides a summary of the findings in this thesis and outlines the areas of concern that should be exposed to further work.

### 1.3 Summary
This chapter has been an introduction to this thesis. We have described the area of concern and the purpose of this thesis. We have also given an outline of the remaining chapters.
2. Problem examples

The need for a repository for platform independent models have emerged from several projects. To illustrate the need for such a repository I will in this chapter present two projects I have been involved in which in their own way both need some kind of repository.

2.1 Infrastructure of Norwegian e-commerce

This project was initiated in the spring of 1999 by Norsk EDIPRO (Norwegian Centre for EDI and Trade Procedures). Norsk EDIPRO is now known as NorStella. Other participants in the project were:

- Norsk Hydro
- IBM
- C. Tybring-Gjedde
- Telenor
- EdiSys
- Skandinavisk Transportsystem
- Antares
- SINTEF Tele og Data

The purpose of the Infrastructure project was to address the new B2B technologies such as ebXML and Web Services to see how they could best be applied in Norway.

The project resulted in a series of documents concerning different aspects of e-commerce including:

- Strategy and architecture [5].
- Requirements to the description technique to harmonize collaboration processes [6].
- Describing collaboration models in an open infrastructure [7].
- Application readable models in an open infrastructure [8].
- Requirements to establishing and maintaining registry and repository [9].
- The use of registry and repository in an open infrastructure in electronic collaboration [5].

Myself, I was part of the taskforce producing the last of these documents, *The use of registry and repository in an open infrastructure in electronic collaboration* [5]. In the following I will present the most important findings in this document.

---

*All of these documents are available for download at the NorStella web site http://norstella.no/*
2.1.1 The use of registry and repository in an open infrastructure in electronic collaboration

The project defines a few basic preconditions for an open infrastructure of e-commerce to be realized:

- The companies must describe their processes related to the exchange of data (messages) with different trade partners in an unambiguous and structured way.
- The descriptions must be publicly available for all actors in the form of collaboration models that describe functionality and flow of information related to e-commerce transactions between the actors.
- It must be described how the publicly available information can be stored and accessed by third parties in a way that makes it possible, especially for small and medium sized companies, to connect to existing and potential trade partners.

From these preconditions we see that the existence of a registry/repository is vital to an open infrastructure of e-commerce.

We will discuss two aspects of the registry/repository:

- Scenarios for using the registry/repository.
- Contents of the registry/repository.

2.1.1.1 Scenarios for using the registry/repository

The registry/repository is first of all meant for business and system developers. However, over time the companies’ internal applications are supposed to access a registry/repository and use the information they find without any human intervention whatsoever.

Figure 2-1 illustrates a typical use of a registry/repository. It is in Norwegian but all the steps will be thoroughly explained.
Figure 2-1 Typical use of an ebXML registry/repository

1. **Find relevant collaboration models.**
   Company A searches in the RR-bases to find out which harmonized collaboration models (including message structures and business processes etc.) are available and suited for its business. The collaboration models of interest are downloaded.

2. **Adjustment to own system.**
   The local systems at company A are adapted to the collaboration models.

3. **Publish own use.**
   The fact that company A now can do e-commerce according to the collaboration model of interest is published to the RR-bases. This can for example be accomplished by publishing a complete collaboration profile describing what the local systems at company A offers, and in which way this is done, or by publishing information about the actual realization one has based oneself upon.

4. **Find collaboration partners.**
   Another company, company B, accesses the RR-bases to look for potential collaboration partners. After identifying one or more interesting partners, company B downloads information about the collaboration models and realizations that can be used to do e-commerce with the partners of interest.

5. **Enter into collaboration agreement.**
   Company A and B enter into a collaboration agreement to do e-commerce according to some collaboration model/realization and adjusts/configures their systems mutually to be able to do business transactions according to this collaboration model.

6. **Exchange of business transactions.**
   Company A and B accomplish business transactions based on the agreement.
2.1.1.2 Requirements to the contents of the registry/repository

The Infrastructure project has defined a number of documents to describe electronic business. In the context of a registry/repository these are all referred to as e-commerce objects. The purpose of an RR-base is to store such e-commerce objects including:

- Collaboration model
- The activity model of a collaboration model
- The interaction model of a collaboration model
- The information model of a collaboration model
- Domain model
- Message model belonging to a collaboration model (there can be several per collaboration model)
- Realization document specifying the realization on a specific platform (Web Services, ebXML or other): XSD-schema, BPSS, CPPA, WSDL, WSFL, etc.
- Semantic mapping document (a set of pointers from realization document to model object)
- Context specific core component (Business Information Entity)

One has also defined points of access to an RR-base. The e-commerce objects are stored according to these points of access.

Figure 2-2 Infrastructure project architecture

Figure 2-2 explains the Infrastructure project architecture. We will not go in great detail about the figure, but explain the main idea. Every model in the Infrastructure project, be it a PIM or PSM, consists of three parts; activity model, interaction model and information model. The two big boxes to the upper left are the PIMs for people (UML) and for applications (XML). The bottom box symbolizes PSMs, and then there are some mapping
rules between PIMs and PSMs. The idea in the Infrastructure project is that all this information goes into registries/repositories, as depicted in the figure.

Eventually the RR-bases will contain a variety of different models and realizations. To effectively access all this information one needs to characterize the e-commerce objects. An adequate characterization will probably need a number of dimensions such as:

- Geography
- Business sector
- Business function
- Process
- Product category
- Official/legal frame conditions
- Role

We will give a brief description of these dimensions in the following.

**Geography.**

This dimension indicates the geographical area of the e-commerce object. If no value is given, the e-commerce object is valid within all geographical areas.

Geographical areas can be given as:

- **Continents**
- **Areas of economical cooperation**
- **Countries**
  Values should be taken from ISO 3166.1
- **Region within country (state, county etc)**
  Values should be taken from ISO 3166.2

**Sector.**

This dimension indicates the sector of interest for the e-commerce object. If no value is given, the e-commerce object is of interest within all sectors.

Values can for instance be taken from:

- **International Standard Industrial Classification (ISIC)**
  Published by the United Nations Statistics Division (UNSD)
- **United Nations Standard Product and Service Code (UN/SPSC)**
  where the first two digits indicates sector
  Published by the United Nations
- **UN/EDIFACT data element 7293 – Sector area identification code qualifier**
  Published by the United Nations/Economic Commission for Europe
Business function.

This dimension indicates the business function of the e-commerce object.

Values can be taken from:

- **UN/EDIFACT data element 7293 – Sector area identification code qualifier**
  Published by the United Nations/Economic Commission for Europe

Process.

This dimension indicates in what kind of process the e-commerce object is of interest.

Values can for instance be taken from:

- **UN/EDIFACT data element 7187 – Process type description code**

Product category.

This dimension indicates within which product category the e-commerce object is of interest. If no value is given, the e-commerce object is valid within all product categories.

Values can for instance be taken from:

- **United Nations Standard Product and Service Code (UN/SPSC)**
  Published by United Nations
- **Standard International Trade Classification (SITC Rev. 3)**
  Published by United Nations Statistics Division (UNSD)
- **Harmonized Commodity Description and Coding System (HS)**
  Published by United World Trade Organization (WTO)

Official/legal frame conditions.

This dimension indicates within which official/legal frame the e-commerce object is of interest. If no value is given, the e-commerce object is valid within all official/legal frames.

Official/legal frames can be given as:

- **Legal and regulatory conditions**
  For example principles for making contracts or customs regulations
- **Standards**
- **Documented best practices**
Role.

This dimension indicates the role of the e-commerce object.

Values can be taken from:

- **UN/EDIFACT data element 3035 – Party function role qualifier**
  Published by United Nations/Economic Commission for Europe

### 2.1.2 Summary

In this section we have described the project Infrastructure of Norwegian e-commerce and its need for a registry/repository. This is summed up in figure 2-2, which is explained earlier.

### 2.2 ACE-GIS

ACE-GIS (Adaptable and Composable E-commerce and Geographic Information Services) is a research project of the Information Society Technologies Programme of the European Union. The project has a web site at http://www.acegis.net/.

The partners of the project are:

- **Norwegian Mapping Authority (Norway)** – Administrative co-ordinator and responsible for the Environmental Pilot Demonstrator
- **Ionic Software (Belgium)** Exploitation Manager and responsible for GI services
- **e-blana (Ireland)** – responsible for EC services and the Emergency Pilot Demonstrator
- **University of Münster (Germany)** – responsible for Semantic Interoperability and interoperability architecture
- **University of Jaume I (Spain)** – responsible for conformance testing tools, standards integration and dissemination
- **SINTEF (Norway)** – Technical co-ordinator and responsible for open source model transformation toolkit
- **INESC-ID (Portugal)** – responsible for Open Source Composition and workflow services

The objective of the ACE-GIS project is to provide a set of tools and a service infrastructure that enable:

- Developers to efficiently build Web services, integrate existing services from multiple sources, and compose services to form new value-added compound services
- Service providers to register services, monitor service use and regulate contracts with users
- Users to discover, access, configure and use available geographic information services through adaptable applications.
With this objective in mind, students from SINTEF have realized a small scenario of use and development of Web services addressing some of the issues being raised in the ACE-GIS project. This scenario we have called the Modusa gas dispersion Web service. Modusa is an acronym developed at SINTEF which means *model driven user oriented system development*.

The Modusa gas dispersion Web service is developed in Java. For more information on Java Web services see [10], [11] and [12].

### 2.2.1 The Modusa gas dispersion Web service.

The Modusa gas dispersion Web service is a Web service that will, given the location of a chemical accident, provide a fire officer with a map of the area with a gas plume superimposed on it. The gas plume shows the area affected by the chemical release. This information will give the fire officer a quick way of determining what areas needs to be evacuated. Today there exist applications that can do this, but they often require the fire officer to provide lots of information concerning the chemical and the weather. In times of emergency the fire officer will want to provide as little information as possible and receive the information he needs as quickly as possible. To sum up, the goal of the Web service is to provide enough information for the fire officers to initiate the evacuation in the most exposed areas first, with a minimal of information gathering by the fire officer.

The way we have dealt with this problem is to develop a Web service that is composed of four other Web services which will provide weather data, chemical data, a map and a gas plume estimate. Putting the responsibility of providing information on Web services instead of the fire officer, gives not only a quicker response to the officer, but it also makes it an easier system to use.

The Web service is a composition of four Web services, a weather service, a chemical properties service, a gas dispersion service and a web map service. This is shown in figure 2-3.
The Web services involved in our example are described below.

- The weather service gives the weather information needed to make the estimate of the gas dispersion as accurate as possible. In this example the only weather information needed is wind speed and wind direction. In real life you would want to take more factors into consideration, but for this example the accuracy of the estimation is not the focus. The weather service we have used in this example is developed by Cape Science, and gives weather information for all airports in the world that have an ICAO code.

- The chemical properties service gives the chemical properties of a given chemical, which in real life also would be taken into consideration when calculating the gas dispersion estimate. This example case does not use the chemical information for calculating the estimate. The chemical service will also, given the location of an accident, provide chemical information about the chemical(s) stored in that location. This Web service is developed by us and is only a dummy service mimicking how a chemical information service could function if there existed any.

- The gas dispersion service calculates an estimate of how far the chemical might spread given the current weather conditions and the emission rate of the chemical. The service returns an image of a gas plume, which can then be superimposed on a map of the area. This service is developed by Ionic Software as part of the ACE-GIS project.

- The web map service gives a map of the given area. This service is developed by Statens Kartverk and only gives maps of Norway.
Figure 2-4 shows a sequence diagram of the order in which the Modusa Web Service invokes the other services.

The order in which the Modusa Web Service invokes the other services is as follows:

- **ModusaWebService**
- **ChemicalReport**
- **GlobalWeatherCapescience**
- **IonicPlume**
- **StatenskartverkWMS**

The important aspect for the sequence is that the Ionic gas dispersion service is invoked last, after the information needed is provided by the other Web services.

Figure 2-5 shows an example of how a client might use the Web service. This is a screenshot of a client we initially developed to familiarize ourselves with the technology, and relies on four different Web services not just one.
2.2.1.1 Limitations

As a consequence of the immaturity of the Web services technology, our gas dispersion Web service has some limitations that come from the fact that there currently are not many Web services available. Most Web services that you can find today are in the experimental phase, showing the direction the evolution might take. The limitations of our gas dispersion Web service are as follows; it gives an estimate of the gas dispersion for chemical accidents in airports in Norway.

2.2.2 Platform Independent Model (PIM) of the Modusa Web service

This section is a description of the PIM and our considerations connected to it and the modeling process. What we wanted to emphasis in the PIM was the general functionality within the application. Our focus was to model the functionality in a manner that could be represented throughout various types of environments such as J2EE, .NET, and different languages within these platforms. Thereby the PIM will be used as a context model for our application. This means that we had to disregard elements that could be seen as environment specific and focus on more general concerns instead. Examples of such concerns were the application structure and design, relationships between the different components, work flow, and other architectural issues.
Other elements that were important was how to best model and solve application interaction concerns. How was our Web service best to communicate with others? And what type of signatures and interfaces were to be used? Since the Modusa web service uses other Web services we needed to investigate such problems already at the PIM level. We solved these problems by adding data classes to the Web service connectors, by doing this we were able to add or change information about the other Web services without editing the static connectors.

Another issue was how to best model for changes in the external Web services. By changes we mean replacements by new Web services or changes in the existing Web services. This we have done by introducing implementation specific classes which implement our static interface. This might be realized by one implementation per Web service, as in our example, or by a controller class that is responsible for choosing among available Web service implementations which perform the same task. This is very useful for critical services that cannot be dependent on another Web service not breaking down.

Figure 2-6 shows a UML class diagram of the PIM of the Modusa Web Service
2.2.3 Platform Specific Model (PSM) of the Modusa Web Service

The next phase in the Modusa development project was to build a PSM on the basis of the PIM, before actually implementing and deploying the service.
Figures 2-7 and 2-8 show the PSM of the Modusa Web service. Because of the size of the model it has been split into two figures. Some of the classes have been replicated in both figures to improve readability.

