DESIGN AND IMPLEMENTATION OF A DATASPACE MODEL FOR E-SCIENCE APPLICATIONS

eingereicht von:

Adnan Muslimovic

DIPLOMARBEIT

zur Erlangung des akademischen Grades
Magister rerum socialium oeconomicarumque
Magister der Sozial- und Wirtschaftswissenschaften
(Mag.rer.soc.oec.)

Studienrichtung: **Wirtschaftsinformatik**
Studienkennzahl: **A175**
Begutachter: **Univ.-Prof. Dr. Peter Brezany**

Wien, im November 2008

Ich versichere:

- dass ich die Diplomarbeit selbstständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfe bedient habe.
- dass ich diese Diplomarbeit bisher weder im In- noch im Ausland (einer Beurteilung bzw. einem Beurteiler zur Begutachtung) in irgendeiner Form als Prüfungsarbeit vorgelegt habe.
- dass diese Arbeit mit der vom Begutachter beurteilten Arbeit übereinstimmt.

Wien, im November 2008 Adnan Muslimovic

Abstract

Modern collaborations in science are very often based on large scale linking of databases that were not expected to be used together when they were originally developed. Within the distributed database community, database integration approaches traditionally focus on structural heterogeneity. However, in many scientific applications, there is additionally a strong demand to solve problems of semantic heterogeneity.

The heterogeneous and distributed mix of various data sources nowadays requires intelligent management systems in order to provide an unified view over such a data. The research challenge motivating the work on this Thesis is faced by the vision of dataspaces which main idea is to abstract from the underlying data source structures by providing a system managing various and heterogeneous data as single information data source. The dataspace concepts are presented as a vision, however their implementation in e-Science application environments opens new research challenges, especially, in distributed dynamic environments, like scientific grids.

The main effort of this work is to provide an integrated view over data being collected in scientific collaborations through e-Science life cycles. These life cycles represent a process of collecting data for significant analysis by introducing a hierarchical and iterative model, which includes several different activities. Each activity contains a number of tasks gathering information from multiple heterogeneous data resources that are organized as participants in scientific dataspaces. The e-Science Life Cycle Dataspace model is presented as ontology specified in the Web Ontology Language, which allows building semantically rich relationships among e-Science life cycle iterations and its participating data elements.

To my parents.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Scientific data are being collected to a great extent in various research domains. They are stored on multiple national sites in various databases. Scientific collaborations are targeting to provide access to these *primary data* by the means of e-Science applications. Through portals scientists are able to undertake these data for significant analyses in the context of their interest. The output of these analyses aims at defining a large number of predictions and might provoke further experimentation, which in turn may take days or weeks, depending on computational and human resources available. However, the resulting data – called *derived data* – that have arisen from the research task represents valuable information not only to the acting research group, but also to other groups with respect to other research areas.
Main objective is to link those derived data with their corresponding primary data by providing semantically rich relationships. Further, to make both relationships and data available within a space of data for people from various groups of organizations who might have use of it and who want to collaborate by the means of virtual organizations in the context of e-Science.
The success of such a dataspace will be highly dependent on the power of the used relationship concept as well as its flexibility. Rich relationships between the participants are going to be the backbone of such a system, with the basic necessity to support semi-automatically creation of them as well as their improvements and maintenance.
The development of a suitable relationship model customizable towards various application needs is therefore an important issue, to be challenged by the *e-Science life cycle data model*, which proposes a hierarchical and iterative metamodel providing a life cycle view of scientific data showing what ideally should happen to

data in e-Science environments. The e-Science life cycle model is implemented as a ontology, whose major role is to describe and semantically enrich the existing relationship among primary and derived data sets in e-Science applications. This is the basis for elaboration of intelligent and more powerful paradigms for the creation, representation and advanced searching of relationships among participants of a dataspace.

1.2 Objectives and Methods

The main objective of this work is to design and implement an intelligent relationship model for scientific data sources using OWL Web Ontology Language. This model is proposed as the e-Science Life Cycle Ontology and its core idea is to support scientific research work by providing an unified view over primary and derived data used in scientific environments by the means of an e-Science application which enables scientific collaborations and sharing of scientific knowledge.
This profound knowledge about scientific e-Science experiments, consolidated within instances of the ontology will highly contribute to the development of high productivity e-Science frameworks.

1.3 Organization of the Thesis

The outcome of this work - Design and Implementation of a Dataspace Model for e-Science Applications - is described in this document and organized as following. The beginning Chapter 1 of the Thesis introduces the most basic paradigms and concepts with regard to e-Science applications and Dataspace paradigms providing an insight into defined goals and achieved results. Chapter 2 describes the basic concepts of Semantic Web while regarding the most important ontology languages used for knowledge representation. In Section 2.6 the basic SPARQL Protocol and RDF Query Language concepts are illustrated.
Chapter 3 deals with the most important key dataspace concepts introduced in Section 3.2 and their realization by the data management community including following sections: In Section 3.3 the personal Dataspace management project *iMeMex* is described. Section 3.4 describes the Storage Resource Broker and in Section 3.5 its extended system called *iRODS* is described. In Section 3.6 we describe features and functionality of IBM WebSphere Information Integrator. Sections 3.7 and 3.8 cover data management systems of smaller scale. In Section 3.11 the related work is discussed in comparison with e-Science life cycle data model, and Section 3.12

provides an overview of all described systems by the means of an comparison matrix regarding Dataspace relevant paradigms.
Chapter 4 describes the e-Science Life Cycle data model and focuses on a ontology based data management solution for e-Science applications. Chapter 5 deals with the main e-Science Life Cycle Ontology concepts. We discuss the implementation issues and present the the main methodologies for ontology creation. The e-Science Life Cycle Ontology is evaluated in Chapter 6 using SPARQL query language. And finally, the Thesis is concluded in Chapter 7.

1.4 Results

With the help of the e-Science life cycle ontology, it is made possible for scientists to describe, execute and share their e-Science experiments with other researchers in an efficient manner. While searching the dataspace for performed scientific experiments in the form of life cycle iterations, a scientist can explore relationships among published results and generated data sets answering specific questions.
We present an ontology based relationship model with strong regard on the key dataspace concept, providing intelligent creation, representation, and searching of semantically rich relationships among primary and derived data sets in e-Science applications.
With this in mind, it will be easier for research groups to engage collaboration, provide knowledge transfers within collaborations and among different research groups with respect to different research areas. In conclusion, the e-Science life cycle meta-model is likely to unify the process of publishing primary, derived, and background data sets as well as the their interconnection and make it easy for scientists to register, describe and execute new e-Science experiments and for users to find, explore and understand these applied experiments.

Chapter 2

Basics On Ontology Languages

2.1 Introduction

The term ontology originates from a subfield of philosophy, where it is associated with the nature or subjects of existence and gives this particular field of metaphysics its name. [Wik08b] The concern of this branch of metaphysics is the identification of all kinds of existing things and their descriptions. The things and specific objects that made up the world can be categorized into groups which are represented through abstract classes sharing common properties with each other. This can be seen as the main application of a typical ontology [Wik08a].

The definition of the term Ontology in context of computer science is according to T.R. Gruber from Standford University [AvH04]: "a description of the concepts and relationships that can exist for an agent or a community of agents.", which was later refined by R. Studer [Gru08]: "An ontology is an explicit and formal specification of a conceptualization."

The main commitment designing Ontologies is to share knowledge about formal concepts of domain models and the relationships among these concepts. In general, an ontology consists of a finite set of concepts or objects having characteristic properties and existing relationships among these objects. If we take a university as an ontology example we can identify some important concepts, which are staff members including faculty members, professor, assistant or administrative stuff, then students, lectures, and courses. All of these concepts or classes can be represented in a hierarchical way through the concepts of subtypes or subclasses featuring the properties or relationships which describe the way the classes are related to each other as illustrated in Figure 2.1.

For example a property lectures can describe one specific relation between the objects professor and course. Further, ontologies may have restrictions which for example define that only staff members may give a lecture on particular course.

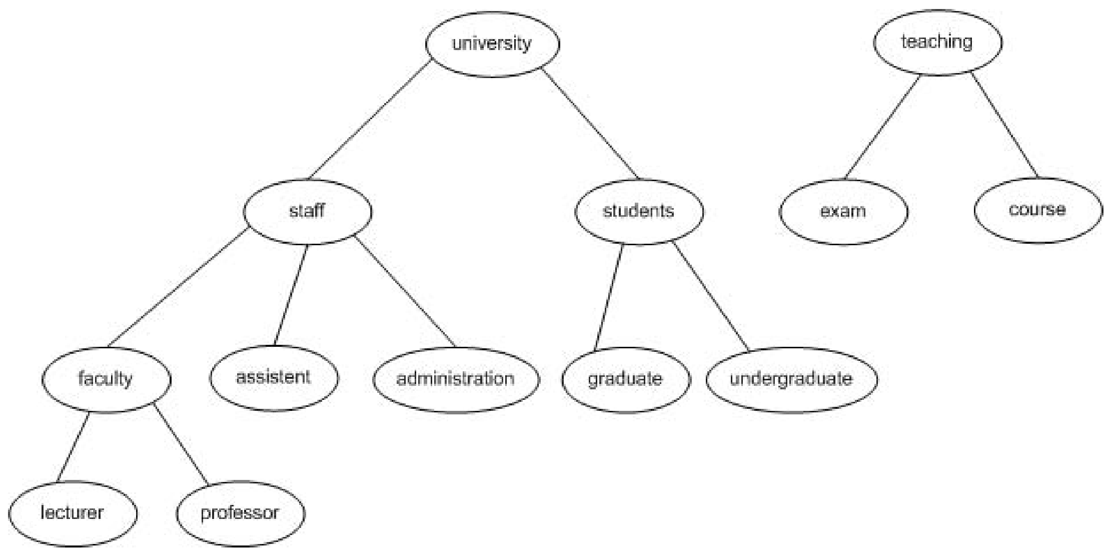


Figure 2.1: An ontology hierarchy

In the context of the Web, the ontologies are very useful for organization and navigation through the information of web sites providing a share of knowledge about specific domain models, which provides the main foundation for the Semantic Web [FHvH+01]. The most important formal languages used to built ontologies in the Web are the following:

- XML [BPSM+06]- provides basically a serialized syntax for structured documents allowing developers to use self defined extensions.
- XML Shema [FW04] - is a language for defining restrictions on the structure of XML documents.
- RDF [Bec04]- defines a simple data model for representing relations among objects providing very simple semantic information, which can be formalized and represented using XML sytax.
- RDF Schema [BG04] - introduces a description language for creating properties and classes for RDF objects. Further, it allows definition of ranges and domains for described properties.
- OWL [BvHH+04] - is a description language that has much richer vocabulary then RDF or XML. Specific relationships between classes (e.g., subSlassOf, disjointhWith), property restrictions (e.g., allValuesFrom), cardinality constraints, data types and annotations can be defined enriching semantic possibilities.

2.2 Semantic Web

The Semantic Web is provided by The World Wide Web Consortium (W3C), the main international standardization for the Web, and it is based upon a vision of knowledge exchange introduced by Tim Berners-Lee, the person who founded the WWW in the late 80s. The core idea of the Semantic Web is to structure the information in such a way as to provide a semantic approach to information and make it more understandable for machines [Web08].
Most of the today's web information, such as text, audio or video, is formalized in an unstructured way, which makes it difficult for machines to query and understand the semantics of information. Therefore it has become essential to create a framework, which defines semantic relationships among data and so allows more advanced knowledge management systems [Pal01].
Tim Berners-Lee developed and introduced a layered approach as shown in Figure 2.2 [BL00] to the Semantic Web and its languages. The idea is to establish a standardized approach for development of the Semantic Web [FHvH^{+}01].

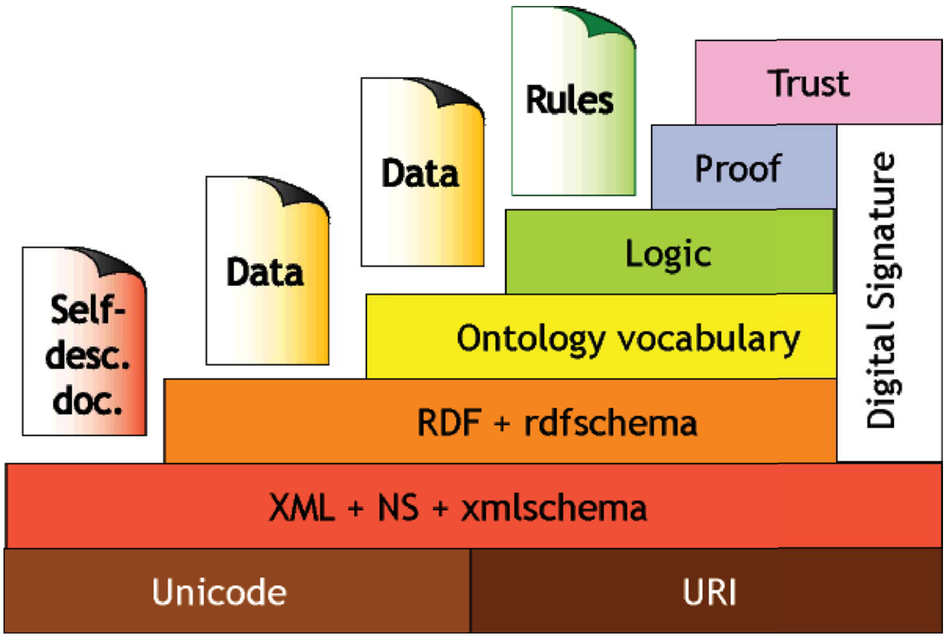


Figure 2.2: A layered approach of the Semantic Web [BL00]

XML, a language that allows writing structured Web documents, can be found at the bottom of the layered structure supporting user-defined extensions [Ame01]. The Syntax of RDF is based on XML and it is located above the XML layer. RDF provides a simple relationship data model for creating structured statements about Web resources. RDF Schema goes one step further and and builds a hierarchical model for describing the structure of this Web resources, their relationships, data types, and domain and range restrictions. RDF Schema can be seen as a simple language for writing ontologies. The ontology layer provides a way of defining more semantic relationships and is a extension of the RDF Schema. The logic layer uses a vocabulary of the ontology in order to expose implicit ontological language by making logical conclusions, which can be executed and proved by the Proof layer located above the logic layer. Finally, the trust layer, sitting on the top of the semantic web layered structure is dedicated to the quality and trust of the provided services and information by using certificated digital signatures [KM01].

2.3 XML

2.3.1 Introduction

XML stands for Extensible Markup Language, and allows developers to use self-defined extensions or tags in order to describe data objects forming structured XML documents understandable for machines [Wik08d]. XML is recommended by the W3C (World Wide Web Consortium), and is used for data exchange between applications on the web. It was developed as an extension of the Standard Generalized Markup Language (SGML). If we take a simple example of a Web page that contains some particular information on books represented as a XML document [BPSM+06], this information might look as follows:

```
<book>
    <title>Ontological Engineering</title>
    <author>A. Gomez-Perez</author>
    <author>M. Fernandez-Lopez</author>
    <author>O. Gorcho</author>
    <publisher>Springer</publisher>
    <year>2004</year>
    <ISBN>1852335513</ISBN>
</book>
```

We can see from this representation that the content is provided with semantic information about what role the tags play. For every opening tag there must exist a closing tag referring together to one element. The structure of the document defines the relation of the elements to each other.

2.3.2 Basic Concepts

An XML document may contain a prolog, consisting of an XML declaration and an optional reference to external structuring documents:

```
<?xml version="1.0" encoding="UTF-8"?>
```

The version and the information about which character encoding is used are defined. a reference to an external document may look as follows:

```
<!DOCTYPE person SYSTEM "person.dtd">
```

The local file called "person.dtd" contains the information about the structure of the document. The file might also be a reference to a URL. These, so called subsets, may be referenced using a SYSTEM and/or a PUBLIC label.
XML comments are defined as follows:

```
<!-- This is a comment. -->
```

XML documents are syntactically correct if they are well-formed [BHLT06]. This requires some syntax rules which are defined as follows:

- *"there is only one root element in the XML document"*

  ```
  <book> ... </book>
  ```

- *"each element must be enclosed between an opening and a corresponding closing tag".*
- *"each attribute name within an element must have an unique name."*

  ```
  <person name="Jimi" phone="659999"/>
  ```

- *"xml elements have to be properly nested, they may not overlap"* as seen in the example below, meaning that *"the tags have to be closed in the correct order."*

  ```
  <title>Ontological Engineering<author>O. Gorcho</title></author>
  ```

- *"all xml element and tag names has to be permissible"*

XML documents are valid if they are well-formed [BPSM+06], which implies that the document has to conform to semantic rules described above. The structure of XML documents can be defined in two ways: XML Schema, which offers constraints extensions on the document and mainly data type definitions, and DTDs, allowing a number of restrictions on the document.

DTD (Document Type Declaration) components can be defined separately in an external DTD file or included in the XML document itself. An external definition allows a usage of the same DTD file across several XML documents, and is therefor more efficient. Let's look at a simple DTD Example desribing some attributes of university professor shown below:

```
<!ELEMENT staff (professor*)>
<!ELEMENT professor (name, phone, institute, researchArea, gender?)>
<!ELEMENT name (\# PCDATA)>
<!ELEMENT phone (\# PCDATA)>
<!ELEMENT institute (\# PCDATA)>
<!ELEMENT researchArea (\# PCDATA)>
<!ELEMENT gender (\# PCDATA)>
<!ATTLIST person}
   id ID  \# REQUIRED
   address  CDATA  \# REQUIRED
   projectLeader IDREF \# IMPLIED
```

This DTD defines which element types can be used in an XML document. The elements: *name*, *phone*, *institute*, and *researchArea* belong to the element *professor* and have to be specified in this particular order. The element *professor* is an instance of the element *staff*, and may contain any number of *professor* elements which is denoted using *** parameter. Using the parameter *?* we can indicate that the specification of element *gender* is optional. # PCDATA (parsed character data) is the only element data type and may contain any content.

Additionally to the definition of elements, a attribute list can be defined including: attribute name, attribute type and attribute values. *ID* is associated with an unique name which can be used across the entire XML document. *CDATA* indicates a string, a sequence of characters. *IDREF* references to another element with the same value having the same ID attribute as the *IDREF*. These attribute types are the most important ones.
The value types give the information if the attribute is *# REQUIRED* within an

element or its appearance is *# IMPLIED*, meaning optional.

XML Schema
XML Schema [TBMM04] provides much richer language then DTD, and is based on XML syntax. It allows reuse and refine of already created schemas and types while restricting or extending already available types and schemas. In comparison to DTD, which only have string as data types [BPM04], XML schema offers a set of data types that can be used for XML documents. Here is a list of some of the most important *bulit-in data types*:

- numeric, such as *integer, byte, long float, decimal, short*
- string, such as *string, ID, IDREF, CDATA, Langugae*
- data types for date and time, such as *time, Date , Month, Year*

2.4 RDF

2.4.1 Introduction

As we have already seen XML is mostly used as a language for data exchange among applications where the application itself is concerned about how to interpret the semantics of the tags used in an XML documents. The RDF (Resource Description Framework) is basically an abstract data model based on XML syntax that is able to define statements about resources in the form of so called triple expressions [MM04]. These triples consists of an object, a predicate, and a subject. For example if we would use RDF triples to represent the statement saying that *professor teaches courses*, *professor* would represent a subject, *teaches* a predicate, and *courses* would indicate an object.
The idea of RDF language in the first place was to represent meta data about resources in the WWW such as web page meta information about the author, title, date of creation, and so on. Each RDF resource or object can have different values describing the relation between the resources. These relationships, also called properties, are described in a mechanism called RDF Schema defining which vocabulary is used for RDF data model.
RDF identifies the resources (objects) and the properties of these objects using URIs which can be represented as nodes in a graph structure for better understanding. In order to make RDF documents accessible and processable for machines RDF

uses XML syntax provided by the tag *rdf:RDF* [Bec04]. These XML elements that represent RDF documents contain certain descriptions *rdf:Description* on resources.

2.4.2 Basic Concepts

As already mentioned RDF resources use URIs (URI references) to identify simple statements, called RFD triples. Let's say we have following statements:
http://www.RDFexample.ac.at/index.html has a an owner whose name is Peter Mayer and *http://www.RDFexample.ac.at/index.html has a date of creation whose value is September 17, 2008* illustrated as an RDF graph in the figure below:

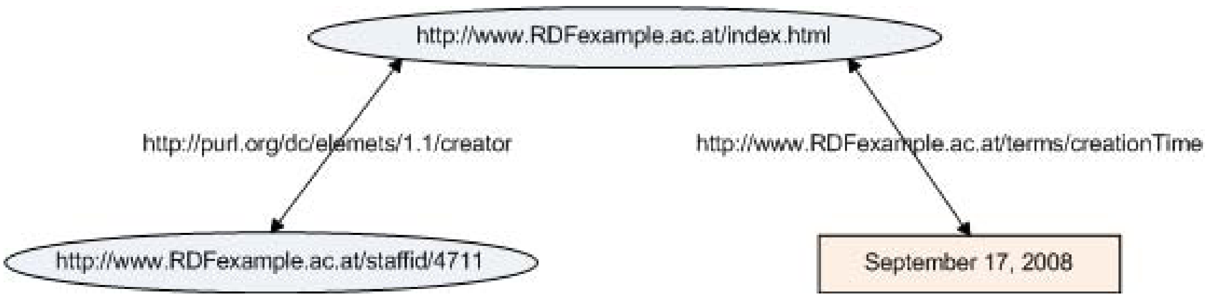

http://www.RDFexample.ac.at/index.html
http://purl.org/dc/elemets/1.1/creator
http://www.RDFexample.ac.at/terms/creationTime
http://www.RDFexample.ac.at/staffid/4711
September 17, 2008

Figure 2.3: A simple RDF graph

We have a following RDF triple statement including:

- a subject - *http://www.RDFexample.ac.at/index.html* that is represented as an URI reference and illustrated as a node of the RDF graph model
- a predicate - *http://purl.org/dc/elemets/1.1/creator* which also uses an URI as reference and an arc in the RDF model representation.
  http://purl.org/dc/elemets/1.1/ contains a Dublin Core Metadata vocabulary that provides a set of defined property elements describing the objects, which was initially developed at the the Metadata Workshop in Dublin, in March 1995.
- a object - *http://www.RDFexample.ac.at/staffid/4711*, represented as an URI reference and a node in the RDF graph model.

The URI reference *http://www.RDFexample.ac.at/terms/creationTime* uses a string to represent a property value. We see from this that RDF objects (references) may contain either URIs (illustrated as ellipses) or constant character strings (illustrated as boxes in the RDF graph), also called literas, as property values. Because URI references can result in long lines, a QName (XML qualified name) can be specified containing a prefix followed by the namespace URI, which might look as follows:

prefix *RDFex:*, namespace URI: *http://www.RDFexample.ac.at/*
prefix *terms*: namespace URI: *http://www.RDFexample.ac.at/terms*
prefix *staff*: namespace URI: *http://www.RDFexample.ac.at/staff*
prefix *dc:*, namespace URI: *http://purl.org/dc/elements/1.1/*

Using the newly created prefixes *RDFex*, *terms*, *staff*, and *dc*, we could also write the above discussed statements in the following triple (subject, predicate, object) notation.

RDFex:index.html - dc:creator - staff:4711
RDFex:index.html - terms:creationTime "September 17, 2008"

Adding further personal information to our RDF graph would result in following graph structure.

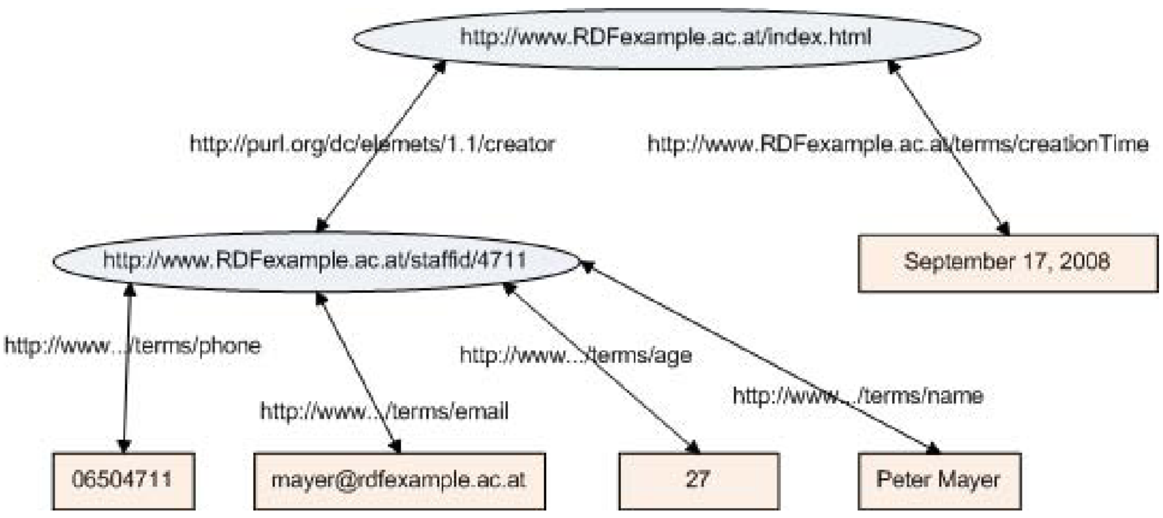

http://www.RDFexample.ac.at/index.html
http://purl.org/dc/elemets/1.1/creator
http://www.RDFexample.ac.at/terms/creationTime
http://www.RDFexample.ac.at/staffid/4711
September 17, 2008
http://www..../terms/phone
http://www..../terms/age
http://www..../terms/name
http://www..../terms/email
06504711
mayer@rdfexample.ac.at
27
Peter Mayer