The platform we have chosen is pure Java, so the PSM is not very different from the PIM, as we do not introduce any complex middleware. The main difference is that we have introduced an implementation class for every connector interface.

![Diagram of Modusa PSM]

**Figure 2-7 Modusa PSM**
Figure 2-8 Modusa PSM

2.2.4 Summary

In this chapter we have described the Modusa gas dispersion Web service, which shall be used in case studies in chapters 8 and 11.
3. Requirements

So, what is a repository anyway? Just as a digression, let us see what we get when we look up the word repository in the Merriam-Webster's Online Dictionary:

1. a place, room, or container where something is deposited or stored: depository
2. a side altar in a Roman Catholic church where the consecrated Host is reserved from Maundy Thursday until Good Friday
3. one that contains or stores something nonmaterial <considered the book a repository of knowledge>
4. a place or region richly supplied with a natural resource
5. a person to whom something is confided or entrusted

OK, so a repository is a place for storing stuff. Furthermore, one should be able to assume that what one has deposited in a repository is being kept safe. That is, we have to be able to trust the repository. A repository could potentially store both material and nonmaterial content. In our case it could be argued if the content is in fact material or nonmaterial. Meta data for sure is nonmaterial, but as soon as it takes the form of a document people could claim it is a material resource. That discussion, however, is not that important to us.

Now that we have a general understanding of the term repository, we need to refine it to fit our purpose. So, what exactly is our purpose? Is it to share all the meta data and documents one could possibly be in the possession of? Or is it to share information on a need to know basis? That is, to share the minimum required information for someone to do business with you or to take advantage of the services you might be offering in the domain of geographic information services, e-commerce or some other business domain. I guess our repository should be able to facilitate both these scenarios. However, I have a feeling that what I like to refer to as the marketplace scenario is the most plausible one. That is, businesses sharing meta data on a need to know basis to conduct e-commerce. On the other hand, sharing a little more information than what is needed for syntactically correct correspondence might prove useful to shed some light on the semantics of the meta data. That is actually what we do when put the PIM of the Modusa gas dispersion Web service in a repository. The only thing we need to share for others to take advantage of the service, is the external interface, in this example the signature of the method getGas DispersionReport(). However, people downloading the PIM also get to see which other Web services Modusa calls to complete its tasks. The idea is, that the more people know about a service, the easier it is to make efficient and correct use of it.

We have now established a basic conception of our repository. But what kind of functionality do we expect it to expose? Let us say that a user of the repository wants to upload some documents describing some piece of software. Will the repository at any point in time have any notion of the contents of these documents other than what was explicitly pointed out by the submitter. We have chosen to answer both yes and no to this question. That is, from now on, we will be talking about two kinds of repositories. There is the inter-
enterprise wide external repository, which lets you upload and download documents according to some context, with no ability to access the documents themselves. Then there is the content aware internal repository which lets one query and edit the downloaded material. The internal repository is private to one user or one enterprise.

### 3.1 External repository

So, what is it that we want from the external repository? What kind of documents do we want to share? Obviously documents describing computer software, but more specifically? And what information do we require for describing and cataloging these documents? Last but not least, which explicit functions and services will the repository be offering?

To start with the beginning, the repository needs clearly defined users. The users must be able to register themselves with personal information so that other users will know who they are and how to contact them. Personal information includes name, email address, postal address, visiting address, telephone number etc. Also, registering of organizations might be just as important as registering of people, because in the corporate world knowing which company you are doing business with is kind of vital. Consequently, users will have to state which organization they belong to.

Of course, some kind of log in session is implied for most computer systems. More importantly, the repository should implement some set of predefined associations that one user could have to a document. For example, if a user is the submitter of some document, he should also have permission to delete or deprecate the very same document, while it would be unnatural to give this right to a random user. Furthermore, the repository should offer associations or roles that could also be applied from one document to another, from user to document, or between any other objects in the repository.

Another requirement we could put on the external repository is that of event notification. Any registry object, registry object being a registered document, user, organization, link or something else, will typically have a status such as approved, deprecated, submitted or withdrawn. Any change to an object’s status will, among other reasons, cause an event. Examples of events could be creation, deletion, deprecation, updating and versioning. One could imagine that a user would like to subscribe to notifications of such events to any given object.

Yet another requirement would be to be able to categorize objects according to some given scheme or taxonomy. Such a classification scheme should be within any dimension such as geography, sector, business function, process, product category, official/legal frame conditions or role as mentioned in chapter 2.1.1.2: Requirements to the contents of the registry/repository. One should also be able to define new classification schemes at repository runtime.

So, these documents that we are going to store in our repository, what kind of documents are they? Well, since they are to describe computer software, I suggest we use UML models. And, because we want the repository to be aligned with the principles of model driven architecture, I think we want to store so called platform independent models, or PIMs. But, what exactly is a platform independent model? Well, it is a UML model that
does not include any elements specific to any programming language, middleware or operating system. However, be aware that the term platform is relative in the end.

We want to populate the repository with PIMs, because then people can download these PIMs and target them for their internal platform no matter what it is. But is a PIM a class diagram, an activity diagram, or what is the concise definition of PIM? See, that does not really matter to us repository guys at this point. When the modeling guys finally come up with their concluding definition of a PIM, we can easily change our definition in the repository. That is not a problem.

Note that for sakes of convenience for the users, we also want the repository to store platform specific models (PSMs), realizations (such as WSDLs) and even links to fully up and running Web services.

Now we have specified requirements for the content of the external repository. However, we need a set of possibilities for the user to find this content. He should be able to browse the content by classification schemes or object types, and also make free text searches. If a query finds no matches in a repository it should be propagated to other repositories. Therefore a repository must have an interface for maintaining links to other repositories.

Table 3-1 sums up the requirements for the external repository.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXR1: Content owner information</td>
<td>Being able to register information about content owners such as persons and organizations</td>
</tr>
<tr>
<td>EXR2: Access control</td>
<td>Being able to assign different access policies to one object to different users</td>
</tr>
<tr>
<td>EXR3: Event notifications</td>
<td>Allow for subscription to event notifications</td>
</tr>
<tr>
<td>EXR4: Classification schemes</td>
<td>Allow for uploading of classification schemes and for assigning objects to values within these schemes</td>
</tr>
<tr>
<td>EXR5: PIMs, PSMs, realizations and bindings</td>
<td>Allow for storing of PIMs, PSMs, realizations and bindings to services</td>
</tr>
<tr>
<td>EXR6: Searching and browsing</td>
<td>Allow for free text searches and browsing by object type and classification</td>
</tr>
<tr>
<td>EXR7: Distributed queries</td>
<td>Allow for queries to be distributed along a web of repositories</td>
</tr>
</tbody>
</table>

Table 3-1 Requirements for the external repository
3.2 Internal repository

So, what about the internal repository? This is thought to reside on one developer’s computer or on a network drive in the research and development division. We do no longer need all the information concerning the context of the repository documents. The documents (or models) have already been found and are in control of the individual or individuals that are going to use it.

At this time we might be making changes to the downloaded models or generating code from them. Therefore the repository has to offer an interface that makes it possible for users to access and edit every single element of the model. By now, a user might be or most certainly is an application.

Also, it might be nice to have some kind of check out routine if one developer is working on the model and wants to block other developers from accessing it.

Finally, there should be some routine for comparing your private copy of the model with the one residing in the internal repository to point out differences should they be out of sync.

Table 3-2 sums up the requirements for the internal repository.

| INR1: Create modeling elements | Being able to create UML modeling elements such as class, attribute, operation, association et cetera. |
| INR2: Read modeling elements   | Being able to read UML modeling elements such as class, attribute, operation, association et cetera. |
| INR3: Update modeling elements | Being able to update UML modeling elements such as class, attribute, operation, association et cetera. |
| INR4: Delete modeling elements | Being able to delete UML modeling elements such as class, attribute, operation, association et cetera. |
| INR5: Searching and browsing  | Being able to search or browse for UML modeling elements such as class, attribute, operation, association et cetera. |
| INR6: Access control          | Allow for fine grained access control to all modeling elements. |
| INR7: Event notifications     | Allow for subscription to event notifications. An event could for example be the creation of a modeling element. |
Table 3-2 Requirements for the internal repository

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INR8: Concurrent user access</td>
<td>Allow for concurrent user access. Functionality similar to source control tools like CVS and Microsoft Visual SourceSafe.</td>
</tr>
<tr>
<td>INR9: Versioning</td>
<td>Allow for versioning of modeling elements.</td>
</tr>
</tbody>
</table>

### 3.3 Summary

In this chapter we discussed the requirements for a repository solution for software development. We came up with two sets of requirements. Table 3-1 sums up the requirements for the external repository, while table 3-1 sums up the requirements for the internal repository.
4. Existing technologies

In the introduction we presented the area of concern for this thesis. Then we introduced two problem examples in chapter 2; the Infrastructure for Norwegian e-commerce and the ACE-GIS project. In the requirements chapter we refined the purpose of this thesis further.

So, by now, we more or less have a clear intention of what we are going to do. We are going to outline specifications for both an internal and an external repository.

Then, exactly how are we going to accomplish this? Would it, for example, be a good idea to lock oneself in ones office and not come out until one has made a thorough specification of the applications in question, without consulting other people? Or would it be a better idea to look around for people, organizations or groups addressing a similar problem, and then consider how contributions from other parties could take part in a final solution, before coming up with a solution all on your own? The answer should be obvious. I think most well-functioning solutions in engineering and scientific disciplines are always based on some earlier work.

So, what existing technologies should we look to for input? My supervisor had the initial thought that we should investigate UDDI and ebXML Registry/Repository. At this time we were just looking for repository technologies. We had not yet come up with the division between the external and the internal repository.

UDDI was the first technology we examined. It did not turn out to be a direct influence to PIMRep. It is, however, included in this thesis because it was our first step on the way to our own solution, and we would like to guide the reader along the same way. Also, the reader will be able to make up his own thoughts concerning the pros and cons of PIMRep versus UDDI.

The next technology we examined was ebXML Registry/Repository. This technology turned out to be the prototype for PIMRep External. I mean, it is always nice when you find something that, with some adjustments, can be used for your own purpose, instead of having to reinvent the wheel. The presentation of ebXML Registry/Repository in this chapter is rather brief, since the information model and services specification is not too different from that of PIMRep External explained in detail in chapter 7. This goes for OGC Registry as well, which is, as PIMRep External, also based on ebXML Registry/Repository.

At some point in time we decided to take a look at something called the Meta Object Facility (MOF) to see if it could be of any use to this thesis. It is after all a technology that OMG defines as one of the core technologies of MDA.

I dare say MOF is quite an abstract concept, and it took me some time to really get a grasp of it. That was about the same time as I came up with the division between an internal and
an external repository. MOF seemed to offer the functionality that ebXML Registry/Repository did not, and vice versa. While ebXML Registry/Repository primarily lets you upload and download documents, MOF exposes an interface for creating, reading, updating and deleting elements within the model.

In this chapter I give a brief description of MOF referring to appendix A and the specification [13] for more details. I mean, there is no point in me going through every little detail of the MOF specification in this thesis. MOF provides you with the terms to create a meta model, and then generates interfaces to create instances of this on any meta level you like.

I do, however, go through the use of two implementations of MOF; dMOF and the Reference Implementation of JMI. I take a rather utilitarian approach to this. MOF is so abstract that the only way to really understand it, in my opinion, is to try it out in real life. So, that is what I did, and that is how I present dMOF and the RI of JMI to the reader.

### 4.1 UDDI

When I first wrote about UDDI, I based my presentation upon the documents UDDI Version 2.03 Data Structure Reference [14] and UDDI Version 2.04 API Specification [15]. This time around I will also try to incorporate the new concepts introduced in the document UDDI Version 3.0.1 [16] which are also pointed out in the white paper The Evolution of UDDI [17].

Universal Description, Discovery and Integration (UDDI) is an industry wide effort driven by the Organization for the Advancement of Structured Information Standards (OASIS) for standardizing a unified way of publishing and finding Web Services. Major actors in the specification of UDDI include:

- Accenture
- Ariba
- Commerce One
- Fujitsu Limited
- Hewlett-Packard Company
- I2 Technologies
- Intel Corporation
- International Business Machines Corporation
- Microsoft Corporation
- Oracle Corporation
- SAP AG
- Sun Microsystems
- VeriSign
4.1.1 UDDI Data Structure

In this section I will present the UDDI Data Structure or the information model of UDDI as described in [14] and [16]. Figure 4-1 below is taken from [14] and shows the five most essential data structures of UDDI. UDDI uses the XML Schema Language to formally describe its data structures, so the data structures will have both XML attributes and XML elements.

![Diagram of UDDI Data Structures]

**Figure 4-1 The five data types that make up a UDDI registration**

The *businessEntity* structure holds descriptive information about an entity or business. It is uniquely identified by its *businessKey* attribute. From an XML standpoint, the *businessEntity* is the top level data structure that holds all data about the business and the services it offers. XML elements of *businessEntity* include *discoveryURLs, name, description, contacts, businessServices, identifierBag* and *categoryBag*. *discoveryURLs* is simply a list of Uniform Resource Locators (URLs) that point to alternate service discovery mechanisms. *name* and *description* are simple textual descriptions of the *businessEntity*. There can be several instances of these elements, for example in different languages. The *contacts* structure is more compound, consisting of *description, person name, phone number, email address and mailing address*. *identifierBag* contains a list of unique identifiers, each valid within its own identifier system. Such identifiers could be tax identifiers or D-U-N-S numbers. For each identifier the *identifierBag* will have a reference, *tModelKey*, to a *tModel* that specifies the identifying scheme in question.
The following code snippet taken from [16] identifies SAP AG by its Dun & Bradstreet D-U-N-S number referencing a tModel:

```xml
<identifierBag>
  <keyedReference
    tModelKey="uddi:uddi.org:ubr:identifier:dnb.cm:d-u-n-s"
    keyName="SAP AG"
    keyValue="31-626-8655" />
</identifierBag>
```

The categoryBag allows for categorizing the business within categories describing different aspects of the business such as industry, product category or geographic region. This categorization is non-unique. That is, there can be several businesses within the same industry within the same region.