Figure 2.4: An RDF graph with some personal information

As mentioned before, RDF documents use XML sytax [Bec04] for writing machine accessible and processable RDF models formalized in RDF/XML spezification. A concrete RDF/XML implementation of the RDF graph illustrated in Figure 2.4 might look as shown in the example below.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns\#"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:terms="http://RDFexample.ac.at/staffid/">
   <rdf:Description rdf:about="
      http://www.RDFexample.ac.at/index.html">
      <terms:creationTime>September 17, 2008</terms:creationTime>
      dc:creator rdf:resource="
      http://www.RDFexample.ac.at/staffid/4711"/>
   </rdf:Description>
   </rdf:Description rdf:resource="
      http://www.RDFexample.ac.at/staffid/4711"/>
      <terms:name>Peter Mayer</terms:name>
      <terms:phone>06504711</terms:phone>
      <terms:email>mayer@RDFexample.ac.at</terms:email>
      <terms:age>27</terms:age>
      </rdf:Description>
</rdf:RDF>
```

The example begins with the XML declaration specifying the *xml version.* Furter, an *rdf:RDF* element is defined followed by the XML namespaces specifying the prefixes for URI identification. The use of *rdf:Description* tag describes the RDF statement and additionally defines the resource by using the tag *about.* Finally, property values and the corresponding URI resources are defined.
RDF Schema, also called RDF vocabulary, describes a set of classes and properties that are used to specify the statements about the recourses, and the their relationships. The prefix *rdfs* identifies the namespace of the RDF Schema [BG04] and it is specified by the URI reference *http://www.w3.org/2000/01/rdf-schema#.* Here are some of the most important classes and properties defined in the RDF Vocabulary Description Langugae.

- rdfs:class - classes can be seen as a set of elements representing the recourses, having individuals (instances) that belong to these classes. resources can be declared as instances of classes called *rdfs:class.*

- rdfs:resource - all classes defined in RDF are derived from the class rdfs:resource. Every object described in RDF is a resource and represents an instance of this class.

- rdfs:property - describes the relation between the defined classes.

- rdfs:subPropertyOf - used to indicate the relation of properties to each other, one property being an instance of another.

- rdfs:subClassOf - used to indicate the relation of classes to each other, one class representing an instance of another.
- rdfs:domain - a property can have a domain specification, meaning a restriction to classes of resources defining which resource can be defined as a subject.
- rdfs:range - a property can have a range specification, meaning a restriction to classes of resources defining which resource can be defined as an object.

2.5 The OWL Language

2.5.1 Intoduction

As we have seen in the previous chapter RDF an RDF Schema provide a formal model for describing the structure of Web documents. The semantic expressiveness of XML documents provided by the RDF schema is mostly limited to a class and property hierarchy with the ability to define domains and ranges of these properties. Some important features for representing the ontological knowledge are missing in the RDF Schema:

- The range property defined in RDF Schema cannot be restricted just to some particular classes. We cannot specify that for example only professor can be a project leader, while assistants and students are project members.
- In RDF Schema we can only declare a subclass relationship. For example, a paper is a subclass of publication, but not that for example a class article is disjoint from a class paper.
- Cardinality restrictions, such as a restriction specifying that a course is taught by at least one professor cannot be expressed using RDF Schema.

Therefor a need for more expressive language for Semantic Web that is processable and understandable by applications has led to a development of the OWL Web Ontology Language.

2.5.2 The species of OWL

W3C Web Ontology Working Group defines tree different OWL sublanguages, which are OWL Lite, OWL DL, and OWL Full. Each of these sublanguages provide certain level of expressiveness and reasoning support for different user requirements, OWL Full being the most expressive one, followed by OWl Dl and OWl Lite [Wik08c].

OWl Lite

OWL Lite has the simplest syntax and it supports a primitive classification hierarchy and some basic constraints. Cardinality constraints defined in OWL Lite are restricted to cardinality values of 0 and 1. The idea of OWL Lite is to provide an easy implementation by providing only a subset of constructs available in OWL language. It is a sublanguage of OWL DL and [MvH04] *"every legal OWL Lite Ontology is also a legal OWL DL Ontology."*

OWl DL

OWL DL (Description Logic) is an extension of OWL Lite and provides restrictions about how OWL and RDF constructors are used, including all constructs of OWL language. For example, one class itself cannot be an instance of another class, while it may be a subclass of different classes. Classes cannot be defined as properties or individuals at the same time neither the properties can be individuals. OWL DL offers efficient reasoning and complete computational features supported by Description Logics perfoming consistency checks. There is a certain lack of compatibility with RDF, meaning that RDF documents mostly need some extensions or restrictions to be legal OWL DL documents, whereas [AvH04] *"every legal OWL DL Document is a legal RDF document."*

OWl Full

OWL Full offers the most expressiveness and is an extension of OWL DL using all primitives of the OWL language with the disadvantage of no computational guaranties. The language has become very powerfull and complex so that no efficient reasoning can be supported. OWL Full allows combination of all primitives (classes, properties, individuals) in different ways with RDF and RDF Schema. A class declared in OWL Full can for example simultaneously be seen as a collection of individuals and as an individual itself. The syntax and semantics of OWL Full are completely compatible with RDF, implying that [AvH04] *"a legal RDF document is also a legal OWL document and any valid RDF Schema conclusion is also a valid OWL Conclusion"*.

Which OWL sublangugae ontology developers will choose depends on construct expressiveness provided by the sublanguages, and user requirements in order to suit their needs. OWL Full will not be able to offer a complete reasoning features, but it may be more powerful if mixing the primitives with RDF Schema facilities in comparison with OWL DL language, which allows more effective reasoning support. There is a strict compatibility distinction between OWL Full, OWL DL, and OWL Lite [MvH04]:

"Every legal OWL Lite ontology is a legal OWL DL ontology."
"Every legal OWL DL ontology is a legal OWL Full ontology."
"Every valid OWL Lite conclusion is a valid OWL DL conclusion."
"Every valid OWL DL conclusion is a valid OWL Full conclusion."

Owl language is based on RDF syntax, whereas instance declaration and RDF descriptions including typing information is the same as in RDF. Some of the most important OWL construct facilities, such as *owl:class*, *owl:ObjectProperty*, and *owl:DatatypeProperty* are derived from *rdfs:resource* and *rdfs:class* constructs.

2.5.3 Header

An OWL document begins usually with a declaration of namespaces. These RDF documents are also called *OWL ontologies* using the root element *rdf:RDF* as an opening tag. Here we declare a number of namespaces:

```
<rdf:RDF
   xmlns:owl="http://www.w3.org/2002/07/owl\#"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns\#"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema\#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema\#"
```

Everything starting with the prefix *owl* uses the vocabulary of OWL, and is related to the namespace reference *"http://www.w3.org/2002/07/owl#"*. The other prefixes rdf (RDF syntax), xsd (XML Schema), and rdfs (RDF Schema) also refer to their namespace declaration references each providing the needed vocabulary.
DOCTYPE, a document type declaration, uses entities to declare ontology definitions in order to avoid writing long URLs as illustrated bellow.

```
<!DOCTYPE rdf:RDF [
   <!ENTITY prof  "http://gridminer.org/university/professor\#">
   <!ENTITY institute "http://gridminer.org/university/institute\#">]
```

The declaration of the Ontology definition consists of the ENTITY attribute, the abbreviations, and the long URL reference for which the newly defined abbreviation can be used.
In the OWL document we can for example use the entity declaration *"&prof;Fischer"* as abbreviation for *"http://gridminer.org/university/professor#Fischer"*. Additionally a set of assertions can be included, provided by the *owl:Ontology* element, which may include other ontologies, version controls and comments:

```
<owl:Ontology rdf:about="">
  <rdfs:comment>A collection of assertions example</rdfs:comment>
  <owl:priorVersion
  rdf:resource="http://gridminer.org/university/institute"/>
  <owl:imports
  rdf:resource="http://gridminer.org/university/professor"/>
  <rdfs:label>Institute Ontology</rdfs:label>
</owl:Ontology>
```

When importing other ontologies we have to consider that for each used namespace there will be an import element providing all the assertions of included ontology. Note that *owl:import* is a transitive property [AvH04]: *"if ontology A imports ontology B, and ontology B imports ontology C, then ontology A also imports ontology C."*

2.5.4 Classes

Classes are the most basic abstraction concept in OWL used to group similar resources together. Every OWL class provides a set of instances extending the class with so called individuals [BvHH+04]. We define OWL classes using the owl:class Element in conjunction with the attribute rdf:ID, which is used to specify the name of the class:

```
  <owl:Class rdf:ID="institute"/>
  <owl:Class rdf:ID="professor"/>
```

Every OWL class, defined by a user, is a subclass of a predefined class *owl:Thing* and therefor each individual is member of this class. Another predefined class *owl:nothing* is defined as *the empty class*, thus [reference]*every class is a superclass of owl:nothing.* The newly created classes *institute* and *professor* can be referenced within an OWL documnets using a # identifier followed by the name of the class: *#professor* and *#institute.* These classes can also be referenced by other ontologies using the URI reference of the ontology and the name of the class, which might look as follows:

```
http://gridminer.org/university/instituteOntology.owl\#professor
```

The more specific construct to group similar resources together is the element *rdfs:subClassOf*. *If A is a subclass of B, then every instance of A is also an instance of B* [BvHH+04]. The *rdfs:subClassOf* is also transitive [BvHH+04]: *If X is a subclass of Y and Y a subclass of Z then X is a subclass of Z.* Here is a simple example of a subclass construct in OWL:

```
<owl:Class rdf:ID="lecturer">
  <rdfs:subClassOf rdf:resource="\#professor" />
</owl:Class>
```

The statement *subClassOf* creates a necessary relation meaning that an individual also needs to be a *lecturer* in order to be a *professor*.
Using the *owl:equivalentClass* element we can define that classes are equivalent linking them together. Classes declared es equivalent contain the same set of individuals:

```
<owl:Class rdf:about="staff">
  <equivalentClass rdf:resource="\#faculty"/>
</owl:Class>
```

We can also say that a class is disjoint from one or more other classes using the owl:disjointWith statement, which indicates that these classes do not share any individuals.

```
<owl:Class rdf:about="\#paper">
  <owl:disjointWith rdf:resource="\#thesis"/>
  <owl:disjointWith rdf:resource="\#book"/>
</owl:Class>
```

2.5.5 Properties

In OWl we distinguish between two types [SWM04] of properties.

- Object property - describes the relation between objects or instances of classes. An example for object property would be a *isLeaderOf* property indicating that an instance of a class *professor isLeaderOf* an instance of a class *project*

- Data type Property - describes the relation between objects and data type values. Examples of data type properties are name, phone, and email etc. OWL does not provide any predefined data types, but it allows XML Schema data types.

A datatype property declaration may look as illustrated below:

```
<owl:DatatypeProperty rdf:ID="publicationDate">
   <rdfs:domain rdf:resource="\#paper"/>
   <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema\#date"/>
</owl:DatatypeProperty>
```

In this example we define a datatype property *publicationDate* which describes a relation between an individual and a XML Schema datatype, in this case *date*. An example of a object property may look as follows:

```
<owl:ObjectProperty rdf:ID="isLeaderOf">
   <rdfs:domain rdf:resource="\#professor"/>
   <rdfs:range  rdf:resource="\#project"/>
</owl:ObjectProperty>
```

A declaration of a object or data type property alone, does not imply which individuals are related to each other. By the use of *range* and *domain* statements we can define which classes are related to each other. A restriction domain indicates that the subject of such declared property has to belong to related instance of a class, which means that the property domain of *isLeaderOf* is restricted to individuals of the class *professor*. A *range* statement indicates that the objects of the property range has to belong to a instance of defined class, which means that the property range is restricted to individuals of the class *project*.

Owl also allows as to define "inverse properties", which interchanges a direction of a range and domain relation. *professor* (declared as domain axiom) *isLeaderOf* a *project* (declared as range axiom) can be specified as an inverse property relation saying that a *project* (declared as domain axiom) *hasProjectLeader professor* (declared as range axiom):

```
<owl:ObjectProperty rdf:ID="hasProjectLeader">
  <rdfs:domain rdf:resource="\#project" />
  <rdfs:range  rdf:resource="\#professor" />
  <owl:inverseOf rdf:resource="\#isLeaderOf">
</owl:ObjectProperty>
```

In OWL we can define equivalence of properties through the use of *owl:equivalentProperty* construct. The next example shows two equivalent properties, which have the same domain and range values.

```
 <owl:ObjectProperty rdf:ID="hasAdvisor">
    <owl:equivalentProperty rdf:resource="\#hasProjectLeader">
 </owl:ObjectProperty>
```

2.5.6 Property Restrictions

Property restrictions in OWL allows us to describe certain conditions about instances of a particular class, which means to put constraints on ranges of properties. In OWL we distinguish between value constraints and cardinality constraints.

Value Constraints
Suppose we would like to specify that *thesis* requires to be approved by *professors* only. An example illustrating that restriction might look as follows.

```
<owl:Class rdf:about="\#thesis>
 <rdfs:subClassOf>
   <owl:Restriction>
     <owl:onProperty rdf:resource="\#isApprovedBy"/>
     <owl:allValuesFrom rdf:resource="\#professor" />
   </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
```

The *owl:allValuesFrom* statement specifies the values of the class that can be members of the property declared using *owl:onProperty*, which means that the property *isApprovedBy* can only take values of the class *professor*. The fact that *owl:Restriction* is suited within the *rdfs:subClassOf* statement illustrates that *owl:Restriction* is a subclass of *owl:class*, an anonymous OWL class, used to describe a class [BvHH$^+$04]. By using *owl:someValuesFrom*statement, we could also define that at least one value of the specified property has to reference to a individual of a particular class. We can for example specify that an university assistant has at least one research area:

```
<owl:Class rdf:about="\#assistant>
 <rdfs:subClassOf>
   <owl:Restriction>
     <owl:onProperty rdf:resource="\#researches"/>
     <owl:someValuesFrom rdf:resource="\#reasearchArea"/>
   </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class> \\
```

With *owl:hasValue* we can specify which set of individuals depending on the property values are members of a specified class. We can for example define that thesis are written in *english*, wheras *english* is representing an individual. At least one value *english* of the property *isWrittenIn* is representing a member of the class *thesis*:

```
<owl:Class rdf:about="\#thesis>
 <rdfs:subClassOf>
   <owl:Restriction>
     <owl:onProperty rdf:resource="\#isWrittenIn"/>
     <owl:hasValue rdf:resource="\#english"/>
   </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
```

Cardinality Constraints

The constraint owl:minCardinality specifies that all individuals of a class must at least have a certain number of individuals specified in a property value. In the following example we say that a research area is researched by at least one individual.