The businessService structure represents a logical service classification. Each businessService structure is the logical child of a single businessEntity structure. XML attributes of businessService include businessKey and serviceKey, which uniquely identifies the business offering the service and the service being offered. XML elements of businessService include name, description, bindingTemplates and categoryBag. Simple textual information about the businessService is given by its name and its description. The categoryBag categorizes the service within dimensions such as geographic region or industry.

The bindingTemplate structure provides information for determining a technical entry point and a lightweight description of technical characteristics of a given implementation. XML attributes include bindingKey and serviceKey, which uniquely identifies the bindingTemplate and the businessService containing the bindingTemplate. XML elements include description, accessPoint, tModelInstanceDetails and categoryBag. Textual information about the bindingTemplate is given by its description. The accessPoint defines a URL representing the network address of the Web service being described. The tModelInstanceDetails structure references a set of tModels representing specifications with which the Web service represented by the containing bindingTemplate complies. The categoryBag contains a list of categorizations that can, for example, indicate whether the bindingTemplate has status “test” or “production”.

A tModel entity is uniquely identified by its tModelKey, which is an attribute. XML elements include name, description, overviewDoc, identifierBag and categoryBag. The name and description elements give simple textual information about the tModel. The overviewDoc references a remote document describing the tModel. In case the tModel is describing a Web service the document will typically be WSDL. The Web Services Description Language (WSDL) has become an industry standard for describing the behavior of a Web service as the result of a massive effort having been done to create a unique language to accomplish this.

Many big corporations and enterprises might feel the need to register multiple businessEntities because they are so diverse. However, one would still like to maintain some kind of relationship between the different businessEntities. Also, one could imagine relationships between businessEntities belonging to different
corporations. The answer to these issues is the publisherAssertion structure. XML elements of the publisherAssertion structure include fromKey, toKey and keyedReference. The fromKey and toKey elements identify the two businessEntity instances between which an assertion has been made. The keyedReference describes the nature of this relation.

### 4.1.2 Functionality and services of UDDI

In this section I will present the UDDI programming API as described in [15] and [16].

**4.1.2.1 Inquiry API functions**

The queries presented in table 4-1 are supposed to be made accessible to anyone wanting to contact a UDDI Operator Site at any time using HTTP-POST only.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>find_binding</td>
<td>This function will return a list of bindingTemplate structures matching the criteria specified in the argument list. A serviceKey is used to specify a particular instance of a businessService element in the registered data, while a set of tModelKeys represents technical fingerprints of bindingTemplate structures contained within the businessService specified by the serviceKey value.</td>
</tr>
<tr>
<td>find_business</td>
<td>This function returns a businessList. A businessList structure contains information about matching businesses and the businessServices that they expose. If a tModelBag, a set of tModelKeys, was used, the resulting structure will only contain data about the businessServices that actually had a matching bindingTemplate.</td>
</tr>
<tr>
<td>find_relatedBusinesses</td>
<td>This API call returns a relatedBusinessesList containing information about businessEntity registrations that are related to a specific business entity whose key is passed in the inquiry.</td>
</tr>
<tr>
<td>find_service</td>
<td>This function is used to locate services matching the conditions specified in the arguments. The businessKey argument specifies whether to search within a particular businessEntity or to search among all businessEntities. The optional name argument specifies the name of a service. The categoryBag argument specifies a category or multiple categories within one or more taxonomies, while the tModelBag argument specifies multiple technical fingerprints.</td>
</tr>
<tr>
<td>find_tModel</td>
<td>This function is used to locate a list of tModel entries that match a set of specific criteria. The criteria one can specify are name, identifier and/or category.</td>
</tr>
<tr>
<td>get_bindingDetail</td>
<td>The get_bindingDetail API call returns a bindingDetail message containing bindingTemplate structures matching the criteria specified in the argument list.</td>
</tr>
</tbody>
</table>
information for a specific registered business API specified in the bindingKey argument.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_businessDetail</td>
<td>This API call returns businessEntity information for the businessEntity registrations matching the businessKey values specified.</td>
</tr>
<tr>
<td>get_businessDetailExt</td>
<td>This API call works in the exact same way as the get_businessDetail API call, but it may return additional attributes if the source is an external registry.</td>
</tr>
<tr>
<td>get_serviceDetail</td>
<td>This API call returns information about a businessService structure specified in the serviceKey argument.</td>
</tr>
<tr>
<td>get_tModelDetail</td>
<td>This API call returns information about a tModel structure specified in the tModelKey argument.</td>
</tr>
</tbody>
</table>

Table 4-1 Inquiry API functions

### 4.1.2.2 Publishing API functions

Common to the messages described in this section is that they require authenticated access to a UDDI Operator Site. They are used to publish and update information, they all behave synchronously and are callable via HTTP-POST only. HTTPS is used exclusively.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>save_xx</td>
<td>save_binding, save_business, save_service and save_tModel are used to register or update information about a bindingTemplate, businessEntity, businessService or tModel respectively.</td>
</tr>
<tr>
<td>delete_xx</td>
<td>delete_binding, delete_business, delete_service and delete_tModel are used to remove information about a bindingTemplate, businessEntity, businessService or tModel respectively.</td>
</tr>
<tr>
<td>add_publisherAssertions</td>
<td>This API call causes one or more publisherAssertions supplied as arguments to be added to an individual publisher’s collection.</td>
</tr>
<tr>
<td>delete_publisherAssertions</td>
<td>This API call takes one or more publisherAssertion structures as arguments. Any exactly matching existing structures will be deleted.</td>
</tr>
<tr>
<td>get_authToken</td>
<td>This API call is used to get an authentication token. Arguments are userID and cred, which is the password.</td>
</tr>
<tr>
<td>discard_authToken</td>
<td>This API call takes an authentication token as argument. The supplied token will be discarded. Subsequent calls using the same token will be rejected.</td>
</tr>
<tr>
<td>get_assertionStatusReport</td>
<td>This API call returns information about the status of assertions that the business have made, as well as assertions that others have made which involve the business.</td>
</tr>
<tr>
<td>get_publisherAssertions</td>
<td>This API call simply returns the set of publisher assertions associated with an individual publisher account.</td>
</tr>
</tbody>
</table>
get_registeredInfo

This API call provides a list of all businessEntity and tModel structures associated with the business making the call.

set_publisherAssertions

This API call sets the set of publisherAssertions that are to be associated with the business making the call.

| Table 4-2 Publishing API functions |

4.1.2.3 Subscription API functions

The subscription API set allows for monitoring of activity in a registry by registering to track new, changed and deleted entries for these entities:

- businessEntity
- businessService
- bindingTemplate
- tModel
- related businessEntity
- publisherAssertion (as long as the subscriber owns at least one of the businesses referenced)

Table 4-3 describes the subscription API functions.

<table>
<thead>
<tr>
<th>delete_subscription</th>
<th>This function cancels one or more specified subscriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_subscriptionResults</td>
<td>This function returns registry data pertaining to a particular subscription within a specified period of time</td>
</tr>
<tr>
<td>get_subscriptions</td>
<td>This function returns a list of existing subscriptions previously saved by the subscriber</td>
</tr>
<tr>
<td>save_subscription</td>
<td>This function establishes a new subscription or changes an existing one</td>
</tr>
</tbody>
</table>

Table 4-3 Subscription API functions

4.1.2.4 Registry interaction

One of the main differences between version 2 and version 3 of UDDI is the concept of registry interaction which is introduced in version 3. This is also pointed out in [17]. I think the best gain with inter operating registries is the ability for queries to propagate to other registries. We shall, however, not go in to details on how registry interaction is accomplished in UDDI at this time.

4.1.3 Summary

In this section we have given an objective presentation of UDDI. Subsection 4.1.1 discussed the UDDI data structure, while subsection 4.1.2 discussed the functionality and services of UDDI. The UDDI track will be continued in chapter 5.1, where we shall...
evaluate UDDI according to the requirements we defined for an external repository in table 3-1 in chapter 3-1.

4.2 ebXML Registry/Repository

ebXML (Electronic Business using eXtensible Markup Language), sponsored by UN/CEFACT and OASIS, is a set of specifications that are supposed to enable enterprises of any size to conduct business over the Internet. UN/CEFACT is a body of the United Nations whose mandate is to support the worldwide development in the area of trade facilitation and electronic business. OASIS is an international nonprofit consortium that promotes open collaborative development of interoperability specifications. For more background on ebXML see [18], [19] and [20].

The ebXML registry/repository is supposed to be able to hold all the information a business would need to use the ebXML framework to conduct e-commerce.

The specification of the registry/repository consists of two documents; OASIS/ebXML Registry Information Model [21] and OASIS/ebXML Registry Services Specification [22].

4.2.1 Registry Information Model

Figure 4-2 is taken from [21] and shows some of the objects in the Registry and their relationships as a UML class diagram. Figure 4-3 shows an inheritance view of the classes inheriting from the RegistryObject class. The RegistryObject class is really the key to the ebXML Registry information model as almost all the other classes inherit from this class. Every RegistryObject is identified by a unique universal identifier. We shall, however, not go into detail on the specific classes as most of these will later be introduced in what we shall call PIMRep External.
Figure 4-2 Overview of the ebXML Registry information model

Figure 4-2 gives an overview of the ebXML Registry information model. We see from the figure that the focal point of the information is the class RegistryObject. We see for example that a RegistryObject can have a set of Classifications according to a set of ClassificationSchemes. The class Classification actually represents the association between a ClassificationNode and a RegistryObject. A ClassificationNode represents one particular value within a given ClassificationScheme.

A RegistryObject can also be classified by a classification scheme that is not uploaded to the registry in its entirety. This is accomplished through the class ExternalIdentifier. The class ExternalLink represents some external source describing the RegistryObject, for example a web page. Note the Association class between ExternalLink and RegistryObject. This is a generic class used throughout the information model.

A RegistryObject can also be a member of a RegistryPackage, which simply is a collection of RegistryObjects.

The class Slot is the extension mechanism of the information model. It can be used to represent arbitrary pairs of properties and values describing a RegistryObject.

Every RegistryObject must belong to an Organization, which has a set of Users affiliated with it. Both Users and Organizations have PostalAddresses, but Users also have TelephoneNumbers and EmailAddresses. The class AuditableEvent keeps track of events involving a given RegistryObject.

If the RegistryObject represents some kind of ebXML defined document, the class specificationLink will link you to this document. If it is deployed as a service one or more
places, ServiceBindings will provide you with URIs to where it is deployed. The class Service keeps track of a set of ServiceBindings deploying the same specification.

Figure 4-3 Inheritance view of the classes inheriting from RegistryObject

Figure 4-3 shows the inheritance view of class RegistryObject. We see that most classes inherit from RegistryObject. Properties of RegistryObject typically include name and unique identifier. Some of the classes also inherit from a class called RegistryEntry. The class RegistryEntry has properties like versioning, for example.

4.2.2 Registry Services Specification

This section describes the services offered by an ebXML compliant registry according to [22]. The services include:

- Lifecycle Management Service
- Query Management Service
- Content Management Service
- Event Notification Service

The purpose of the LifeCycleManagement service is to manage the lifecycle of RegistryObjects. Figure 4-4 taken from [22] depicts the typical lifecycle of a RegistryObject. We see that a RegistryObject can be in four different states: Submitted, Approved, Deprecated and Removed. We also see which functions that can lead to the different states.

The LifeCycleManager implements protocols for submitting objects, updating objects, adding slots, removing slots, approving objects, deprecating objects, undeprecating objects and removing objects. As an example, figure 4-5 taken from [22] shows a UML sequence diagram of the Submit Objects Protocol.
Figure 4-4 The typical lifecycle of a RegistryObject

Figure 4-4 shows a typical lifecycle of a RegistryObject. First it is submitted. Then it is approved by other parties. Eventually it might get deprecated or removed.

Figure 4-5 The Submit Objects Protocol

The purpose of the Query Management Service is to allow clients to search for or query different kinds of registry objects in the registry. The registry supports two kinds of query capabilities; Filter Query and SQL Query.

FilterQuery is an XML syntax that provides a set of queries for the registry. Each query is directed against a class in the ebXML Registry Information Model, for example RegistryObject or ExtrinsicObject. A FilterQuery is submitted as part of an AdhocQueryRequest. Figure 4-6 taken from [22] shows a sequence diagram of a client submitting an ad hoc query:
Figure 4-6 A client submitting an ad hoc query

SQL Query is an optional feature of the ebXML Registry. It allows advanced users to define SQL statements themselves. The ebXML Registry Services Specification provides a description of which subset of SQL that can be used plus a mapping from the ebRIM to a relational schema. However, this does not mean that one is required to use relational databases in a Registry implementation.

In the following we will present some examples of Content Management Services. Content Management Services are meant to improve the quality of contents submitted to the registry.

Content Validation provides the ability to enforce validation rules upon submitted content and reject it if it contains invalid data as depicted in figure 4-7 taken from [22].

Figure 4-7 Content validation

Content Cataloging provides the ability to convert submitted RegistryObject and repository items into metadata defined by ebRIM in a content specific manner as depicted in figure 4-8 taken from [22].
Event notification is an optional but normative feature of the registry. It allows users to be updated on events of interests without having to periodically poll the registry.

A user might want to use an Event Notification Service to know when a new service of interest is offered, to know whenever his submitted documents are downloaded, or to monitor the price of a specific product.

Notice that we have not really gone into detail about the specifics of the functions comprising the ebXML Registry service interfaces. This is because many of the same functions will be incorporated into PIMRep External and therefore described in another chapter.