```
<owl:Class rdf:about="\#researchArea>
 <rdfs:subClassOf>
   <owl:Restriction>
     <owl:onProperty rdf:resource="\#isResearchedBv"/>
     <owl:minCardinality rdf:datatype="\&xsd;nonNegativeInteger">1
     </owl:minCardinality>
   </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
```

The data type *nonNegativeInteger* used to specify a property value is referred to XML Schema datatypes by using the *xsd* namespace. Additionally, we can also define a *owl:maxCardinality* value, which specifies the number of property values individuals of a class may contain at most. Let's assume that, for purpose of this example, a research group may have at most twenty researchers:

```
<owl:Class rdf:about="\#researchGroup>
 <rdfs:subClassOf>
   <owl:Restriction>
     <owl:onProperty rdf:resource="\#hasResearcher"/>
     <owl:maxCardinality rdf:datatype="\&xsd;nonNegativeInteger">20
     </owl:maxCardinality>
   </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
```

2.5.7 Property Characteristics

There are some characteristic properties in OWL, such as *owl:TransitiveProperty*, *owl:Inv- erseFunctionalProperty*, *owl:FunctionalProperty*, and *owl:SymmetricProperty*, which can be specified directly.
owl:FunctionalProperty is declared as a subclass of *rdf:property* class and can be applied on both datatype and object properties. This property definition may contain one value for each object at most. In the following example we specify that a postgraduate student has at most one professor as supervisor.

```
 <owl:ObjectProperty rdf:ID="hasSupervisor">
   <rdf:type    rdf:resource="\&owl;FunctionalProperty"/>
   <rdfs:domain rdf:resource="\#postgraduateStudent"/>
   <rdfs:range  rdf:resource="\#professor"/>
 </owl:ObjectProperty>
```

owl:InverseFunctionalProperty is declared as a subclass of owl:ObjectProperty class and can not be applied on datatype properties in OWL DL. This property definition specifies that different instances can not contain the same values. We might specify that a person has one country of origin only.

```
 <owl:InverseFunctionalProperty rdf:ID="coutryOfOrigin">
   <rdfs:domain rdf:resource="\#person"/>
   <rdfs:range rdf:resource="\#coutry"/>
 </owl:InverseFunctionalProperty>
```

owl:TransitiveProperty is declared as a subclass of *owl:ObjectProperty* class and it provides a property definition which specifies a transitive relation between objects. Let's assume that if *groupX*, *groupY* and *groupZ* are research groups, and *groupZ* is a subgroup of *groupY*, and *groupY* ist subgroup of *groupX*, then the OWL reasoner is able to resolve that *groupZ* is also a subgroup of *groupX*, which is illustated in the next example:

```
<owl:TransitiveProperty rdf:ID="subReasearchgroupOf">
  <rdfs:domain rdf:resource="\#researchGroup"/>
  <rdfs:range rdf:resource="\#researchGroup"/>
</owl:InverseFunctionalProperty>
```

owl:SymmetricProperty is also declared as a subclass of *owl:ObjectProperty* class and it is defined as follows [BvHH+04]:*"if the pair (x,y) is an instance of P, then the pair (y,x) is also an instance of P*. We might specify a symmetric property as shown below. Domain and range properties have the same value:

```
<owl:SymmetricProperty rdf:ID="InCollaborationWith">
  <rdfs:domain rdf:resource="\#researchGroup"/>
  <rdfs:range rdf:resource="\#researchGroup"/>
</owl:InverseFunctionalProperty>
```

2.5.8 Class Operators

OWL allows a use of union, intersection, and complement class operators that represent the logic AND, OR, and NOT operators.
intersectionOf is used to create a intersection of instances of two classes. Let's assume we specify all the individuals of the class *professor* that are members of a *researchGroup* and intersect this class with *reaserchGroupAdvisor* to give a advisor of a research group:

```
<owl:Class rdf:ID="researchGroupAdvisor">
 <owl:intersectionOf rdf:parseType="Collection">
 <owl:Class rdf:about="\#professor"/>
  <owl:Restriction>
    <owl:onProperty rdf:resource="\#isMemberOf"/>
    <owl:hasValue rdf:resource="\#researchGroup"/>
  </owl:Restriction>
 </owl:intersectionOf>
</owl:Class>
```

owl:unionOf is used to create a union of instances of two classes. Suppose, we define a new class linking the individuals of two classes together:

```
<owl:Class rdf:ID="student">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="\#postgraduateStudent"/>
    <owl:Class rdf:about="\#undergraduateStudent"/>
  </owl:unionOf>
</owl:Class>
```

owl:complementOf is used to specify a set of individuals that are complement of individuals of an other class. The class *postgraduateStudent* contain all the instances that are not members of the class *undergraduateStudent*

```
<owl:Class rdf:ID="postgraduateStudent">
 <owl:complementOf>
   <owl:Class rdf:about="\#undergraduateStudent"/>
 </owl: complementOf>
</owl:Class>
```

2.5.9 Enumerations

An enumeration statement is defined using an *owl:oneOf* property element. It lists a set of individuals of a defined class containing exactly these instances of a class. The representation uses the *rdf:parseType="Collection"* construct to list the individuals:

```
<owl:Class rdf:ID="season">
 <owl:oneOf rdf:parseType="Collection">
   <season rdf:about="\#spring"/>
   <season rdf:about="\#summer"/>
   <season rdf:about="\#autumn"/>
   <season rdf:about="\#winter"/>
 </owl:oneOf>
</owl:Class>
```

2.5.10 Individuals

Individuals are instances (members) of classes and in OWL they are defined using a RDF statement:

```
<professor rdf:ID="4711"/>
```

We can also use the *rdf:type* property to declare an equivalent instance as illustrated above:

```
 <owl:Thing rdf:about="\#4711">
   <rdf:type rdf:resource="\#professor"/>
 </owl:Thing>
```

Because of the fact that same or different individuals can be referenced in various ways while having different names or notations, OWL offers tree constructs for specifying individual identity: *owl:sameAs*, *owl:differentFrom*, and *owl:AllDifferent*. *owl:sameAs* links two individuals together stating that these two individuals are identical while having the same URI reference. Such statements can be applied for ontology mappings when different notations refer to same individual. An example might look as follows:

```
 <course rdf:ID="pid1147">
   <owl:sameAs rdf:resource="\#pnr1147"/>
 </course>\\
```

owl:differentFrom indicated that two individuals are not identical while having different URI references. Here is a simple example to illustrate this statement:

```
 <professor rdf:ID="4711">
   <owl:differntFrom rdf:resource="1147"/>
 </course>
```

For a large number of different individuals OWL provides a *owl:AllDifferent* statement. Therefore OWL defines a special property *owl:distinctMembers* in order to establish links among different individuals:

```
<owl:AllDifferent>
 <owl:distinctMembers rdf:parseType="Collection">
   <professor rdf:about="\#4711">
   <professor rdf:about="\#1147">
   <professor rdf:about="\#4117">
 </owl:distinctMembers>
</owl:AllDifferent>
```

2.6 SPARQL

2.6.1 Introduction

SPARQL stands for *SPARQL Protocol and RDF Query Language* and it is developed by the *RDF (DAWG) Data Access Working Group.* It introduces a RDF query language, which is officially recommended by the W3C (World Wide Web Consortium). SPARQL query language can be used for any data on which RDF mappings can be applied and it provides tree different specifications:

- The query language specification [PS08] - specifies the semantic and syntax definitions of the language.
- SPARQL Query Results XML Format [BB08] - it provides a result representation format for SPARQL queries, which is described in XML.
- SPARQL Data Access Protocol for RDF [CFT08] - defines simple protocols in order to transfer SPARQL queries to a processor implementing a query service.

There are several implementation of SPARQL language facilities. Here is a list of some of the most important tools.

- *ARQ* - a JENA query engine
- *Pellet* - is an OWL DL reasoner, implemented in JAVA, that has SPARQL query support.
- *Rasqal* - is a RDF query library, which is open source and can handle SPARQL und RDQL gueries.
- *RDF::Query* - is a Perl implementation, which also handles SPARQL und RDQL language gueries.
- *twinql* - is a Lisp implementation for SPARQL query language.

There are some important SPARQL language objectives and constructs that are still not available or under active development, which are: a support for modifications (update and insert facilities) on RDF data, a *COUNT* and *AVG* statements, and a use of a *select* statement within an other selection [nw08].

2.6.2 Basic Concepts

SPARQL builds upon the RDF triple statement construct, which consists of a subject, a predicate, and an object, whereas each triple statement can represent a variable [PS08]. The next example illustrates how to query the names of all professors from a professor ontology. Let's assume we have a professor ontology containing some basic personal information. A SPARQL query might look as follows:

```
PREFIX prof: <"http://gridminer.org/university/professor\#">
SELECT ?name
  FROM <http://gridminer.org/university/professor.owl>
  WHERE { ?professor prof:name ?name. }
```

First we declare a prefix *prof* which refers to certain URI, specifying a namespace. The newly created prefix can be used anywhere in the query as a abbreviation for the URI reference. The *select* statement consists the data variables that will be returned in the query result. The statement *from* provides an instance, represented as URI, of the data being queried. In the *where* clause we use a triple construct representing the graph pattern of the data.
The SPARQL results are represented in a table form, whereas every row represents one query answer. Each variable used in the select statement represents a column in the result table. Suppose, we have tree individuals, the query result may look as illustrated in the table below:

row	**Name**
1	Jimi
2	John
3	Scott

Table 2.1: SPARQL Results Table Form

In the following example we select the name, age and email of all individuals in the professor ontology. Additionally to prefix declaration specifying a namespace in the example above, we can use s *BASE* statement in order to provide one further abbreviation by defining *BASE ¡http://gridminer.org/university/professor/¿* and so avoid long URIs.

```
BASE <http://gridminer.org/university/professor/>
PREFIX prof: <professor\#>
SELECT ?name ?age ?email
FROM <professor.owl>
  WHERE {
        ?professor prof:name ?name.
        ?professor prof:age ?age.
        ?professor prof:email ?email.
         FILTER(?age > 30)
  }
```

As shown in the example above we can specify a *FILTER* to return only those individuals who are older than thirty. For each variable declared in the *select* statement we have to use one separate pattern establishing a graph pattern. Only those *professor* instances that match all tree variables will be returned as a result, otherwise the triple pattern is not conformed. The variables that are not specified in a *where* statement can not be selected and will not be returned in the result. The query processor has to attach every variable to an RDF triple graph pattern [Dod05].
Instead of listing the variables in the *select* statement we could also use the *** parameter to query all the variables while delegating the order of the columns to the query processor.
Suppose, we add a new variable *middleName* to our *select* clause. Because some *professors* may not have a middle name and we already know that all the variables has to be bound to an existing triple pattern in order to be returned, we will have to specify the new pattern as *optional*:

```
BASE <http://gridminer.org/university/professor/>
PREFIX prof: <professor\#>
SELECT ?name ?age ?email ?mName
FROM <professor.owl>
  WHERE {
        ?professor prof:name ?name.
        ?professor prof:age ?age.
        ?professor prof:email ?email.
        OPTIONAL {?professor prof:middleName ?mName.
  }
```

If we for example wish to query for all professors who lectures in German as well as in English, we could use an UNION expression to query for both languages. An example may look like this:

```
BASE <http://gridminer.org/university/professor/>
PREFIX prof: <professor\#>
SELECT ?name ?id
FROM <professor.owl>
   WHERE{
        ?professor prof:name ?name.
        ?professor prof:id ?id.
        ?professor prof:langugae prof:english.
   }UNION {
        ?professor prof:name ?name.
        ?professor prof:id ?id.
        ?professor prof:langugae prof:german.
   }
ORDER BY ?name
LIMIT 15
OFFSET 15
```

the returned result can be ordered by any selected variable using the ORDER BY clause as illustrated above. DESC and ASC (default) expressions can also be attached to the variable changing the order of the result:

```
ORDER BY DESC(?name)
```

In addition to specifying the order of the results we may wish to subdivide the returned results into pages by giving the number of displayed results. This can be achieved by using a OFFSET, and LIMIT expressions, as illustrated above.
As already mentioned SPARQL query results can be returned using the SPARQL Query Results XML Format to display the results as an XML or RDF format.

Chapter 3

Related Work

3.1 Introduction

This chapter aims in identifying what are the most important key dataspace paradigms and further how these are realized by the data management community. It also aims in identifying bottlenecks of proposed and implemented approached as well as in providing more appropriate solutions concerning some Dataspace paradigms. It has a strong impact on the e-Science Life Cycle Ontology future work, which is to develop key Dataspace paradigms within a Dataspace management system for large scale proposes.

3.2 Dataspaces

The idea of Dataspaces was firstly introduced by Franklin et al. [FHM05] in 2005. It is represented in a visionary way describing data management on a higher abstraction level. A dataspace consists of a set of participants and a set of relationships among participants [FHM05]. A participant can be any element containing data in some way. Relationships describe how two participants are related to each other. Relationships can be expressed by single word-relationships, such as replica-of, related-to, view-of, etc. In the extreme example they can be semantic mappings of database schemas. The challenge is to raise up the level at which data is managed [Hal05]. Systems providing the required services over dataspaces are considered to be Dataspace Support Platforms (DSSPs) [HFM06]. In [EBT06] such a system is defined as *"a set of software programs that controls the organization, storage and retrieval of data in a dataspace. It also handles the security and integrity of the dataspace."*

The most important components [FHM05] of a dataspace system are illustrated in the Figure below:

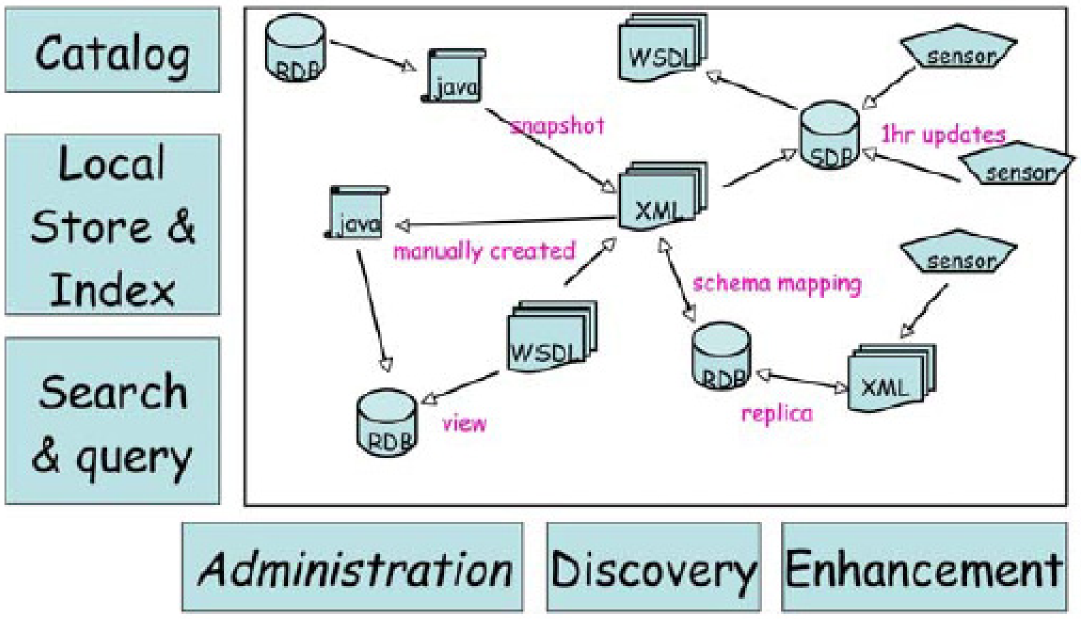


Figure 3.1: A dataspace system components [FHM05]

Catalog
The catalog provides the entire information about all the data sources acting as participants in a dataspace. It manages not only the metadata information about the participants, but also how they are related to each other. The catalog should offer a browsing feature allowing to explore the content of a dataspace. Additionally a relationship modeler should be implemented in order to allow a user to generate semantic relationships between participating data sources.

Search and Query
A dataspace system should be aware of query facilities with no regard to the underlying data structure of the dataspace participants. Simple keyword queries, as well as more advanced querying should be supported. All participant metadata managed by a Catalog could be included into query statement definitions in order to provide more sophisticated query results.

Local Store and Index
All participants should be indexed in order to provide more intelligent and efficient query answers. Further it should enable efficient data query results without actuality going into data resource itself.

Discovery
This component helps discover the participants and their relations stored in a dataspace. It keeps track of participant relationships management, monitoring the evolution of the participants and their relations to each other.

Administration
The administration component should offer a central management instance being responsible for authentication, access rights, permissions, and further management facilities regarding synchronization of the dataspace components.

3.3 *iMeMex* A Personal Dataspace Management System

3.3.1 Introduction

In 1945, Vannevar Bush developed the theoretical proto-hypertext computer system, a vision of a personal information management system named *memex* (memory extender), which influenced the development of the Personal Computer, hypertext and the World Wide Web [Bus45]. The memex would offer an individual to browse and share personal information with other individuals, all integrated into a large desk. This vision has influenced *iMeMex*, *"a unified solution to personal information management"* [BDG+07]. Nowadays thousands of files of different formats are stored in the local file systems and in many remote data sources, such as network drivers, email and web servers. This results in a heterogeneous and distributed mix of personal information. *iMeMex* is a software platform which handles the Personal Dataspace of an individual. Personal Dataspace contains all the personal information stored by one certain user [BDG+07].

iMeMex allows data management functionalities such as querying, updating, performing backup and recovery operations. Further, *iMeMex* provides a solution to close the gap between the structure of the information inside files and the outside structure of the information provided by the files&folders hierarchies. Key to this approach is to represent all available data using a single graph data model [Dit06].

3.3.2 *iMeMex* Architecture

iMeMex is a software platform that manages the whole personal Dataspace of a user by providing an own data model called *iMeMex Data Model* (iDM) and a new query language called *iMeMex Query Language* (iQL). The idea is to represent unstructured, semi-structured and structured data inside a single model. *Resource Views* represent data elements within the iDM and allow to be linked to each other in directed graph structures. *iMeMex* includes two important sublayer: *iQL Query Processor* and *Resource View Manager*. The main task of the *iQL Query Processor* is parsing iQL queries and creating concepts how to query them. The *Resource View Manager* (RVM) includes four major components: *Data Source Proxy*, *Content2iDM Converters*, *Replica&Indexes Module* and *Synchronization Manager* [DS06].

The core idea of *iMeMex*, influenced by the vision presented in [FHM05], introduces a logical layer that provides an abstraction from underlying substructure and data sources such as file systems, email servers, network shares, iPods, RSS feeds, etc. The logical layer is called Resource View Layer and is shown in Figure 3.2 within the *iMeMex* architecture.

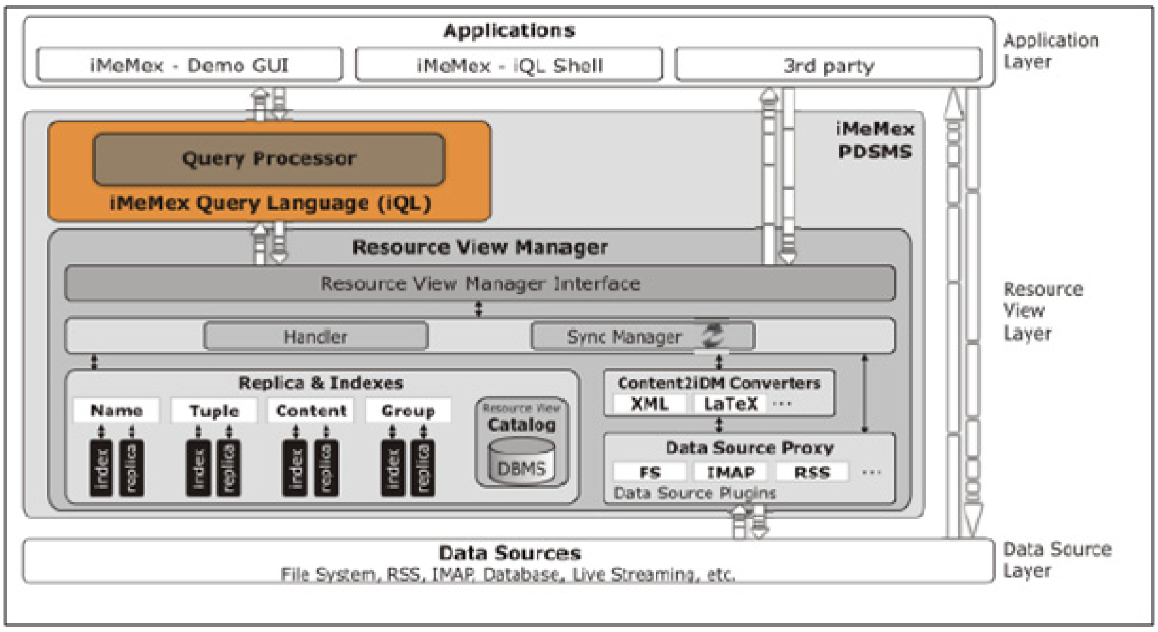


Figure 3.2: The *iMeMex* architecture [DS06]

The *Data Source Proxy* is connected to the various types of subsystems. It implies a set of Data Source Plugins that are used for data representation from the different subsystem types in the form of iDM graph structure. Currently *iMeMex* provides a plugin set for file systems, IMAP email servers and RSS feeds. The *Content2iDM Converter* additionally converts the content information extracted from the data source proxy component into iDM graph structure establishing further information provided by the structure of the iDM subgraphs. *iMeMex* offers at the moment converters for XML and LaTex.

The *Replica&Indexes Module* contains a *Resource View Catalog* wheras all resource views are registered within that catalog. For each resource view component this module offers an option to create a replica and/or an index of the data source. A Replica component is responsible for creating copies of data resources within the RVM and could be used for replication of all resource views which were extracted from remote data sources. Because of the trade-off between the distributed data that has to be queried and then shipped versus local data the *iMeMex* system must consider various strategies while creating replicas. The main task of the Index module is to create particular data structures in order to improve the look-up times. An created index of the resource views can not retrieve the original content information of the components that were deployed while creating the index, and therefor the replication itself is not provided by the indexing task.

The *Synchronization Manager* provides a persistent update function for all registered data sources. After the registration of the new data source in the RVM, the *Synchronization Manager* analyzes the data content of the data source and provides the information about the definition of created resource views to the *Replica&Indexes Module*. The *Synchronization Manager* scans permanently the data sources, registered by the RVM layer, checking for updates while synchronizing the catalog with the generated indexes and replicas.

3.3.3 *iMeMex* Data Model - iDM

iDM is used as a representation model for all kind of data including unstructured, semi-structured and structured data within a basic graph model establishing resource views that are linked to each other forming directed graph structure. If we look at the personal information of an individual nowadays we can identify a mix of heterogeneous data consisting of emails, XML documents, LATEX recourses, word documents, pictures, music and video files, address books, and so on. This personal information is usually found in various file systems, distributed machines, stored in variable file formats. The core idea of the *iMeMex* approach is to represent all available personal information within a structure of a single graph model. This approach allows querying the stored information in such a manner which can not be accomplished with state-of-the-art tools managing personal information nowadays. Figure 3.3 illustrates how *iMeMex* maps heterogeneous personal data into a single resource view graph.

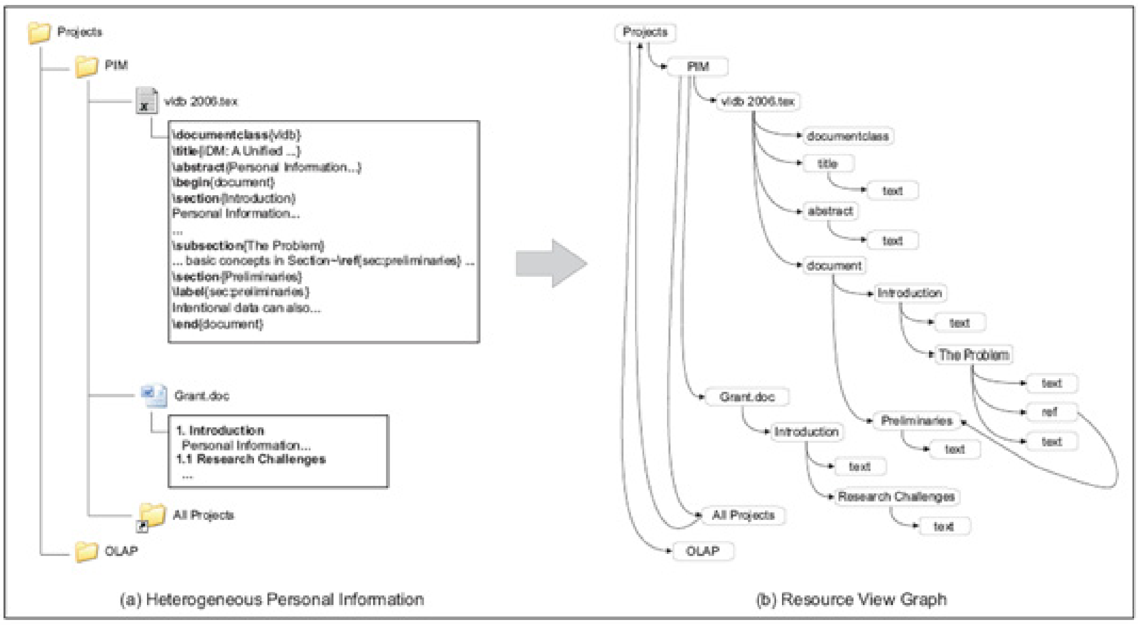
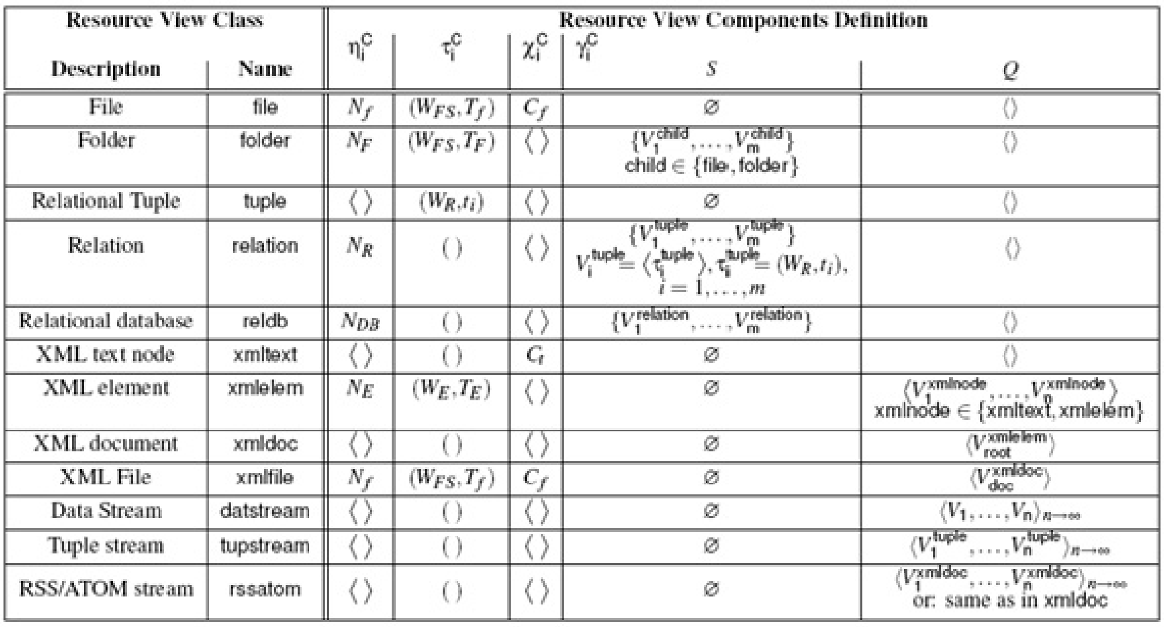


Figure 3.3: iDM represents heterogeneous personal information as a single resource view graph [Dit06]

One of the main issues developing *iMeMex* is to design *iMeMex Query Language* (iQL), because the *iMeMex* developers claim that the XPath and XQuery languages are too complex to use to query personal information systems. The IQL language requires on the one hand only a minimal learning effort and provides on the other hand more advanced users the same range of functionalities to pose more sophisticated queries. iQL includes features important for a *Personal Dataspace Management System*, offering update support, keyword search, more sophisticated querying, structured information within a graph model using a single language [Dit06]. A resource view V_i is defined [DS06] as a *"4-tuple $(\eta_i, \tau_i, \xi_i, \mu_i)$"*, representing a name, tuple, content, and group component.