### 4.2.3 Summary

In this section we have given an objective presentation of the ebXML Registry/Repository. Subsection 4.2.1 discussed the registry information model, while subsection 4.2.2 discussed the registry services specification. The ebXML Registry/Repository track will be continued in chapter 5.2, where we shall evaluate ebXML Registry/Repository according to the requirements we defined for an external repository in table 3-1 in chapter 3-1.

### 4.3 OGC Registry

The Open GIS Consortium (OGC) is a standardization body within the Geographic Information Systems (GIS) society. The OGC has discussed technological issues for a registry based on the ebXML Registry specification but with a specific focus on Geographical Information Systems. Figure 4-9 shows a UML component diagram for a fully capable registry service according to the OGC and is taken from [23]. We shall not be very concerned with the information model for the OGC Registry at this point as it is based on the ebXML Registry information model and is the basis for what we shall later present as the PIMRep External information model.
4.3.1 OGC Registry interfaces

The service interfaces of the OGC Registry are pretty much the same as those for ebXML Registry and shall be described later. There are, however, a couple of interfaces that are specific to the GIS society. They will be described in the following.

4.3.1.1 WRSQuery interface

The WRSQuery interface includes query operations that are based on similar Web Feature Server (WFS) operations. The operations are illustrated in figure 4-10 and explained in table 4-4.

Figure 4-10 WRSQuery

*getCapabilities(body, GetCapabilitiesRequest) : GetCapabilitiesResponse*
*getRecords(body, GetRecordsRequest) : GetRecordsResponse*
*getRecordById(id, anyURI) : GetRecordResponse*
getCapabilities() operation
This operation is used to get the capabilities of the registry service.

get Record() operation
This operation allows you to specify a number of parameters to search for a RegistryObject.

getRecordByID() operation
This operation is used to get a specific RegistryObject by specifying a unique id.

describeType() operation
This operation is used to retrieve XML schemas for common types.

Table 4-4 WRSQuery

4.3.1.2 WRSTransaction interface

This interface is based on the OGC WFS transaction interface. The operations are illustrated in figure 4-11 and explained in table 4-5

![WRSTransaction Diagram]

transaction() operation
This operation is used to insert, update and delete registry objects.

registerResource() operation
This operation is used when one wants to update an expired resource.

lockRecord() operation
This operation is used to prohibit access to a registry object such as to avoid concurrency.
4.3.2 Summary

In this section we have given an objective presentation of the OGC Registry. Subsection 4.3.1 discussed the registry OGC Registry interfaces, which are an extension of standard ebXML Registry/Repository. The OGC Registry track will be continued in chapter 5.3, where we shall evaluate OGC Registry according to the requirements we defined for an external repository in table 3-1 in chapter 3-1.

4.4 Model Driven Architecture and the MOF

Model Driven Architecture (MDA) is the new focus of the Object Management Group (OMG). Instead of concentrating on only one middleware platform (CORBA), OMG decided to build a framework for developing platform independent models, which is MDA. One of the key technologies in MDA is MOF.

Figure 4-12 The MDA logo
4.4.1 MOF

The Meta Object Facility (MOF) is an abstract modeling language specified by the OMG. The MOF is defined in prose and thus does not have any formal notation. It can be used to define a number of other languages, for example UML, CWM, Java and CORBA IDL. The MOF does also define itself, i.e. the concepts constituting the MOF are used to describe the same concepts. Figure 4-13 shows an overview of the MOF model and is taken from [24]. The classes in figure 4-13 constitute the basis for the definition of any modeling language. For a more elaborate description of MOF please see appendix A or the MOF specification [13].

The next two existing technologies to be presented, dMOF and the Reference Implementation of JMI are both MOF-based.

Figure 4-13 An overview of the MOF model

The elements in figure 4-13 are used to define meta models. Again, the meta models are used to define models or meta data, which is the same in this context.
4.5 dMOF

The Distributed Systems Technology Centre (DSTC) is an Australian IT research and development centre focusing on the needs of the government, defence, health, telecommunications, finance and education sectors.

DSTC’s dMOF product is a set of command line tools that implement the OMG’s MOF specification [13]. The key features of the dMOF product line are summarized as follows in the dMOF user guide [25]:

- The dMOF meta-model repository. This is a persistent implementation of the OMG’s MOF meta-model repository specification.
- The MODL compiler. This loads a MOF meta-model (specified in the MODL language) into the meta-model repository.
- The MOF meta-model interchange tools. These allow you to interchange MOF meta-models encoded in XMI.
- The MOF IDL generator. This produces CORBA IDL for meta-data repositories defined by a user meta-model.
- The Moflet generator. This produces moflets; i.e. Java code implementing the repository meta-objects that represent a user’s metadata.
- The XMI generators. These produce a DTD and encode/decode software, allowing the interchange of the user’s metadata via XMI.

The objective of dMOF is to generate so called meta-model repositories implemented in CORBA. These repositories are in the user guide [25] referred to as Moflets.

The first step in deploying a Moflet is to define a meta-model. Figure 4-14 shows an example meta-model called SimpleTrader expressed in UML. It defines a simple type system for an object service trader.

![SimpleTrader meta-model](image)

Figure 4-14 SimpleTrader meta-model
Given a meta-model expressed in UML, we must now create a formal meta-model definition in the MODL language. MODL is a text-based language for specifying MOF meta-models. It is defined by DSTC and it is loosely based on OMG IDL. The SimpleTrader meta-model can be expressed in MODL as follows:

```modl
package SimpleTrader {
    enum PrimitiveType {pt_bool, pt_int, pt_float};
    abstract class BaseType {
        attribute string name;
    };
    class PropertyType : BaseType {
        attribute PrimitiveType value_type;
    };
    class ServiceType : BaseType {
        attribute string interface_id;
        reference supers to supertype of Inherits;
        reference props to property of Has;
    };
    association Has {
        end single ServiceType service;
        composite end set [0..*] of PropertyType property;
    };
    association Inherits {
        end set [0..*] of ServiceType supertype;
        end set [0..*] of ServiceType subtype;
    };
}
```

### 4.5.1 Generating a Metadata Repository

When the meta-model has been specified in MODL, the next step is to generate a metadata repository. The process of generating the metadata repository is depicted in figure 4-15 taken from the dMOF user guide [25].
Metadata repository generation is explained step by step in the user guide [25]:

1. Use the MODL compiler (modl2mof) to compile the meta-model(s) and load them into the developer’s MOF meta-model repository. The MODL compiler does a degree of semantic checking on the input MODL file to eliminate problems down the track.

2. Use the IDL generator (mof2idl) to produce the CORBA IDL for a metadata repository. The IDL generated by this step allows a CORBA based client program to create, update and access metadata in a repository. This generated IDL conforms to the standard MOF IDL mapping.

3. Use the Moflet generator (mof2moflet) to produce Java code that implements the MOF metadata repository functionality. This moflet code implements the defined semantics of the standard MOF to IDL mapping. The code is internally transactional and supports metadata persistence using a variety of database back-ends.

4. Implement special semantic hooks for the generated moflet. This is only necessary if the meta-models use advanced features such as Operations, derived Attributes or derived Associations.

5. Implement a metadata server main program, or the equivalent. There are a variety of ways to package moflet code to provide metadata service.

### 4.5.2 Building Metadata Applications

Having implemented the metadata repository, the developer will now have to implement metadata tools needed by the system. These metadata tools may include:

- Input/output tools such as compilers, pretty printers, diagram editors and browsers.
- Interchange tools such as XMI DTDs and upload/download tools.
- Versioning, comparison and archiving tools.
4.5.3 Summary

This chapter has provided the reader with a general understanding of what dMOF is and how to use dMOF. The dMOF track will be continued in chapter 5-5, where dMOF is being evaluated according to the requirements for an internal repository defined in table 3-2 in chapter 3-2.

4.6 Reference implementation of JMI

The Reference Implementation of the Java Metadata Interface Specification is copyright of Unisys Corporation and Adaptive Ltd. The implementation is presented as a Java desktop application, and it is in the user guide [26] referred to as CIM.

In addition to the JMI and MOF specifications, CIM defines three new concepts – facility, repository and metadata server:

Facility:
*A CIM facility is a persistence store for metadata. A facility is the 'root' container of the CIM service. CIM standard supports XMI persistence for a facility.* [26]

Repository:
*In MOF, the highest level at which interoperability is guaranteed is the outermost package. As MOF does not prescribe how outermost packages are created, CIM introduces the concept of a repository. The repository is nothing more than the CIM container for the MOF outermost package. Each repository contains a single outermost package.* [26]

Metadata Server:
*The metadata server is the CIM implementation of the Java APIs for a metamodel. As mentioned before, JMI specifies the APIs that are to be generated for manipulating instance data of a metamodel. A CIM implementation of these APIs is referred to as a metadata server. CIM provides the capability to generate metadata servers for any MOFcompliant M2 metamodel.* [26]
Figure 4-16 Workbench GUI

Figure 4-16 depicts the CIM user interface and names the different elements in the GUI.

4.6.1 Example Usage of CIM

First we create a new facility called SimpleRdbFac. The new facility contains a metadata server called com.Unisys.jmi.mof.Model Server which implements the JMI APIs for the for the MOF 1.4 Model. A repository named MofMetaMetamodel containing the MOF 1.4 Model is pre-loaded into the pre-installed metadata server.

Now we need to define our own metamodel and load it into the metadata server. Figure 4-17 describes a partial relational database model.

Figure 4-17 SimpleRDB model
Next, we define a new M2 repository in com.Unisys.jmi.mof.Model Server called RDB Model. Loading data into the repository can be done in two days. Either programatically using the metadata server APIs or by using the reflective XMI import/export capabilities. In order to import it as an XMI file we need to generate an XMI file from our favourite UML modeling tool. It will look something like the following:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<XMI xmlns:Mof="http://org.omg.mof/Model">
  <XMI.header>
    <XMI.metamodel xmi.name="Model" xmi.version="1.2" />
  </XMI.header>
  <XMI.content>
    <Model:Tag xmi.id='a3CFE889E03D2.packagePrefix' name='javax.jmi.packagePrefix=examples' annotation='' tagId='javax.jmi.packagePrefix' elements='a3CFE889E03D2' />
    <Model:Package xmi.id = 'a3CFE889E03D2' name = 'SimpleRDB' annotation = '' visibility = 'public_vis' >
      <Model:Namespace.contents>
        <Model:Class xmi.id = 'a3CFE88D401BE' name = 'Table' annotation = '' visibility = 'public_vis' isSingleton = 'false' >
          <Model:Namespace.contents>
            <Model:Reference xmi.id = 'ra3CFE891B0122' name = 'column' annotation = '' scope = 'instance_level' visibility = 'public_vis' isChangeable = 'true' type = 'a3CFE88EA033D' referencedEnd = 'a3CFE891B0122' />
            <Model:Attribute xmi.id = 'a3CFE88DC01CA' name = 'name' annotation = '' scope = 'instance_level' visibility = 'public_vis' isChangeable = 'true' isDerived = 'false' type = 'G.4' />
        </Model:Reference>
        <Model:Reference xmi.id = 'ra3CFE891B0122' name = 'column' annotation = '' scope = 'instance_level' visibility = 'public_vis' isChangeable = 'true' type = 'a3CFE88EA033D' referencedEnd = 'a3CFE891B0122' />
        <Model:Attribute xmi.id = 'a3CFE88DC01CA' name = 'name' annotation = '' scope = 'instance_level' visibility = 'public_vis' isChangeable = 'true' isDerived = 'false' type = 'G.4' />
      </Model:Namespace.contents>
    </Model:Class>
  </XMI.content>
</XMI>
```
Now that the Simple RDB metamodel is loaded into the repository, we need to generate a metadata server for it. The metadata server generation window provides us with a set of options:

- **Generate APIs only.** Check this if you do not want the server to generate a full implementation of the generated APIs.
- **Debug.** Check this if you want to include debug information with your server.
- **Documentation.** Allows you to enter a block of text that will appear at the top of every Java Class that is generated as part of the metadata server.

![Figure 4-18 The SimpleRDB Metadata Server](image)

Figure 4-18 depicts the CIM Workbench after being exposed to our example scenario.

### 4.6.2 Summary

This chapter has provided the reader with a general understanding on how to use the Reference Implementation of JMI. The RI of JMI track will be continued in chapter 5-5, where the RI of JMI is being evaluated according to the requirements for an internal repository defined in table 3-2 in chapter 3-2.

### 4.7 Summary

In this chapter we have investigated existing technologies whose purpose are somewhat similar to ours.

As candidates for external repositories we have introduced UDDI, ebXML Registry/Repository and OGC Registry.
For the internal repository we have chosen to focus on MOF technology, and we have introduced dMOF and the Reference Implementation of JMI.
In the previous chapter we gave objective presentations of some existing technologies. In this chapter we will evaluate the very same technologies according the requirements formulated in chapter 3.

UDDI, ebXML Registry/Repository and OGC Registry will be evaluated according to the requirements for an external repository, while dMOF and the Reference Implementation of JMI will be evaluated according to the requirements for an internal repository.

Also, for UDDI, ebXML RR and OGC Registry we will present some screenshots of client applications actually using the technologies. I think, being exposed to graphical user interfaces gives a more real understanding of the technologies. Trying out stuff in real life has helped me a lot when writing this thesis, and I would like to pass that experience on to the reader.

The MOF technologies dMOF and the RI of JMI are really in an infant stage, and at this time no client applications illustrating the real potential of these technologies, exist.

5.1 UDDI

In this section we are going to resume the thread from chapter 4.1, where we described the UDDI technology, and evaluate it according to the requirements in table 3-1 in chapter 3-1.

So, what about UDDI? Does it work? Figure 5-1 shows a screenshot of a web browser based GUI of a Microsoft implementation of UDDI hosted by e-blana which is one of our partners in the research project ACE-GIS. As ACE-GIS among other things is a project for finding ways to discover and access Web services, e-blana has done a good job in registering their services in UDDI. In the screenshot there are tabs that hold different information. You can for example click on the Services tab and see which Web services are offered by e-blana, which by the way also are displayed in the leftmost pane. Furthermore, there are tabs for contacts, categories, relationships et cetera. You cannot see this in the figure, but I assure you they are all conscientiously filled out. You might not think this sounds like a great deal, but what I have discovered after looking through quite a few UDDIs including the so called Universal Business Registries (UBR) hosted, among others, by IBM and Microsoft, is that the greatest problem with UDDI is that people does not use it as intended. Instead of providing information on how to access their Web services, they might just fill in an e-mail address or a phone number instead. This should indicate that UDDI might seem a little blurred to the general public. It seems to me that some of the concepts of UDDI should be clarified a little for UDDI to reach a broader audience. For example, the concept of a tModel seems to me at one time to be referring to
one kind of information, while at another time be referring to something completely different.

As UDDI is to be considered an external repository, table 5-1 evaluates UDDI according to the requirements matrix for external repositories as given in table 3-1 in chapter 3.1.

<table>
<thead>
<tr>
<th>EXR1: Content owner information</th>
<th>You are able to register contact information such as phone number, visiting address and email address.</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXR2: Access control</td>
<td>Not very sophisticated access control. You can either be in the role of a publisher or a searcher. You can only edit information published by yourself.</td>
</tr>
<tr>
<td>EXR3: Event notifications</td>
<td>The UDDI servers that are up and running at this time are of version 2 and do not allow for subscription to event notifications. However, version 3 allows for subscription to some events.</td>
</tr>
<tr>
<td>EXR4: Classification schemes</td>
<td>Classification schemes can be uploaded as tModels.</td>
</tr>
<tr>
<td>EXR5: PIMs, PSMs, realizations and bindings</td>
<td>UDDI does not allow for storing any kind of documents (PIMs or PSMs). It only provides</td>
</tr>
</tbody>
</table>
binding information for Web services. While it is possible to have a link to a PIM or a PSM, that is not the intention of UDDI. If, however, linking to a PIM or a PSM, the PIM or PSM would be a tModel as far as UDDI would be concerned.

| EXR6: Searching and browsing | UDDI allows for free text searches and browsing by category. |
| EXR7: Distributed queries | One instance of UDDI does not propagate its queries to other instances of UDDI. |

Table 5-1 Evaluation of UDDI

In this section we have evaluated UDDI according to the requirements matrix for external repositories. While meeting some of the requirements, UDDI fails to fulfill others. Our conclusion is that we shall not base PIMRep External on UDDI. The reason for this is that, in our opinion, the UDDI data structure is too ambiguous and incomplete to work with.

5.2 ebXML Registry/Repository

In this section we are going to resume the thread from chapter 4.2, where we described the ebXML Registry/Repository, and evaluate it according the requirements in table 3-1 in chapter 3-1.

ebXML is being developed rapidly around the world, and the scene includes a diversity of actors. In this jungle of ebXML development it is quite a challenge knowing where to begin to get some hands on experience. I chose to join the user community of something called the Reference Implementation of ebXML Registry/Repository. It is an open source project hosted by Sourceforge at http://ebxmlrr.sourceforge.net. At the time there did not exist any decent binary distribution of ebxmlrr so I had to download the source and compile and configure it by myself. As we all know, these things never quite work at once, it takes some effort, so I guess you could say that I have gained quite some insight into this project. The project is being led by Sun Microsystems.

Figures 5-2, 5-3 and 5-4 shows authentic screenshots of a client that came with the ebxmlrr project. The client communicates with the ebXML Registry/Repository through the protocol Java API for XML Registries (JAXR).

In figure 5-2 I am running the client against a trial version of ebxmlrr which is hosted at http://registry.csis.hku.hk:8201/ebxmlrr/registry/soap which actually happens to be at the Department of Computer Science and Information Systems at the University of Hong Kong! It was the first version 2 compliant registry to go online.

In the figure I am searching for an ExtrinsicObject with name CompanyA. I find one match of object type CPP which is some XML format specific to ebXML that describes ones company ability to do e-business. CPP is by the way short for Collaboration Protocol Profile. Then I mark the search result and right-click it with the mouse and a menu pops up...
as shown in figure 5-3. I then choose the Retrieve Item option and the Save As.. dialogue box as depicted in figure 5-4 shows up. I can now save the CompanyA file to my local computer. The file is an XML file and the CPP for company A.

The ebXML Registry/Repository can also be used to register regular Web services. What distinguishes ebXML R/R the most from UDDI, in my opinion, is that the ebXML R/R Reference Implementation is open source. That opens for a more dedicated user community. For ebXML R/R, you have a common code base which can be applied to different domains with only slight modifications. Whereas for UDDI, making your own implementation means so much overhead that you will probably just go with the Microsoft UDDI Server. Another thing is that ebXML R/R has a much more explicit information model.

Table 5-2 evaluates ebXML Registry/Repository according to the requirements matrix for external repositories as given in table 3-1 in chapter 3.1.

![Figure 5-2 Searching for CompanyA](image)

**Figure 5-2 Searching for CompanyA**

Figure 5-2 shows an ebXML Registry/Repository Java Swing client. We have specified the search string companyA and object type ExtrinsicObject. We see that it is also possible to specify classifications, external identifiers, specifications and external links. Also, by clicking the tab Submission, we get an interface for submitting new objects to the repository.
When pressing the button with the key on the tools menu, we get to identify ourselves for operations that require authentication.

Figure 5-3 Retrieving item

In figure 5-3 we get a pop up menu with different choices for what to do with the object CompanyA. We can edit, remove, save, browse, show audit trail or retrieve item.
Figure 5-4 Downloading file

Figure 5-4 allows us to specify where to download the file to our local file system.

| EXR1: Content owner information | In ebXML Registry/Repository you can register contact information such as telephone number, email address and visiting address for both users and organizations. |
| EXR2: Access control | ebXML R/R has quite a sophisticated access control as multiple users can have edit access to one object. |
| EXR3: Event notifications | ebXML R/R allows for subscription to event notifications. |
| EXR4: Classification schemes | In ebXML R/R you can upload new classification schemes dynamically, and you can also give objects new characteristics arbitrarily by using slots. |
| EXR5: PIMs, PSMs, realizations and bindings | You can not store PIMs and PSMs in ebXML R/R, but ebXML conformant documents. |
| EXR6: Searching and browsing | ebXML R/R allows for both free text searches and browsing by category. |
| EXR7: Distributed queries | ebXML R/R allows for queries to be distributed along a web of other ebXML Registries/Repositories. |

Table 5-2 Evaluation of ebXML Registry/Repository

In this section we have evaluated ebXML Registry/Repository according to the requirements matrix for external repositories. Actually, ebXML Registry/Repository meets most of the requirements. However, it does not allow for storing PIMs and PSMs. I guess you could upload a file representing a PIM, but you would have to tell ebXML...
Registry/Repository that it is an ebXML compliant document, such as a CPP or something. That would be meaningless since no one would know what it really is.

However, the information model of ebXML Registry/Repository is flexible and easy to work with. So, ebXML shall be the basis we build upon when defining PIMRep External.

### 5.3 OGC Registry

The Open GIS Consortium Registry was introduced in chapter 4.3. In this section we will present an actual implementation of this specification called askthespider, and we will evaluate the OGC Registry according to the requirements matrix for external repositories.

Figure 5-5 shows a web site called www.askthespider.com which is actually the front end of an OGC compliant registry developed by Ionic Software who is also a partner in the ACE-GIS project. The registry is totally based on ebXML R/R but it is hard to detect from the GUI in the figure. askthespider lets you choose a geographic area in which to search for a Web service by marking an area on a map with the mouse pointer. You can choose between two different kinds of Web services to search for, which are Web Feature Services (WFS) and Web Map Services (WMS) which are both OGC conformant. You can also choose between some ISO categories such as farming, biota, boundaries et cetera.

Table 5-3 evaluates OGC Registry according to the requirements matrix for external repositories as given in table 3-1 in chapter 3.1. Note that we are evaluating OGC Registry as per specification, not askthespider in particular.
**EXR1: Content owner information**

In OGC Registry you can register contact information such as telephone number, email address and visiting address for both users and organizations. (Same as for ebXML R/R).

**EXR2: Access control**

OGC Registry has quite a sophisticated access control as multiple users can have edit access to one object. (Same as for ebXML R/R).

**EXR3: Event notifications**

OGC Registry allows for subscription to event notifications. (Same as for ebXML R/R).

**EXR4: Classification schemes**

In OGC Registry you can upload new classification schemes dynamically, and you can also give objects new characteristics arbitrarily by using slots. (Same as for ebXML R/R).

**EXR5: PIMs, PSMs, realizations and bindings**

You can not store PIMs and PSMs in OGC Registry, but ebXML and OGC conformant documents.

**EXR6: Searching and browsing**

OGC Registry allows for both free text searches and browsing by category. (Same as for ebXML R/R).
EXR7: Distributed queries

OGC Registry allows for queries to be distributed along a web of other OGC Registries. (Same as for ebXML R/R).

| EXR7: Distributed queries | OGC Registry allows for queries to be distributed along a web of other OGC Registries. (Same as for ebXML R/R). |

Table 5-3 Evaluation of OGC Registry

In this section we have evaluated OGC Registry according to the requirements matrix for external repositories. It meets most of the requirements, just like ebXML Registry/Repository on which it is based. However, it is meant to be a repository for documents specific to the GIS community. What we want is a repository for MDA related documents such as PIMs and PSMs. As OGC Registry is a mutation of ebXML Registry/Repository for GIS, PIMRep External will be a mutation of ebXML Registry/Repository for the MDA community.

5.4 dMOF

dMOF was introduced in chapter 4-5, and will now be evaluated according to the requirements for an internal repository as specified in table 3-2 in chapter 3-2.

Table 5-4 gives an evaluation of dMOF.

| INR1: Create modeling elements | dMOF allows you to create UML modeling elements such as class, attribute, operation, association et cetera. |
| INR2: Read modeling elements | dMOF allows you to read UML modeling elements such as class, attribute, operation, association et cetera. |
| INR3: Update modeling elements | dMOF allows you to update UML modeling elements such as class, attribute, operation, association et cetera. |
| INR4: Delete modeling elements | dMOF allows you to delete UML modeling elements such as class, attribute, operation, association et cetera. |
| INR5: Searching and browsing | dMOF does not allow you to search or browse for UML modeling elements such as class, attribute, operation, association et cetera. |
| INR6: Access control | dMOF does not allow for fine grained access control to all modeling elements. |
| INR7: Event notifications | dMOF does not allow for subscription to event notifications. |
| INR8: Concurrent user access | dMOF does not allow for concurrent user access. |
| INR9: Versioning | dMOF does not allow for versioning of modeling |
Table 5-3 Evaluation of dMOF

We see that dMOF meets some of the requirements, but not all of them. The requirements that are not met, could however be implemented as extensions. Anyway, we choose not to base our own PIMRep Internal on dMOF, because we think it is overhead to have to use CORBA.

5.5 Reference implementation of JMI

The Reference Implementation of JMI was introduced in chapter 4.6. Table 5-5 gives an evaluation of the Reference Implementation of JMI according to the requirements in table 3-2 in chapter 3-2.

<table>
<thead>
<tr>
<th>INR1: Create modeling elements</th>
<th>RI of JMI allows you to create UML modeling elements such as class, attribute, operation, association et cetera.</th>
</tr>
</thead>
<tbody>
<tr>
<td>INR2: Read modeling elements</td>
<td>RI of JMI allows you to read UML modeling elements such as class, attribute, operation, association et cetera.</td>
</tr>
<tr>
<td>INR3: Update modeling elements</td>
<td>RI of JMI allows you to update UML modeling elements such as class, attribute, operation, association et cetera.</td>
</tr>
<tr>
<td>INR4: Delete modeling elements</td>
<td>RI of JMI allows you to delete UML modeling elements such as class, attribute, operation, association et cetera.</td>
</tr>
<tr>
<td>INR5: Searching and browsing</td>
<td>RI of JMI does not allow you to search or browse for UML modeling elements such as class, attribute, operation, association et cetera.</td>
</tr>
<tr>
<td>INR6: Access control</td>
<td>RI of JMI does not allow for fine grained access control to all modeling elements.</td>
</tr>
<tr>
<td>INR7: Event notifications</td>
<td>RI of JMI does not allow for subscription to event notifications.</td>
</tr>
<tr>
<td>INR8: Concurrent user access</td>