Resource Views are linked to each other forming directed graph structure. Further, a resource view class C is defined [DS06] *"as a set of formal restrictions on the τ_i, ξ_i, μ_i and η_i components of all views"*. *Resource View Classes* provide a mechanism that enables an easy data integration avoiding the more complex common schema integration from various data models into iDM graph structure. Not all *Resource Views* have to belong to a certain *Resource View Class* Figure 3.4 shows important *Resource View Classes*.



Resource View Class		Resource View Components Definition				
Description	Name	η_i^C	τ_i^C	χ_i^C	γ_i^C S	Q
File	file	N_f	(W_{FS}, T_f)	C_f	$\varnothing$	$\langle\rangle$
Folder	folder	N_F	(W_{FS}, T_F)	$\langle\rangle$	$\{V_1^{child}, \dots, V_m^{child}\}$ child $\in$ {file, folder}	$\langle\rangle$
Relational Tuple	tuple	$\langle\rangle$	(W_R, t_i)	$\langle\rangle$	$\varnothing$	$\langle\rangle$
Relation	relation	N_R	$()$	$\langle\rangle$	$\{V_1^{tuple}, \dots, V_m^{tuple}\}$ $V_i^{tuple} = \langle\tau_i^{tuple}\rangle, \tau_i^{tuple} = (W_R, t_i)$, $i = 1, \dots, m$	$\langle\rangle$
Relational database	reldb	N_{DB}	$()$	$\langle\rangle$	$\{V_1^{relation}, \dots, V_m^{relation}\}$	$\langle\rangle$
XML text node	xmltext	$\langle\rangle$	$()$	C_t	$\varnothing$	$\langle\rangle$
XML element	xmlelem	N_E	(W_E, T_E)	$\langle\rangle$	$\varnothing$	$\langle V_1^{xmlnode}, \dots, V_n^{xmlnode}\rangle$ xmlnode $\in$ {xmltext, xmlelem}
XML document	xmldoc	$\langle\rangle$	$()$	$\langle\rangle$	$\varnothing$	$\langle V_{root}^{xmlelem}\rangle$
XML File	xmlfile	N_f	(W_{FS}, T_f)	C_f	$\varnothing$	$\langle V_{doc}^{xmldoc}\rangle$
Data Stream	datstream	$\langle\rangle$	$()$	$\langle\rangle$	$\varnothing$	$\langle V_1, \dots, V_n\rangle_{n\to\infty}$
Tuple stream	tupstream	$\langle\rangle$	$()$	$\langle\rangle$	$\varnothing$	$\langle V_1^{tuple}, \dots, V_n^{tuple}\rangle_{n\to\infty}$
RSS/ATOM stream	rssatom	$\langle\rangle$	$()$	$\langle\rangle$	$\varnothing$	$\langle V_1^{xmldoc}, \dots, V_n^{xmldoc}\rangle_{n\to\infty}$ or: same as in xmldoc

Figure 3.4: Important resource view classes to represent files&folders, relations, XML, data streams, and RSS [DS06]

3.3.4 iTrails: Pay-as-you-go Information Integration in Dataspaces

There are two different approaches in order to query a set of heterogeneous data, the schema-first and no-schema approach. The schema-first approach (SFA) implies semantic integration over data sources in order to execute sophisticated queries over these data and receive precise answers. In order to accomplish an integrated view over recent created data sets SFA implementor has to provide exact mappings between every source schema of integrating data sources.

The no-schema approach (NSA) handles all data sources in its original source schema without having to map corresponding data sources, and perform information integration in order to provide an unified view over the data. Keyword search and basic structure queries are supported providing all needed information over those data sources [SDK$^+$07].

The main idea of the iTrails approach is to enrich the already structured data with further information about possible relations between existing data and already executed queries adding so called hints (trails) to the no-schema approach. The new created trails that represent the semantic information among data allows more intelligent answers while approaching schema-first aproach in a "pay-as-you-go" [FHM05] fashion. Figure 3.5 shows a Dataspace consisting of four data sources.

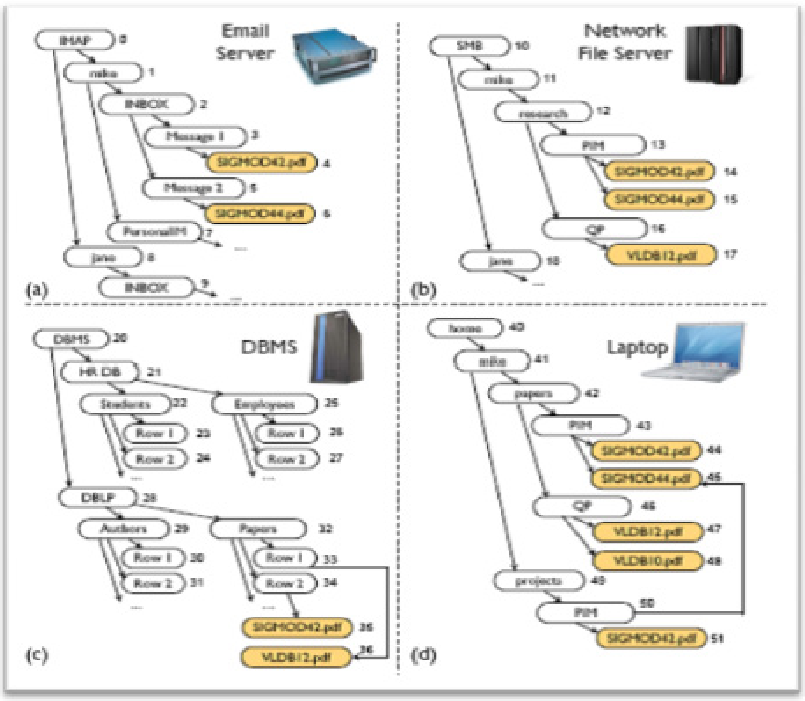


Figure 3.5: A dataspace consisting of four heterogeneous data sources [SDK+07]

iDM graph model represents the data included in the data source. Let's assume that a user wants to query all *yesterday* modified or added pdf documents. Depending on the data source schema and different attribute names used for each data source a user would have to specify separate and complex queries for each data source to evaluate the term "yesterday". The main goal is to provide a way that allows specifying the same query for every data source by simply using the pdf *yesterday* expression. The iTrails approach provides additional trails (hints) information across the generated graph model creating relations and enriching the knowledge over the integrated data in order to achieve this main goal. While performing queries iTrails is able to implement and define new hints while gradually improving the result quality of afterwards executed queries. The iTrails technique intents on modeling relations among data by providing additional semantic information in "pay-as-you-go" manner allowing the Dataspace to grow.

iTrails search queries can provide further semantic information including query expansions and a use of dictionaries, such as Wordnet [RMC07], automatically creating iTrails hints expressing for example the relation between auto and car by specifying a new trail. iTrails can also be used as an data integration tool by integrating mediated data sources, for example hidden web-databases, participating in the dataspace. There are four basic concepts defining a set of trail:

- (1) trails definition using a drag&drop frontend
- (2) trails created based on a feedback of a user
- (3) trails (semi-) automatically created from content
- (4) trail defined from collections provided by shared web platforms

Not only the definition of the trail itself but also a quality of a certain trail can be modeled defining a probability value:

"$0 \leq p \leq 1$"

3.3.5 Conclusion

Personal Information Management has become a key necessity of almost everybody. At the same time, it has become clear that what is missing is a unified approach to create physical and logical data independence to enable a personal Dataspace.

The *iMeMex* Personal Dataspace Management System introduces a logical layer on top of the data sources that provides a mechanism for managing personal logical and physical information. *iMeMex* addresses some major research challenges. First, a definition of iDM, an unified data model, which represents the mix of heterogeneous data identified in personal Dataspaces as a hierarchical graph structure. Second, a query language, called iQL offering efficient querying on the iDM data model. These research issues are the main exploration part of the *iMeMex Personal Dataspace Management System* future work.

3.4 Storage Resource Broker

3.4.1 Introduction

SRB is a middleware developed by The San Diego Supercomputer Center (SDSC) [BMRW98] which allows accessing heterogeneous distributed data including, filesystems, database systems and archival storage systems. SRB uses a metadata catalog service, MCAT, to provide a means to organize data in a *"collection- oriented view"*. MCAT provides a set of APIs which allows attribute-based access to data collections and items and provides an execution of distributed applications in order to establish data access at every storage site of the distributed environment. The API offers the capability to information discovery, identification of required data collection, and selection and data retrieval of various distributed data sources which may be located in WAN networks.

3.4.2 SRB Features

The data resource access which is managed by the SRB MCAT catalog is provided by the use of attribute names of the data sets independent from the physical file location, maintaining a location transparency.
The data sets collected by the SRB include descriptive as well as system metadata. Descriptive metadata represents the contents, whereas system metadata enables *"location and access control information"* [RWM03a] for collected data sets. Data recorded using the SRB is arranged as a hierarchy of collections and sub-collections, whereas data sets arranged by the same collection can be distributed across heterogeneous storage environments.
The SRB can organize data access to archival resources such as *"HPSS, UniTree and ADSM, file systems such as the Unix File System, NT File System and Mac OSX File System and databases such as Oracle, DB2, and Sybase"* [RWM03a]. Further, it offers a logical representation for storage system description, digital file items, and data sets and supports characteristic facilities, which can be applied in *" digital libraries, persistent archive systems and collection management systems"* [RWM03a]. A replicated data, the authentication information on access control of items and data sets is managed by the SRB facilities. The SRB system offers search capabilities based on the user-defined metadata, which can be saved as a collection or a object. Figure 3.6 shows an overview of the architecture of SRB.

Tickets Tickets is a mechanism for managing data read access. A user can grant a ticket permission to other users, or group of users, on that data items or collection they have control privilege. These tickets are only allowed for a certain period of time or for a specified usage number to limit the number of allowed access operations.
There are two different types of users in MCAT: registered and unregistered users. For registered users the required metadata is already provided by the MCAT. Unregistered users act as guests in the MCAT Catalog. Both types of users can issue tickets. Unregistered users need a special request while connecting to the SRB server whereas their range of functions is limited to read data for which they have issued a ticket.

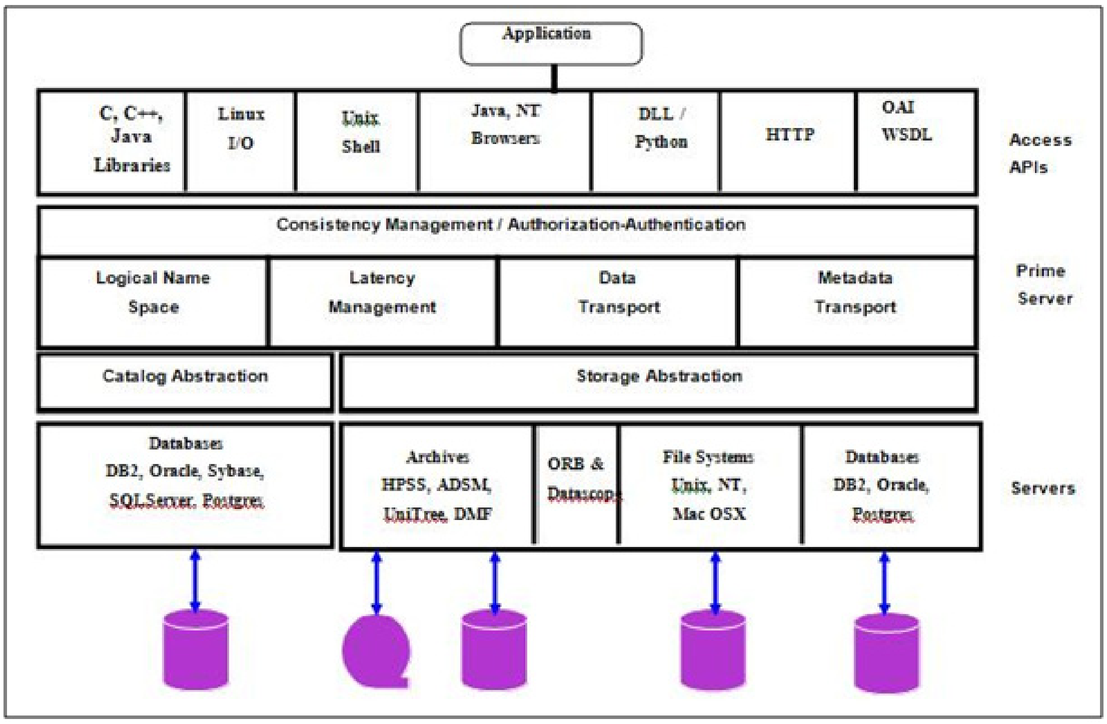


Figure 3.6: SRB architecture [BMRW98]

Physical Storage Resources (PSRs) and Logical Storage Resources (LSRs)
Depending on the data source, PSR is defined as [BMRW98]:

- *"For storage resources with file system interfaces: a PSR is the (hostname, pathname) combination, representing a certain directory path on a certain host."*
- *"For storage resources with database system interfaces: a PSR is the (hostname, database id, table id) triple, representing a host, a database on that host, and a table within that database."*

A group of declared PSRs, which for example can be a table in an Oracle database, directory path in HPSS or AIX filesystem, are joined together forming a single logical storage resource (LSR). Client APIs provide references to LSRs.
A replication of a data set which is linked to the logical storage resource (LSR) is performed replicating each PSR. Data items are represented as collections, which is accomplished by the use of LSRs, whereas every data set can be read by any corresponding PSRs [BMRW98].

Access to distributed data resources is provided by a federation of SRB servers, that manage a distinct set of PSRs, and allows SRB server to act as clients to each other. This enables a client application to access distributed data, even if there is no direct connection between application and the controlling SRB server.