<td>RI of JMI does not allow for concurrent user access.</td>
</tr>
<tr>
<td>INR9: Versioning</td>
<td>RI of JMI does not allow for versioning of modeling elements.</td>
</tr>
</tbody>
</table>

Table 5-4 Evaluation of the Reference Implementation of JMI
We see that the Reference Implementation of JMI meets some of the requirements, but not all of them. The requirements that are not met, could however be implemented as extensions. Also, Java is a grateful technology to work with, so our conclusion is that JMI could very well be the basis on which to implement PIMRep Internal.

5.6 Summary

In this chapter we have evaluated the technologies presented in the previous chapter.

UDDI, ebXML Registry/Repository and OGC Registry have been evaluated according to the requirements for the external repository.

dMOF and the Reference Implementation of JMI have been evaluated according to the requirements for the internal repository.

As a final note, we could have imagined not using MOF for the internal repository. Instead we could have hard coded a proprietary interface for models to a relational database. We could for example have a table Class, where all the classes created in a model gets put. However, there is no reason to do it this way, when we have technologies for generating these interfaces. It is faster to implement, and easier to make interoperable with other systems.
To sum up, MOF repositories provide a programmatical interface to edit a model while UDDI/ebXML registries/repositories provide a data structure and operations to store and retrieve Web services/models. We feel that for an inter-enterprise repository the right choice is a UDDI/ebXML approach, because it would be more natural to download the model to your own system before doing anything with it. The natural way to implement model persistence in the repository would be to store the models as XMI files.

Based on the ebXML registry/repository we have specified an application called PIMRep External which we present in detail in the next chapter. The PIMRep External application, along with PIMRep Internal, is the base of what I like to refer to as the PIMRep architecture. Figure 7-1 shows a high level view of the PIMRep Architecture focusing on the communication between servers and clients. The communication links between the servers are supposed to illustrate the fact that a query to a server is propagated through the web of servers which the initial server is aware of, until a match is found. The communication links between client and server illustrates the client accessing one of these interfaces at the server: LifeCycleManager, QueryManager or LinkManager. These interfaces will be thoroughly described in the next chapter.

Figure 6-1 also shows which part PIMRep Internal has in the architecture. PCs E, F and G are assumed to be in the same local area network. They have an instance of PIMRep Internal running on a workgroup server which they all access. The user of PC H is a single developer with PIMRep Internal running on his own PC.

Note that there is no direct communication between PIMRep Internal and PIMRep External. As models are being developed they reside in PIMRep Internal. When they are complete, developers can choose to make them publicly available by manually uploading them to an instance of PIMRep External.
Figure 6-1 Illustration of communication between servers and clients

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Server running PIMRep External</td>
</tr>
<tr>
<td>B</td>
<td>Server running PIMRep External</td>
</tr>
<tr>
<td>C</td>
<td>Server running PIMRep External</td>
</tr>
<tr>
<td>D</td>
<td>Server running PIMRep External</td>
</tr>
<tr>
<td>E</td>
<td>PC at company X accessing PIMRep External at server B</td>
</tr>
<tr>
<td>F</td>
<td>PC at company X</td>
</tr>
<tr>
<td>G</td>
<td>PC at company X</td>
</tr>
<tr>
<td>H</td>
<td>PC at company Y accessing PIMRep External at server D</td>
</tr>
</tbody>
</table>

Table 6-1 Legend for figure 6-1
The client side of the PIMRep Architecture can consist of a single computer or a number of computers in a Local Area Network. Typically, what happens at the client side is that one downloads UML models as XMI files. But, what are we going to do with these files? XMI is a MOF based format so it would be natural to have some kind of MOF based solution on the client side. My recommendation is depicted in the block diagram in figure 7-2. I suggest having a MOF repository as the backbone for a series of tools for operating on the model such as an IDE, a code generator and a visual UML modeling tool. I think the JMI technology should be used because it is easier than CORBA and you can find open source implementations such as the JMI Reference Implementation and NetBeans MDR. Also, DSTC have seized to make new versions of dMOF.

Figure 6-2 Client side infrastructure
6.1 Summary

In this chapter we have introduced the PIMRep Architecture. It consists of a server side repository called PIMRep External and a client side repository called PIMRep Internal. PIMRep External exposes an interface for uploading, finding and downloading models, while PIMRep Internal exposes an interface for creating and editing models.
7. PIMRep External

PIMRep (Platform Independent Model Repository) External is largely based on the ebXML Registry/Repository and the extensions of this in the OGC Registry. It will, however, be more aligned with the principles of Model Driven Architecture, as the repository will focus more on hosting platform independent models.

7.1 PIMRep External information model

For purposes of readability and presentation, we have divided the PIMRep External information model in four packages: AuditTrail, Core, Types and Security, as shown in figure 7-1. The conceptually most important classes are contained within the Core package. The AuditTrail view contains classes necessary to maintain an audit trail, such as classes describing users and their organizations. The security view includes classes governing the access control to operations on a given RegistryObject instance. The Types view includes enumerations of data types used throughout the entire PIMRep External information model.

Figure 7-1 Views of the PIMRep External information model
7.1.1 Core elements

In this section we shall present the Core package of the information model of PIMRep External. Figure 7-2 provides a UML class diagram of the Core package. UML is being used as prescribed by Martin Fowler in his book UML Distilled [2] and in the UML specification [1]. Table 7-1 gives a short description of each and one of the classes.

The most important class in the Core elements view is the RegistryObject class. This class holds the kind of information that is common to all other classes, such as name, identity and object type. The RegistryEntry class adds some more attributes, such as majorVersion, minorVersion and status. RegistryPackage, Service, ExtrinsicObject and ClassificationScheme are subclasses of RegistryEntry.

Association, ExternalIdentifier, ExternalLink, SpecificationLink, ServiceBinding, ClassificationNode and Classification are other subclasses of RegistryObject. All of these classes are described in detail in table 7-1. The class Slot is the extension mechanism of the PIMRep External information model. For every RegistryObject you can define as many Slots as you like. The class Slot has attributes for describing both the type and value of a property that you would like to assign to a RegistryObject.

Association is quite an important class in the PIMRep External information model as it defines relationships between RegistryObjects. For example, figure 7-3 shows how an Association can describe the relation between to objects of class ClassificationScheme. The association type here is Supersedes. The example is a newer version of the NAICS\(^5\) classification scheme superseding an older one. However, an Association object can describe any of the association types in the enumeration AssociationType in figure 7-7, and between any kinds of RegistryObjects. For example, we can have an Association of type SubmitterOf between a User and a PIM, or the Association type RelatedTo between two PIMS.

Figure 7-4 describes the difference between the classes ClassificationScheme and ClassificationNode. Industry is an instance of class ClassificationScheme with ClassificationNodes Health Care, Automotive and Retail. Classification is the class that for example binds the RegistryEntry yourDadsCarInc to ClassificationNode Automotive.

\(^5\) North American Industry Classification System
Class RegistryObject

The RegistryObject class is an abstract base class used by a lot of the other classes in the model.
<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RegistryEntry</td>
<td>This class extends the RegistryObject class and adds attributes used in lifecycle management, such as attributes describing version number.</td>
</tr>
<tr>
<td>Slot</td>
<td>This class provides a way to add arbitrary attributes to instances of the RegistryObject Class. This enables extensibility within the information model.</td>
</tr>
<tr>
<td>Association</td>
<td>An instance of the Association Class may be used to associate RegistryObject instances. Figure 7-3 taken from [21] shows an Association between a new version of the NAICS ClassificationScheme and an older version of the NAICS ClassificationScheme.</td>
</tr>
<tr>
<td>ExternalIdentifier</td>
<td>Instances of this class provide additional identifier information to RegistryObject such as DUNS number or Social Security Number.</td>
</tr>
<tr>
<td>ExternalLink</td>
<td>This class has a URI that is used to point to content external to the registry, for example the home page of an organization.</td>
</tr>
<tr>
<td>ClassificationScheme, ClassificationNode</td>
<td>The UML Object Diagram in figure 7-4 taken from [21] shows the use of the classes ClassificationScheme and ClassificationNode. Examples of real world classification schemes are NAICS, UNSPSC and ISO 3166.</td>
</tr>
<tr>
<td>Classification</td>
<td>This class classifies a RegistryObject instance by referencing a node defined within a particular ClassificationScheme.</td>
</tr>
<tr>
<td>RegistryPackage</td>
<td>RegistryPackage instances group logically related RegistryObject instances together.</td>
</tr>
<tr>
<td>Service</td>
<td>Instances of this class provide information on services, such as Web services. A Service has a collection of ServiceBindings.</td>
</tr>
<tr>
<td>ServiceBinding</td>
<td>This class represents technical information on a specific way to access a specific interface offered by a Service instance. A ServiceBinding has a collection of SpecificationLinks.</td>
</tr>
<tr>
<td>SpecificationLink</td>
<td>This class provides the linkage between a ServiceBinding and one of its technical specifications that describes how to use the service.</td>
</tr>
</tbody>
</table>
using the ServiceBinding. The SpecificationLink has an attribute specificationObject that typically is an ExtrinsicObject instance representing the technical specification (e.g., a WSDL document or a CORBA IDL document).

Class ExtrinsicObject

This class provides metadata that describes submitted content whose type is not intrinsically known to the registry. It must therefore be described as a mime type. Examples of content described by ExtrinsicObject are Collaboration Protocol Profiles (ebCPP) and WSDL.

Class FileExtrinsicObject

This class is used to address files in the repository, such as PIMs and PSMs.

Table 7-1 Description of the classes of the Core package

Figure 7-3 Association between NAICS1997 and NAICS2001
7.1.2 AuditTrail

In this section we describe the AuditTrail package the PIMRep External information model. Figure 7-5 provides a UML class diagram of the AuditTrail package, while table 7-2 gives a short description of each of the classes.

The main class of the AuditTrail view is the AuditableEvent class. The AuditableEvent class has information of what kind of event, such as creation, deletion and deprecation, happened to which RegistryObject at which time and was triggered by which User.

The User class holds information such as telephone number, email address, person name and postal address. A User typically belongs to an Organization. An Organization also has information such as telephone number and postal address.
Figure 7-5 AuditTrail view of the PIMRep information model

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AuditableEvent</strong></td>
<td>This class record events that effects a change in a RegistryObject. An ordered collection of AuditableEvent instances provide a complete audit trail for a RegistryObject.</td>
</tr>
<tr>
<td><strong>User</strong></td>
<td>This class represents users that have registered with a registry. It is also used in an AuditableEvent to keep track of the identity of the editor.</td>
</tr>
<tr>
<td><strong>PostalAddress</strong></td>
<td>This class defines the attributes of a postal address.</td>
</tr>
<tr>
<td><strong>EmailAddress</strong></td>
<td>This class defines the attributes of an email address.</td>
</tr>
<tr>
<td><strong>Organization</strong></td>
<td>Instances of this class provide information on organizations such as a submitting organization. An organization may have a reference to a parent organization.</td>
</tr>
</tbody>
</table>
Class TelephoneNumber | This class defines the attributes of a telephone number.

Table 7-2 Description of the classes of the AuditTrail package

7.1.3 Security

In this section we will present the Security view of the PIMRep External information model. Figure 7-6 provides a UML class diagram. A Principal is someone trying to access PIMRep External, be it a human being or some piece of software. The Principal has a set of PrivilegeAttributes. These include at least an Identity, but may also include different Roles or memberships to Groups.

On the other hand, every RegistryObject in PIMRep External has an AccessControlPolicy. For every method in the RegistryObject, the AccessControlPolicy has a Permission. The Permission has a set of Privileges which again has a set of PrivilegeAttributes. If any of these PrivilegeAttributes matches any of the Principal’s PrivilegeAttributes the Principal may invoke the method in question.
7.1.4 Types

In this section we will describe the Types view of the PIMRep External information model. Figure 7-7 shows a UML class diagram of this view. The Types package consists of enumerations defining types used throughout PIMRep External. The RegistryObject class has an attribute objectType. In addition to some of the values known from ebXML, PIMRep’s ObjectType enumeration includes WSDL, AdhocDocument, PIM, PSM, Realization and IDL. The meanings of these types are quite self understood. WSDL means that the RegistryObject is a WSDL document. PIM and PSM mean that we have documents according to these terms in MDA. WSDL and IDL are concrete examples of Realizations. Realization can also be much more, for instance Java et cetera. If you have an undefined document, for example a Powerpoint Presentation, you type it AdhocDocument.
The StatusType enumeration refers to a RegistryEntry, which can either be Approved, Deprecated, Submitted or Withdrawn. A RegistryEntry also has some level of stability, which can be Dynamic, DynamicCompatible or Static.

An AuditableEvent has a type which can be any of the values of the enumeration EventType: Created, Deleted, Deprecated, Updated or Versioned.

The enumeration AssociationType describes associations that can exist between any two RegistryObjects. And nodeType is an attribute of ClassificationScheme which can either be EmbeddedPath, NonUniqueCode or UniqueCode.

![Figure 7-7 Types view of the PIMRep External information model](image)

### 7.1.5 Summary

In this section we have presented the PIMRep External information model. It consists of the four views Core Elements, AuditTrail, Security and Types. The Core elements view includes the most important classes for making up a registration. The AuditTrail view
includes classes for keeping track of AuditableEvents. The Security view includes classes for providing access control. And the Types view includes enumerations of different types used throughout the information model.

7.2 PIMRep External services specification

In this section we shall present the PIMRep External services specification. Figure 7-8 shows a UML component diagram of PIMRep External. As you see, PIMRep External consists of one major component offering three different interfaces: LinkManager, QueryManager and LifecycleManager.

![UML component diagram](image_url)

Figure 7-8 A UML component diagram of the PIMRep External application with the three interfaces that it implements
### 7.2.1 LifeCycleManager interface

This is the interface offered by PIMRep External that implements the object lifecycle management functionality of PIMRep External. Figure 7.9 presents the LifeCycleManager interface in UML syntax. Table 7.1 provides a summary of the semantics of each method in the LifeCycleManager interface.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>acceptObjects(AcceptObjectsRequest req)</code></td>
<td>This method accepts one or more objects to PIMRep External during object relocation. The syntax of the structures RegistryResponse and AcceptObjectsRequest are as specified in [21]</td>
</tr>
<tr>
<td><code>approveObjects(ApproveObjectsRequest req)</code></td>
<td>This method approves one or more previously submitted objects. The syntax of the structure ApproveObjectsRequest is as specified in [21]</td>
</tr>
<tr>
<td><code>deprecateObjects(DeprecateObjectsRequest req)</code></td>
<td>This method deprecates one or more previously submitted objects. The syntax of the structure DeprecateObjectsRequest is as specified in [21]</td>
</tr>
<tr>
<td><code>removeObjects(RemoveObjectsRequest req)</code></td>
<td>This method removes on or more previously submitted objects. The syntax of the structure RemoveObjectsRequest is as specified in [22]</td>
</tr>
<tr>
<td><code>submitObjects(SubmitObjectsRequest req)</code></td>
<td>This method submits one or more objects and related metadata such as Associations and Classifications. The syntax of the structure SubmitObjectsRequest is as specified in [22]</td>
</tr>
</tbody>
</table>
This method updates one or more previously submitted objects. The syntax of the structure UpdateObjectsRequest is as specified in [22]

This method adds slots to one or more registry entries. The syntax of the structure AddSlotsRequest is as specified in [22]

This method relocates one or more objects from one registry to another. The syntax of the structure RelocateObjectsRequest is as specified in [22]

This method removes specified slots from one or more registry entries. The syntax of the structure RemoveSlotsRequest is as specified in [22]

Table 7-3 Method summary of LifeCycleManager

7.2.2 QueryManager interface

This is the interface offered by PIMRep External that implements the query functionality of PIMRep External. Figure 7.10 presents the QueryManager interface in UML syntax. Table 7.2 provides a summary of the semantics of each method in the QueryManager interface.

This method submits a request to PIMRep External to get event notifications. The syntax of the structures GetNotificationsResponse and GetNotificationsRequest are as specified in [22]

This method submits an ad hoc query request to PIMRep External. The syntax of the structures AdhocQueryResponse and AdhocQueryRequest are as specified in [22]