Authentication The SRB *communications protocol* provides a set of different authentications. Password information used as authentication is managed by the MCAT. Adopted from the SEA Systems, SRB implements data encryption and authentication of users, which is provided by a RSA mechanism called *a public-private key*. The update facilities performed on metadata and storage resources are executed by SRB MCAT Catalog system, which can be specified by the user while generating a data set or a collection. Logging operations can be turned off while accessing an object.
Figure 3.7 illustrates the steps of a client-server connection process. First, client sends a connect request to the SRB Master. After authentication process done by the SRB Agent, using the SEA library, the SRB Master returns the connection respond to the client. Each client connection is managed by a distinct SRB Agent.

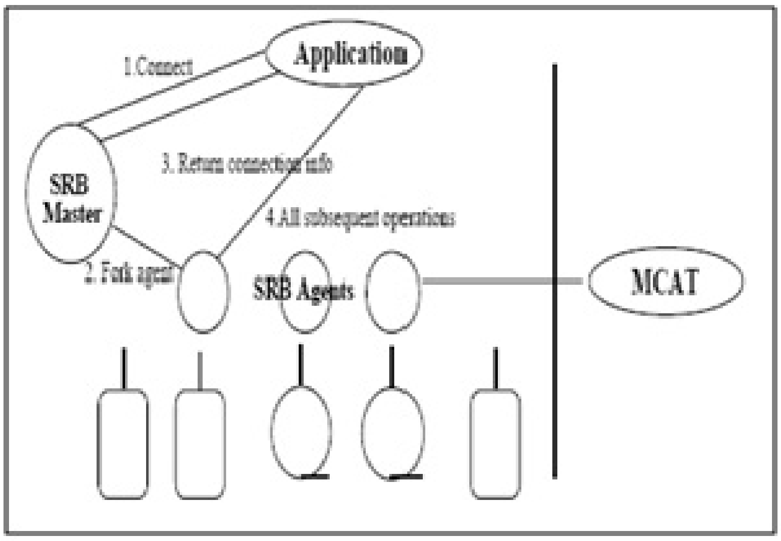


Figure 3.7: The SRB process model [BMRW98]

3.4.3 System Architecture

The core system architecture of the SRB middleware application builds upon two processes, one representing a SRB Master daemon and the other a SRB Agent, whereas each Agent references a corresponding Master. Each of them is identified by a hostname and a port number, while being responsible for a particular set of PSRs.

The LSR and PSR system metadata is managed by the MCAT service, whereas the SRB Agent frequently connects the MCAT sending a request for metadata in order to provide the Agent with needed information for client communication.

The Metadata Catalog (MCAT) The MCAT is a catalog service providing a facilities for storing information by the SRB system. SRB distinguishes between two kinds of data: internal system data (system management requirements) and application data (data items). As already mentioned MCAT provides an attribute-based access to data collections and items, which can be used to identify the data resources rather by a logical attribute name referencing a location of the data itself. Additionally, some other important operations such as a control of the access management, hierarchical organization and content information of data collections, and the ticket system contain metadata that are handled by the MCAT catalog.

The SRB Server The SRB Server is in constant communication with the clients receiving their requests and sending responses after they have collected the requested information on data sets provided by the MCAT service.
SRB represents *a federated server system* [RWM+03b], whereas every SRB server is responsible for a particular group of storage resources. The implementation of the federated SRB server offers following advantages [RWM+03b]:

- *"Location transparency"* - the data can be provided by connecting from one federated SRB Server to another server using a logical attribute name or the item and data collections.
- *"Improved reliability and availability"* - the federated servers manages the data replication being performed on various hosts and storage systems in order to provide efficient load facilities.
- *"Logistical and administrative reasons"* - while using one single authentication environment, the storage systems can be proceeded on distributed hosts implementing different security mechanism.
- *"Fault tolerance"* - if one of the storage systems is not available, the global identifier is aware of other available replicas, while automatically linking to them.
- *"Integrated data access"* - the data access is provided the same way locally as it is for distributed resources establishing an integrated access to distributed data environments.

- *"Persistence"* - the replicated data elements on different storage systems are represented by their logical attribute names providing the same data item properties and so maintain an unique access management.

3.4.4 The SRB Client

The SRB Client offers an user tool that allows communication by sending user requests to SRB Servers. Main Client implementations are:

- Windows GUI named InQ that offers a file-manager-like interface providing an easy way for users to manage their data stored on SRB. InQ facilitates various management operations such as traditional drag-and-drop operations for a transfer of files, a supports authentication tools managing the access, data replication, metadata management etc [fEe07b].
- A web based Client, named MySRB [RWM+03b]. My SRB is described within the next Section.

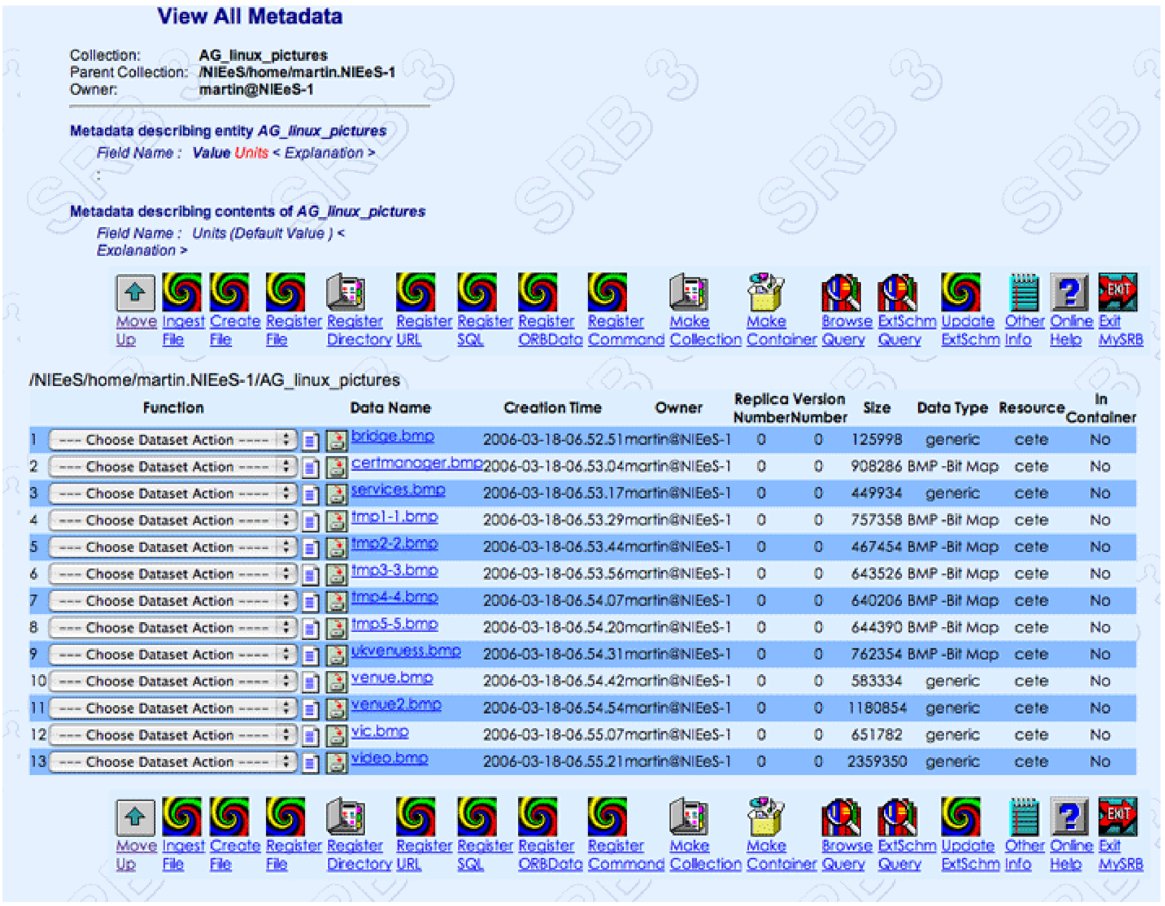
MySRB - a web-based interface to the SRB MySRB is a web-based interface that allows secure access and share of distributed data collections and data items stored in the SRB. Files can be organized in a hierarchical way provided by the data collection and items structure referring to a logical attribute notation representing the physical resources.

MySRB supports three basic facilities [RWM+03b]:

- *"collection and file management"* - facilities to collect, delete and create data, further operations to ingest, reload, registration, replication, and data movement, authentication. Versioning and Locking Operations are a part of a future work.
- *"metadata handling"* - facilities to ingest, copy, extract, maintain, update, and delete and standardized metadata, which can be derived from element definitions such as the Dublin Core, or based on other Semantic Web definitions.
- *"access and display of files and metadata"* - facilities to browse files organized as data items and data collections and operations to query all kind of metadata such as the system and user metadata.

MySRB bulids upon the secure protocol (https) using the 128-bit authentication, provided by RSA. Every session is attached to one particular time limit (max limit are 60 minutes) and each of the MySRB sessions is provided by a key, which is unique.

The web-browser interface for MySRB is used to display metadata about the data items and collections queried by the user, shown in Figure 3.8. When opening a file, the information about the attributes and the content of the data sources is displayed in the main window as illustrated in the Figure below. Another MySRB interface is used to visualize the information about different kind of metadata such as the information about attribute values and their properties, and metadata defined by users.



View All Metadata

Collection: AG_linux_pictures
Parent Collection: /NIEeS/home/martin.NIEeS-1
Owner: martin@NIEeS-1

Metadata describing entity *AG_linux_pictures*
Field Name : Value Units < Explanation >
:

Metadata describing contents of *AG_linux_pictures*
Field Name : Units (Default Value) < Explanation >

/NIEeS/home/martin.NIEeS-1/AG_linux_pictures

	Function	Data Name	Creation Time	Owner	Replica Number	Version Number	Size	Data Type	Resource	In Container
1	--- Choose Dataset Action ----	bridge.bmp	2006-03-18-06.52.51	martin@NIEeS-1	0	0	125998	generic	cete	No
2	--- Choose Dataset Action ----	certmanager.bmp	2006-03-18-06.53.04	martin@NIEeS-1	0	0	908286	BMP -Bit Map	cete	No
3	--- Choose Dataset Action ----	services.bmp	2006-03-18-06.53.17	martin@NIEeS-1	0	0	449934	generic	cete	No
4	--- Choose Dataset Action ----	tmp1-1.bmp	2006-03-18-06.53.29	martin@NIEeS-1	0	0	757358	BMP -Bit Map	cete	No
5	--- Choose Dataset Action ----	tmp2-2.bmp	2006-03-18-06.53.44	martin@NIEeS-1	0	0	467454	BMP -Bit Map	cete	No
6	--- Choose Dataset Action ----	tmp3-3.bmp	2006-03-18-06.53.56	martin@NIEeS-1	0	0	643526	BMP -Bit Map	cete	No
7	--- Choose Dataset Action ----	tmp4-4.bmp	2006-03-18-06.54.07	martin@NIEeS-1	0	0	640206	BMP -Bit Map	cete	No
8	--- Choose Dataset Action ----	tmp5-5.bmp	2006-03-18-06.54.20	martin@NIEeS-1	0	0	644390	BMP -Bit Map	cete	No
9	--- Choose Dataset Action ----	ukvenuess.bmp	2006-03-18-06.54.31	martin@NIEeS-1	0	0	762354	BMP -Bit Map	cete	No
10	--- Choose Dataset Action ----	venue.bmp	2006-03-18-06.54.42	martin@NIEeS-1	0	0	583334	generic	cete	No
11	--- Choose Dataset Action ----	venue2.bmp	2006-03-18-06.54.54	martin@NIEeS-1	0	0	1180854	generic	cete	No
12	--- Choose Dataset Action ----	vic.bmp	2006-03-18-06.55.07	martin@NIEeS-1	0	0	651782	generic	cete	No
13	--- Choose Dataset Action ----	video.bmp	2006-03-18-06.55.21	martin@NIEeS-1	0	0	2359350	generic	cete	No

Figure 3.8: MySRB view of a data collection [fEe07a]

User can organize the attached their files by adding them to already declared collections or by defining a new sub-collection. All files are referenced by their logical name representing the actual source, which can be a file, filesystem, or a database, managed by the SRB. The replicated files use the same logical representation of the underlying data source.
The MySRB application offers a variety of facilities for manipulation of various types of metadata and data collections recorded in the SRB system.

In MySRB we distinguish between five different kinds of metadata [RWM+03b]:

- *"system-defined metadata"* - managed by the SRB, all system metadata can be viewed and used for queries by any user.
- *"user-defined metadata and type-oriented (domain-oriented) metadata"* - only those users who own a permission on created data collections may define metadata.
- *"file-based metadata"* - can not be used for queries, contains object metadata in a file.
- *"annotations and commentary metadata"* - these metadata can be declared by anybody who owns a object read permission.

3.5 iRODS - integrated Rule-Oriented Data Systems

3.5.1 Introduction

iRODS (integrated Rule-Oriented Data Systems) is *"a second generation data grid software"* [iRG08b] developed by the SDSC Storage Resource Broker team and collaborators. It provides a standardized view and access to distributed data sources across WLAN networks.
The Storage Resource Broker (SRB) system that was introduced in the previous chapter, offers a logical abstraction concept of data collections and data sets provided by the use of logical attribute names that are independent from the physical file location, establishing a location transparency for clients communicating with SRB servers. iRODS bulids upon SRB abstraction concept by providing further data management abstraction. This abstraction process is named policy abstraction [RWMS06].

3.5.2 Basic Concepts

iRODS implements a RULE Engine, which is responsible for rule interpretation in order to evaluate how to reply to different client requests. iRODS is open source under a BSD-type license.

The present SRB facilities do not provide easy data management facilities for manipulation of created data sets and data collections. Let's assume we have a user