RegistryObject getRegistryObject(String id)  
This method submits a request to PIMRep External to get the RegistryObject that matches the specified id. The syntax of the structure RegistryObject is as specified in [22].

RepositoryItem getRepositoryItem(String id)  
This method submits a request to PIMRep External to get the repository item that matches the specified id, which is the same as the id of the ExtrinsicObject that catalogs this repository item. The syntax of the structure RepositoryItem is as specified in [21].

Table 7-4 Method summary of QueryManager

7.2.3 LinkManager interface

This is the interface offered by PIMRep External that implements the management of links between different instances of PIMRep External to arrange for distributed queries. Figure 7.10 presents the LinkManager interface in UML syntax. Table 7.2 provides a summary of the semantics of each method in the LinkManager interface. The LinkManager interface is based on the LinkManagement interface described in ISO/IEC 13235 [27].

Figure 7-11 The LinkManager interface
In this section we have described the PIMRep External services specification. We have seen that it consists of three interfaces which include the LifecycleManager interface, the QueryManager interface and the LinkManager interface. The LifecycleManager interface has methods for tasks such as accepting, approving and deprecating RegistryObjects. The QueryManager interface has methods for submitting queries to PIMRep External. The LinkManager interface has methods for maintaining links to other instances of PIMRep External.
In this chapter we will apply the PIMRep External application to the Modusa gas dispersion Web service student project presented in chapter 2.2. We will illustrate the use of PIMRep External through tentative dialog box interfaces made in a drawing tool and UML object diagrams, showing the state of the involved objects.

8.1 Registering organization and user

Figure 8-1 and 8-2 show graphical user interfaces for registering organization and user respectively. We see that most of the information is typed by the user in text boxes or chosen from drop down menus (such as country). Note that we have omitted the log in procedure which obviously has to take place. We could for instance imagine that a password is mailed to the user first time he registers.
Figure 8-1 Register Organization

Figure 8-1 shows the dialog box for registering an Organization. We can fill in name, city, country, postal code, street, street number and phone number.
Figure 8-2 Registering User

Figure 8-2 shows the dialog box for registering user. We provide first name, last name, email address and organization.

Figure 8-3 shows the state of the involved objects of the PIMRep External information model after the user Hans Stubberud and the organization SINTEF have been registered. Note that, including to the information provided by the user, the objects inheriting from RegistryObject, also have been assigned a universally unique identifier based on a DCE 128-bit UUID.
8.2 Registering a platform independent model

Now that we have registered ourselves as the user Hans Stubberud from the company SINTEF we are ready to upload our Platform Independent Model of the Modusa gas dispersion Web service. We suppose that we have the PIM in figure 2.5 in a file in XMI format on our local system. The graphical user interface for registering the object is as depicted in figure 8.4. First, we provide information in the text boxes and from the combo boxes. Notice that the choices to choose between can be found in the enumerations in the types view of the PIMRep External information model. Then we upload the file containing the PIM from our local file system to PIMRep External.
Also, we suppose that the two classification schemes UN/SPC and ISO 3166 familiar from the problem example Infrastructure of Norwegian e-commerce presented in chapter 2.1 have been uploaded to PIMRep External. This allows for us to classify our Modusa Web service PIM according to these two taxonomies clicking one of the Classify buttons. We could for example imagine an expandable tree view showing up in a dialog box letting us mark the taxonomy node of interest. This will later allow for enhanced possibilities of searching and browsing.

Figure 8-4 Registering ModusaWebServicePIM

Figure 8.5 shows the FileExtrinsicObject after registering the object. Note that also this object has been assigned a universally unique identifier. Also be aware of the fact that there is an Association object of type SubmitterOf connecting this FileExtrinsicObject and the Hans Stubberud User object.
We would use the same approach for uploading the PSM of figures 2-7 and 2-8. Only then we would have registered the object as type PSM.

<table>
<thead>
<tr>
<th>: FileExtrinsicObject</th>
</tr>
</thead>
<tbody>
<tr>
<td>id = urn:uuid:801c8fcd-3176-41a6-81f5-dc23ceff08h</td>
</tr>
<tr>
<td>name = ModusaWebServicePIM</td>
</tr>
<tr>
<td>objectType = PIM</td>
</tr>
<tr>
<td>majorVersion = 1</td>
</tr>
<tr>
<td>minorVersion = 0</td>
</tr>
<tr>
<td>stability = Static</td>
</tr>
<tr>
<td>status = Submitted</td>
</tr>
<tr>
<td>mimeType = text/xml</td>
</tr>
<tr>
<td>contentURL = &lt;&lt;Pointer to a location in the repository&gt;&gt;</td>
</tr>
</tbody>
</table>

Figure 8-5 FileExtrinsicObject

### 8.3 Searching and browsing for objects

Now that we have uploaded ModusaWebServicePIM to PIMRep External, we want it to be accessible to other people so they can download it and make their own legacy systems interact with our service. This can basically be accomplished in two ways; by searching and by browsing. Say we classified ModusaWebServicePIM by ISO 3166 to be relevant to the geographic area Norway. Then the model can be found as depicted in figure 8-6. One specifies the geographic region to be Norway and the object type to be PIM, then the ModusaWebServicePIM will be found in the Results list when pressing the Browse.. button. The ModusaWebServicePIM XMI file can now be downloaded by pressing the Download Selected Item button.

Another way to find the ModusaWebServicePIM is to perform a free text search as depicted in figure 8-7. You know that you are looking for a PIM with the word Modusa in the title, but you do not know how it is classified. By searching for *Modusa* you are able to find the ModusaWebServicePIM.
Figure 8-6 Browsing by ISO 3166

Figure 8-6 is a figure I made with the Visio drawing tool. It is supposed to be a tentative screen shot of a client accessing PIMRep External. This is just an example of what such a client might look like. It is, for example, likely that it also would be implemented with a HTML user interface, and that you could run it in a web browser. Figure 8-6 is a Windows form, which is the user interface of a standard Windows application. We could also imagine thicker clients. The PIMRep External client functionality could for example be included in a more all-inclusive application for software development, also incorporating for example a UML modeling tool or an IDE.

ISO 3166 is a classification scheme of the International Standards Organization which basically maps each and one of all the names of the countries of the world to two letter codes. For example, the value pair Norway and NO belong together in the classification scheme. There are also more fine-grained geographical sub values within the value NO, for example the names of cities, but we shall not be concerned with that for our purposes.

The idea behind the dialog box in figure 8-6, is that you choose a geographic region from a combo box according to ISO 3166. The reader is probably used to combo boxes listing up all the countries from other applications. Then you choose what object type to look for in another combo box. You will typically be able to look for PIMs, PSMs, different kinds of realizations like WSDL for instance, or object types like Organization for that matter. Then you press the Browse button, and registry objects matching the specified geographic region and object type will appear in the list box to the left. In our example there is only one object matching the criteria, the ModusaWebServicePIM. The idea is that if you mark any of the items in the list box and press the button Download selected item, you get the download file dialog box, known from many applications, asking you were to store the file. In our example, the file is a MOF XMI file containing the PIM of the Modusa gas dispersion Web service.
While the figure 8-6 depicted a browsing scenario, figure 8-7 shows a dialog box for searching for an object. It is not that different from the dialog box in figure 8-6. The Object type field is the same, but instead of choosing a geographic region from a combo box, you are supposed to enter a search string. The search string in our example is *Modusa*. The asterisk, ‘*’, is a wild card. So, when hitting the Search button the application will return all registry objects containing the string ‘Modusa’ in its name. In this case, the only result is ModusaWebServicePIM. By marking this in the Results list box and pressing the Download selected Item button, we are able to download the XMI file representing the PIM of the Modusa gas dispersion Web service to our local computer.

**Figure 8-7 Searching for object**

8.4 PIM versus PSM

In this thesis we have talked a lot about PIMs and PSMs and the Model Driven Architecture. And we believe it is the right approach to software engineering. However, MDA has yet to find its final form.

I mean, what is really a PIM? In the Modusa gas dispersion Web service we have used quite a clearly set out UML class diagram as an example. But a PIM could also be a UML activity diagram, and so could a PSM. In the Infrastructure of Norwegian e-commerce project one had a certain definition of what constitutes a PIM and what constitutes a PSM. The same way that ebXML defines which documents are needed to describe how to do business.

Some other people from SINTEF have developed a scenario for the ACE-GIS project that is not too different from the student scenario. This scenario is described in the ACE-GIS deliverable D5.2 Model-driven Service Composition [28].

---

91
Figure 8-8 provides a platform specific model of this scenario. It says which external Web services to call, what to do with the results, and in which order. Inesc, which is another partner in ACE-GIS, has actually developed a so called orchestration engine which can actually execute this model after converting it to some proprietary XML format called Workflow XML.

Clearly, it could be beneficial to share such a model in a repository. Maybe you would like to share a model that makes it easier for other people to do business with you, for instance.

So, even though the example in the Modusa gas dispersion Web service is a PIM and a class diagram, PIMRep External should be capable of also handling PSMs and activity diagrams, for example. The discussion of what constitutes PIMs and what constitutes PSMs, and when to use them, is still going on.
8.5 Summary

In this chapter we have shown how PIMRep External is to be used in real life. The case study has been the Modusa gas dispersion Web service introduced in chapter two. We have demonstrated how PIMRep External can register information on organization and user, how a platform independent model can be uploaded, and how it can be found by other parties again.

We have also discussed the issue of what constitutes a PIM and what constitutes a PSM. Finally we have introduced an activity diagram from the ACE-GIS as an alternative to what kind of models one would want to upload to PIMRep External.

When we developed the Modusa gas dispersion Web service we had to search Google to find the Web services we wanted to use, or implement dummy services if we could not find any real ones. If there had been a PIMRep External available at that time, our job would have been much easier. This illustrates the need for PIMRep External.
9. Evaluation of PIMRep External

Table 9-1 evaluates PIMRep External according to the requirements matrix for external repositories as given in table 3-1 in chapter 3.1.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXR1: Content owner information</td>
<td>In PIMRep External you can register contact information such as telephone number, email address and visiting address for both users and organizations.</td>
</tr>
<tr>
<td>EXR2: Access control</td>
<td>PIMRep External has quite a sophisticated model for access control as pointed out in chapter 7.1.3. Access to methods in RegistryObjects can be granted to both different roles, groups and identities.</td>
</tr>
<tr>
<td>EXR3: Event notifications</td>
<td>PIMRep External allows for subscription to event notifications.</td>
</tr>
<tr>
<td>EXR4: Classification schemes</td>
<td>In PIMRep External you can upload new classification schemes dynamically, and you can also give objects new characteristics arbitrarily by using slots.</td>
</tr>
<tr>
<td>EXR5: PIMs, PSMs, realizations and bindings</td>
<td>The purpose of PIMRep External is to store documents related to the MDA framework for software development. This means PIMs and PSMs, with an extra emphasis on PIMs, as pointed out by the name PIMRep, because once a mapping is defined you should be able to generate PSMs from PIMs. To make PIMRep more pragmatic, we also allow for storing realization documents such as WSDLs and IDLs.</td>
</tr>
<tr>
<td>EXR6: Searching and browsing</td>
<td>PIMRep External allows for both free text searches and browsing by category.</td>
</tr>
<tr>
<td>EXR7: Distributed queries</td>
<td>PIMRep External allows for queries to be distributed along a web of other instances of PIMRep External.</td>
</tr>
</tbody>
</table>

Table 9-1 Evaluation of PIMRep External

We see that PIMRep External meets all the requirements we initially defined for the external repository.
10. PIMRep Internal

In this chapter we will present the PIMRep Internal information model and services specification.

10.1 PIMRep Internal information model

The PIMRep Internal information model consists of the two packages Core and Security as shown in figure 10-1.

![Figure 10-1 Views of the PIMRep Internal information model](image)

The core view of the information model of PIMRep Internal is quite similar to that of UML. In fact, the only difference is that the class ModelElement is extended with the attributes presented in figure 10-2.

![Figure 10-2 The ModelElement class](image)

The Security information model for PIMRep Internal is the same as that for PIMRep External described in chapter 7.2.3. This is quite a sophisticated and generic information model for access control.

10.2 PIMRep Internal services specification

The PIMRep Internal services specification is basically identical to the Java Metadata Interface. The generated interface will, however, be slightly different as we have added a few attributes to the ModelElement class.
Also, we have specified a very easy to use and to understand interface for creating and deleting classes and attributes called IEasy. IEasy is depicted in figure 10-3 and explained in table 10-1. It is, however fully functional, mostly meant for illustrative purposes.

<table>
<thead>
<tr>
<th>function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createClass(Name : String)</td>
<td>This function allows us to create a new class. The name of the class is supplied as an argument.</td>
</tr>
<tr>
<td>createAttribute(Class : String, Name : String, Type : String)</td>
<td>This function allows us to create a new attribute with a given type and name for a given class.</td>
</tr>
<tr>
<td>deleteClass(Name : String)</td>
<td>This function allows us to delete a class by supplying the class name as argument.</td>
</tr>
<tr>
<td>deleteAttribute(Class : String, Name : String, Type : String)</td>
<td>This function allows us to delete a given attribute in a given class.</td>
</tr>
<tr>
<td>existsClass(Name : Class) : Boolean</td>
<td>Returns true if there exists a class with the given name, otherwise it returns false.</td>
</tr>
<tr>
<td>existsAttribute(Class : String, Name : String, Type : String) : Boolean</td>
<td>Returns true if there exists an attribute with the given name and type for the given class, otherwise it returns false.</td>
</tr>
</tbody>
</table>

Figure 10-3 Interface IEasy

<table>
<thead>
<tr>
<th>function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createClass</td>
<td>This function allows us to create a new class. The name of the class is supplied as an argument.</td>
</tr>
<tr>
<td>createAttribute</td>
<td>This function allows us to create a new attribute with a given type and name for a given class.</td>
</tr>
<tr>
<td>deleteClass</td>
<td>This function allows us to delete a class by supplying the class name as argument.</td>
</tr>
<tr>
<td>deleteAttribute</td>
<td>This function allows us to delete a given attribute in a given class.</td>
</tr>
<tr>
<td>existsClass</td>
<td>Returns true if there exists a class with the given name, otherwise it returns false.</td>
</tr>
<tr>
<td>existsAttribute</td>
<td>Returns true if there exists an attribute with the given name and type for the given class, otherwise it returns false.</td>
</tr>
</tbody>
</table>

Table 10-1 Explanation of functions of IEasy

### 10.3 Implementational details

In this section we are going to give some pointers on how to implement PIMRep Internal quite seamlessly. For sakes of convenience, we are going to assume that the information model of PIMRep Internal is the simple information model in figure 10-4.

The information model in figure 10-4 has two classes; Class and Attribute. Class has one attributed called Name of type String, and Attribute has attributes Name and Type, both of type String. A Class can refer to another Class as super, this allows for inheritance. A Class can have several Attributes, while an Attribute only belongs to one Class. The association between Class and Attribute is called ClassAttribute.
The metamodel in figure 10-4 was made in the UML modeling tool Rational Rose. Rational Rose stores its files in a proprietary file format with the extension .mdl. You can however download an add-in which allows you to convert your models to XMI files. You can convert your models to either MOF compatible XMI files or UML compatible XMI files. The model in figure 10-4, we converted into a MOF compatible XMI file, because we were going to use it as input to a MOF compatible tool.

The Java Integrated Development Environment called NetBeans has a feature called MDRepository. MDRepository is actually an implementation of JMI. So what we did, was that we uploaded our metamodel to MDRepository as an XMI file. Then we got MDRepository to generate JMI interfaces for our metamodel.

![Diagram of a simple information model]

**Figure 10-4 Simple information model**

MDRepository generated several Java files for us, one of them being SimplePimrepPackage.java. The metamodel in figure 10-4 is actually called SimplePimrep. The file SimplePimrepPackage.java includes an interface called SimplePimrepPackage which extends javax.jmi.reflect.RefPackage. The interface has methods such as getClass() and getAttribute() which lets you get factory or proxy objects of the classes in the metamodel. There are also methods for getting the associations of the metamodel, such as getClassAttribute() which returns a factory object for the association ClassAttribute. Here is the file SimplePimrepPackage.java:

```java
package simplepimrep;

/**
 * SimplePimrep package interface.
 */
public interface SimplePimrepPackage extends javax.jmi.reflect.RefPackage {
  /**
   * Returns Class class proxy object.
   * @return Class class proxy object.
   */
```
Another file created by MDRepository is AttributeClass.java. This file includes the interface AttributeClass which extends javax.jmi.reflect.RefClass. This interface has factory methods for Attribute, such as CreateAttribute which allows you to create an instance of class Attribute with a given name and type. Here is the file AttributeClass.java:

```java
package simplepimrep;

/**
 * Attribute class proxy interface.
 */
public interface AttributeClass extends javax.jmi.reflect.RefClass {
    /**
     * The default factory operation used to create an instance object
     * @return The created instance object.
     */
    public Attribute createAttribute();
    /**
     * Creates an instance object having attributes initialized by the passed
     * values.
     * @param Name
     * @param Type
     * @return The created instance object.
     */
    public Attribute createAttribute(java.lang.String name, java.lang.String type);
}
```
MDRepository generates similar files for Class and the different associations such as ClassAttribute. In addition to the generated interface, JMI offers a lot of generic functionality too. With the generated and reflective interfaces one can easily implement an interface such as IEasy in the previous section. The reflective interfaces allow you to perform more generic operations, like deleting an object for instance. MDRepository also offers persistence. So when the interfaces have been generated, you can start programming against it at once. The UML tool Poseidon is an example of an application which is based on a meta model repository generated by MDRepository.

MDRepository, which has been used as an example in this chapter, was not discussed in chapter 4 Existing technologies, because it was not available to us at that time.

### 10.4 Summary

In this chapter we have presented the PIMRep Internal information model and services specification. We have also demonstrated how PIMRep Internal easily can be implemented using a JMI tool such as MDRepository.

We said that we wanted the information model of PIMRep Internal to be quite similar to UML. However, we included some attributes to allow for versioning of model elements, and we added a Security view. If any of these features become standard UML, we might as well adopt the standard. If the Security view really should be implemented as a part of PIMRep Internal is a matter of discussion. One could argue that this is overhead, and that security should be implemented as an orthogonal component.

We could also imagine instances of PIMRep Internal not offering interfaces to all of UML, but maybe just a subset, such as class diagrams. In some cases that could be all that you need.
To further illustrate the use of PIMRep Internal, we have created some tentative screenshots illustrating a client application accessing PIMRep Internal.

The case study is the same as before, the Modusa gas dispersion Web service.

The client application in this example is actually a UML modeling tool. However, it is not a graphical drawing tool as you would assume. I mean, most UML modeling tools are graphical.

No, this is a UML tool for business users who really do not know anything of UML, and nor do they want to. This is a tool that guides the user through the process of declaring classes and attributes step by step.

In figure 11-1 we declare class WeatherData which you remember from figure 2-6 Modusa PIM.

![Create new class](image)

**Figure 11-1 Declaring class WeatherData**

Figure 11-2 shows how we declare the attribute temperature as a Double in class WeatherData. Similarly we define attributes windDirection, location, windSpeed and humidity. You see that the user interfaces are foolproof, and can be exposed even to business users.
Figure 11-2 Declaring attribute temperature

Figure 11-3 shows a search function. We would like to see if there exists any classes with *weather* as part of its name. We see that there is a class called WeatherData, and we can push the button *View selected class* to get more information about this class.

Figure 11-3 Searching for *Weather*

We see from this example that PIMRep Internal makes it very easy to implement new modeling tools such as the one shown above. That is because PIMRep Internal has a strictly defined interface which lets you operate on metadata according to the syntactical rules of UML.

It is clear that it would be beneficial to have a repository like PIMRep Internal as a backbone for all access to models within a secure intranet.
11.1 Summary

In this chapter we have applied PIMRep Internal to the Modusa gas dispersion Web service. We have introduced some tentative screenshots of a client using PIMRep Internal. The client allows modelers to define classes and attributes. This is just an example client, one could also imagine more advanced clients with functionality similar to UML tools like Rational Rose for instance.
## 12. Evaluation of PIMRep Internal

Table 12-1 evaluates PIMRep Internal according to the requirements matrix for internal repositories as given in table 3-2 in chapter 3-2.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INR1: Create modeling elements</td>
<td>PIMRep Internal allows you to create UML modeling elements such as class, attribute, operation, association et cetera.</td>
</tr>
<tr>
<td>INR2: Read modeling elements</td>
<td>PIMRep Internal allows you to read UML modeling elements such as class, attribute, operation, association et cetera.</td>
</tr>
<tr>
<td>INR3: Update modeling elements</td>
<td>PIMRep Internal allows you to update UML modeling elements such as class, attribute, operation, association et cetera.</td>
</tr>
<tr>
<td>INR4: Delete modeling elements</td>
<td>PIMRep Internal allows you to delete UML modeling elements such as class, attribute, operation, association et cetera.</td>
</tr>
<tr>
<td>INR5: Searching and browsing</td>
<td>PIMRep Internal allows you to search or browse for UML modeling elements such as class, attribute, operation, association et cetera.</td>
</tr>
<tr>
<td>INR6: Access control</td>
<td>PIMRep Internal allows for fine grained access control to all modeling elements.</td>
</tr>
<tr>
<td>INR7: Event notifications</td>
<td>PIMRep Internal does not allow for subscription to event notifications at this time.</td>
</tr>
<tr>
<td>INR8: Concurrent user access</td>
<td>PIMRep Internal does not allow for concurrent user access at this time.</td>
</tr>
<tr>
<td>INR9: Versioning</td>
<td>PIMRep Internal allows for versioning of modeling elements.</td>
</tr>
</tbody>
</table>

### Table 12-1 Evaluation of PIMRep Internal

PIMRep does not fulfill requirement 7 for internal repositories. It does not allow for subscription to event notifications. This could easily have been accomplished by applying the AuditableEvent view from PIMRep External to PIMRep Internal. We do however feel...
that this functionality is not necessarily needed for PIMRep Internal. It is to be used by one set of developers while PIMRep External is exposed to a much broader audience.

Concurrent user access could be implemented by time, but maybe as a plug in instead of an integral part of PIMRep Internal. By concurrent user access we refer to functionality like that of CVS (Concurrent Versions System) or Microsoft Visual SourceSafe.

PIMRep Internal allows for versioning of modeling elements as a consequence of the attributes that we added to the ModelElement class. There is, however, talk about future versions of MOF specifying more sophisticated ways of accomplishing versioning.

PIMRep Internal as described in this thesis can easily be implemented based on the technology of the Reference Implementation of JMI.
13. Conclusions and Future Work

The purpose of this thesis was to specify a repository to be used in the context of software development in a service oriented infrastructure. We came to the conclusion that we needed two repositories; PIMRep External and PIMRep Internal.

The contributions of this thesis are summarized in the following section.

13.1 Contributions

In this section we will sum up the contributions of this thesis.

Evaluation of existing technologies.
In this thesis we thoroughly describe some existing technologies and evaluate them according to some requirements that we have defined. The requirements are based on the problem examples and our expectations of the investigated technologies. The investigated technologies include UDDI, ebXML Registry/Repository, OGC Registry and the MOF technologies dMOF and the Reference Implementation of JMI.

PIMRep External.
In this thesis we have defined an application called PIMRep External. PIMRep External is a repository in the context of software engineering in a service oriented infrastructure, and is supposed to be accessible to everyone with an Internet connection.

PIMRep External allows you to register contact information, it has a sophisticated model for access control, it allows for subscription to event notifications, it allows for classification according to all kinds of schemes, it stores PIMs, PSMs and realization documents, it allows for free text searches and browsing by category, and it can distribute these queries to other instances of PIMRep External.

We have also made a proof-of-concept application of the case study Modusa gas dispersion Web service to PIMRep External.

PIMRep Internal.
In this thesis we have also defined an application called PIMRep Internal. This is sort of the sister application to PIMRep External. It is used by a single set of users (mostly likely applications) for editing models and generating code.

PIMRep Internal lets you create, read, update and delete UML modeling elements. It also lets you browse and search for such elements. PIMRep Internal implements a fine grained access control to modeling elements, and it also allows for versioning.
We have also made a proof-of-concept application of the case study Modusa gas dispersion Web service to PIMRep Internal.

13.2 Future work

In this section we will suggest future work for PIMRep External and PIMRep Internal.

**PIMRep External.**
I think the specification of PIMRep External is really thorough, and does not need any more work at this time. The next step is to implement a production version of it, and then tried it out in real life. Then one can put the experience from that into a next generation PIMRep External.

**PIMRep Internal.**
PIMRep Internal should also be put up and running. A lot of client applications would also have to be implemented for it to be useful. Also, some concurrent user access should be worked out.

As a final thought, I would like to say something about the division between PIMRep External and PIMRep Internal. It is really a division of responsibility. There is no reason that, for example, a PIMRep Internal could not be implemented as a part of a PIMRep External. This would, however, be some time in the future. Suppose you are searching for a model in a PIMRep External. When you find it you will be able to access it in your web browser, because the external repository also implements the functionality of PIMRep Internal!

One way to merge PIMRep Internal and PIMRep External would be to incorporate functionality from PIMRep External in PIMRep Internal. For example, the AuditTrail view from PIMRep External could be adopted by PIMRep Internal. But would it be part of the meta model used to generate the JMI interface for PIMRep Internal, or just imported as a Java library when programming the PIMRep Internal? Questions like this are, in our opinion, interesting foundations for future work.
In this chapter we present some UML class diagrams taken from [23] that explains the most important concepts in MOF. In order to start programming against the MOF or JMI interface it is vital to understand the MOF information model. For a listing of MOF interfaces I refer to [23] and [13].

### A.1 Common Superclasses

Figure 13-1 shows some of the higher-level superclasses of MOF in detail to illustrate what gets inherited. All elements inherit a name and an annotation from ModelElement. Also, all ModelElements are contained in one and only one Namespace.
Figure 13-1 Common superclasses

A.2 Containment Hierarchy

Figure 13-2 shows how the classes of the MOF model are contained within each other. Containment is the most important relationship in the MOF model. It is for example used to relate Classes to their Operations and Attributes.
A.3 Types

Figure 13-3 shows an inheritance diagram of the MOF types – Class, Association and DataType. They will be further described in sub-sections.
A.3.1 Class

A class is a classification of a set of objects exhibiting the same state and behavior. This can for example be represented through operations and attributes.

A.3.2 Association

Figure 13-4 shows Associations in more detail. Note that the containment link between Association and AssociationEnd is not included in the figure.
A.3.3 Datatypes

Figure 13-5 shows the subtypes of DataType.
A.4 Features

Figure 13-6 shows an inheritance diagram for features. Features are important because ModelElements, especially classes, are largely defined by a composition of Features that they contain.

A.5 Tags

The Tag model element is the extension mechanism of the MOF model. As we can see in figure 13-7 a tag has an id that denotes its kind and a collection of values associated with it.
Figure 13-7 Tags
References