who would like to define a facilities that certain data collections are persistent so that no one, not even the creator of the particular data item can perform delete operations on such data sets, whereas other files should not be affected. This type of operation are very challenging to implemented using current SRB facilities. Another interesting feature that might be difficult to provide addressing present SRB operations is a implementation of access control facilities for more sensitive data.
A additional example could be a replication of already created collection items across various storage systems, which also requires complex implementation methods using the SRB [iRG08a].

iRODS system can be classified as a middleware application implementing adaptive features. The *adaptive middleware architecture* (AMA), which is the core of the iRODS system, can be modified and adapted by users in order to accomplish their tasks with no need to perform complex code implementations.
iRODs approach uses Rule Oriented Programming (ROP) for achieving adaptive middleware architecture [iRG06b]. All operations that are accomplished using the iRODS system are proceeded as rules in the iRODS rule engine. These rules are initialized by application calls, while controlling the performed operations.

3.5.3 Rule Oriented Programming

Rule-oriented programming provides user facilities for modifying and controlling the operation functionality of a certain process without relaying on the system or application developers. ROP provides *" small, well-defined operations that perform a certain task"* called Micro services [iRG06b], which are defined by system and application programmers.
While executing the rules user can change the task operations by applying micro-services or my modifying already implemented code provided by micro-services. Different micro-services can be linked together in order to provide a higher level facilities in form of actions. Each action might be executed in various manner containing a certain number of micro-services. An action provides a particular task name, describing the executed operation, whereas the micro-services indicate the corresponding task.

There are two mechanisms for finding the best set of micro-services used for an action [iRG06b]:

- *"condition"* - provides permission control facilities, which can be applied on any micro-service. This mechanism introduces the so called *"(action, condition, chain)-triplet"* [iRG06b], which implements a "rule" in ROP system.

- *"priority"* - this approach is responsible for controlling and testing the order of an executed rule. These rules that have low number will be executed first.

Each time a set of micro-services is attached to a new action and afterwards performed, the system calls the corresponding rule service. If there is a failure while executing micro-services, meaning that the corresponding action could not be performed, the next rule with lower number is executed.

3.5.4 iRODS Architecture

Main facilities of the iRods architecture consists of [RWMS06]:

- Data grid architecture - provides a client-server application and shared data sources.
- A database system - managing the data properties and operation which can be applied on them.
- A rule system - rules executing management.

The iRODS Rule System The iRods Rule Engine is implemented on each iRods server and it represents the core of the iRODS Rule System. Depending on the executed rule and its function, a set of micro-services grouped as an action can be executed by the rule engine.

In iRODS we distinguish between two kinds of rule classes [iRG06a]:

- System Level rules - are server side rules, which are responsible for system management facilities. Examples for such policies are: authentication and access control, data management operations such as representation, replication, data extraction, data replacement and distribution, automotive service logging and auditing operations.
- User Level rules - are client side rules, which are executed using the *irule* command or the *rcExecMyRule* API. Users can send a request to a server asking to execute a particular set of micro-services. All data items are stored on various iRODS servers where requested operations can be performed.

The Rule Execution server can queue and execute rules attached to actions, which are performed in the background controlled by the Delayed Execution Service. Such actions, which consist of different micro-service operations, which are queued and executed afterwards, are for example checksum operations and data replication. The architecture of the iRODs system is illustrated in Figure 3.9.

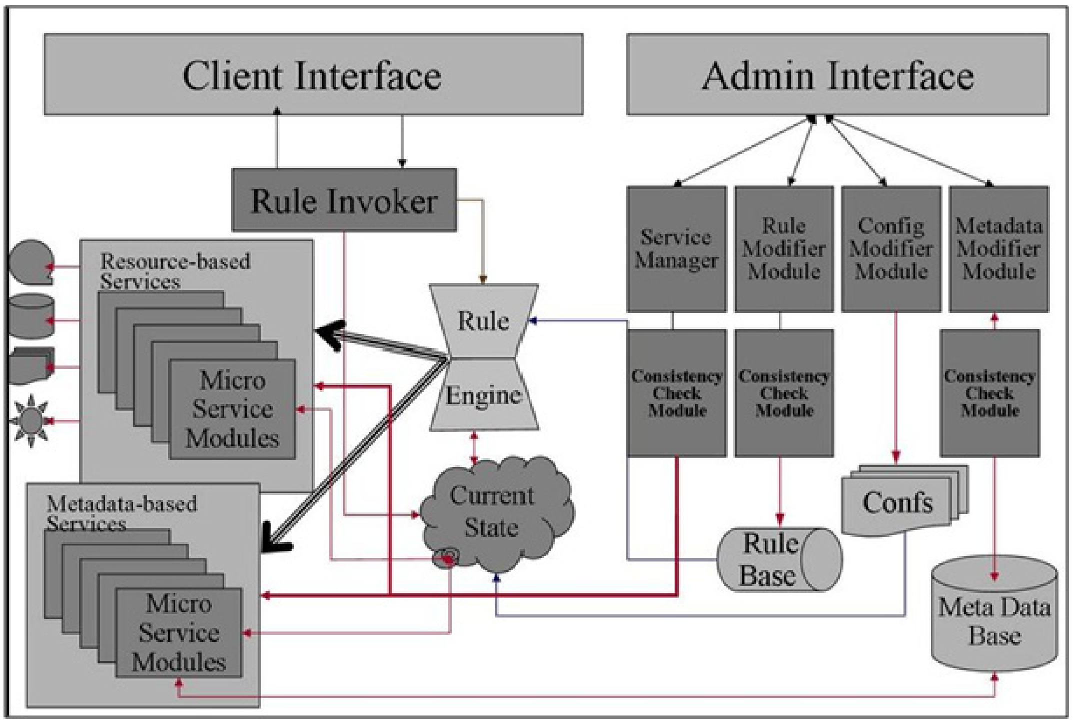


Figure 3.9: The architecture of the iRODS system [RWMS06]

3.5.5 Components of an iRODS System

Virtualization
Every data collection or data source communicating with the iRODS system is represented by its logical name, which is independent from the physical location of the data source itself. The Metadata Catalog manages the data associated with the virtual representation of the logical names [iRG08b].

Data Transport
iRODS Data can be transfered among distributed storage systems in different manner. The files being moved from one system to another can be divided into smaller files and so separately transfered (bulk method) or at once (parallel method). The system choses the method depending on the size of the file [iRG08b].

Metadata Catalog

The iRODS metadata catalog is called iCAT. It offers facilities for metadata management. It handles system and user-defined metadata, and abstract physical-to-logical name mappings. The iRODS V1.0 iCAT can be implemented using various databases [iRG08b].

The Rule Engine

The Rule Engine implements a set of rules which are attached to executed actions that contains a set of micro-services. Any executed task triggers a rule [iRG08b].

The Execution Engine

Micro-service are controlled by the execution engine and any iRODS server. A set of micro-services grouped into actions can also be executed remotely, wheres the result are returned after the action has been performed [iRG08b].

The Scheduler

iRODS system provides a scheduler which can delay task execution. The iCAT handles schedule activity [iRG08b].

Messaging System

Message System is part of the iRODS future work.

3.6 IBM WebShepre Information Integrator

3.6.1 Introduction

WebSphere Information Integrator, formerly known as DB2 Information Integrator enables applications to access distributed heterogeneous data resources such as DB2 UDB, Oracle, and Sybase, and data resources which do not build upon relational data models such as text files and unstructured documents. Further it provides access to XML related sources and documents, and data being accessed using Web Service application.

WebSphere Information Integrator manages metadata of the corresponding data sources using a unified metadata catalog [Erf05], which is provided by a federation of DB2 UDB database engine.

The federated server system provided by the Information Integrator offers transparent object virtualisation while acting as a virtual database. The distributed objects are organized in a table-like manner consisting of:

- *"A DB2 instance"* - represents an instance of a federated server.
- *"A federated database"* - one particular database operate as a federated database for different relational and non-relational data resources.

- *"data sources"* - a number of data sources itself representing the distributed objects.
- *"Client applications"* - remote applications and users access server databases requesting for different data objects, which are identified by their logical nicknames referenced by the clients.

A information integrator federated system provides software components named wrappers which communicate with various data resources. A so called wrapper module provide various wrapper implementations for each data source, offering data access facilities used by federated servers in order to establish connection to these data sources, execute different operations and fetch desired data elements as illustrated in Figure 3.10 below. Each wrapper represents data using a table-like structure [ACD04].

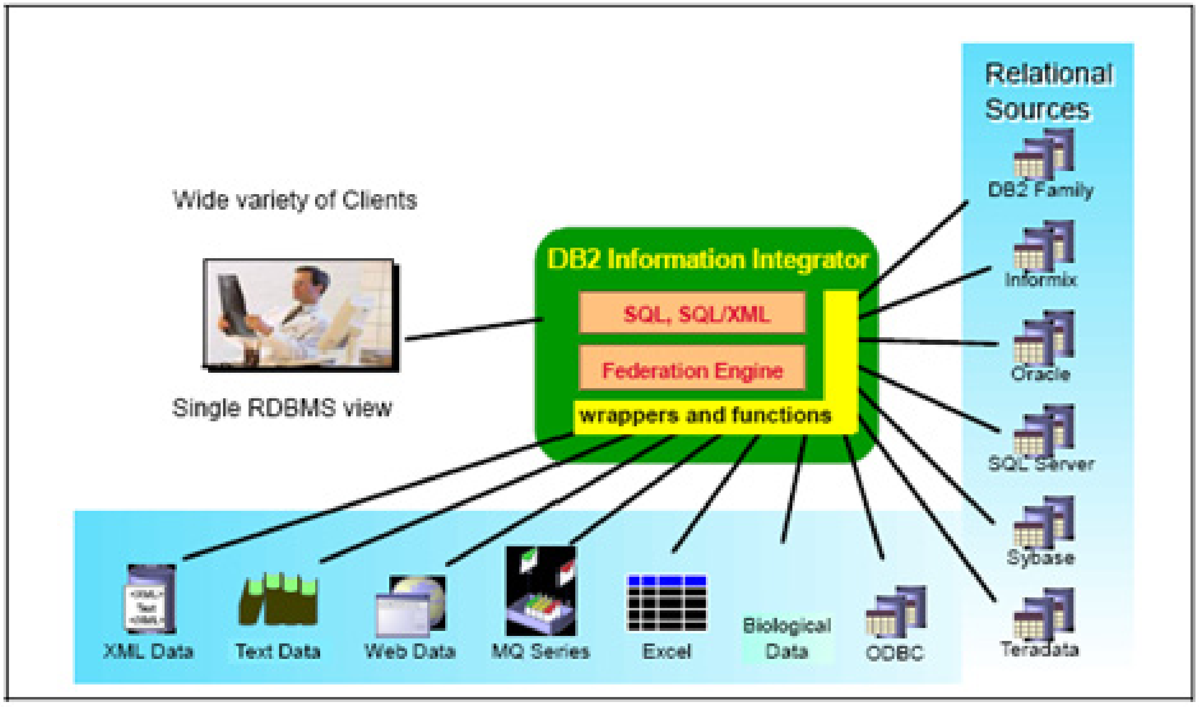


Figure 3.10: WebSphere Information Integrator data federation using wrapper [ACD04]

3.6.2 Overview

Federated databases act as one single database for client application and users connecting to the database. It implements a metadata catalog identifying and managing the metadata information about data source and their properties, which is used for querying the data sources. Because the data sources are organized as relational tables providing a unified view over such a data, the federated server is able to execute SQL queries as if querying a single data source. Non-relational data sources can be mapped to relational data structures allowing unified SQL querying, even if the non-relational sources do not offer SQL facilities [BAB03].

WebSphere Information Integrator has four major components [ACD04] shown in Figure 3.11:

- *"The relational wrappers"* - implements various wrappers for relational database access such as Sybase, Microsoft SQL Server, Oracle, ODBC, and Teradata data sources.
- *"Non-relational wrappers "* - implements various wrappers for mapping non-relational data sources, such as XML files, various research files from different domains such as chemistry, biology and genetics.
- *"Global catalog"* - handles the whole federated system metadata information, including the data items information (operations, tables, attributes) provided by the federated system, metadata information about wrapper module implementations, logical nickname representations, and the information about the data sources itself.
- *"DB2 Net Search Extender"* - provides search mechanism executing SQL queries across various files and documents using automatically updated index data, which are loaded into memory providing effective query results.

The global catalog provides nickname, index, and attribute information of distributed data sources [ACD04]. This metadata information handled by the global catalog is used by the WebSphere query optimizer to execute SQL queries.

3.6.3 WebSphere Information Integrator functions and objects

Four of the following Information Integrator components must be defined in order to provide data access to the Federated Server [BAB03]:

- *Wrappers* - provide data access modules which store information about various data sources and their protocols. Additionally the properties and characteristics of distributed data sources are recorded.

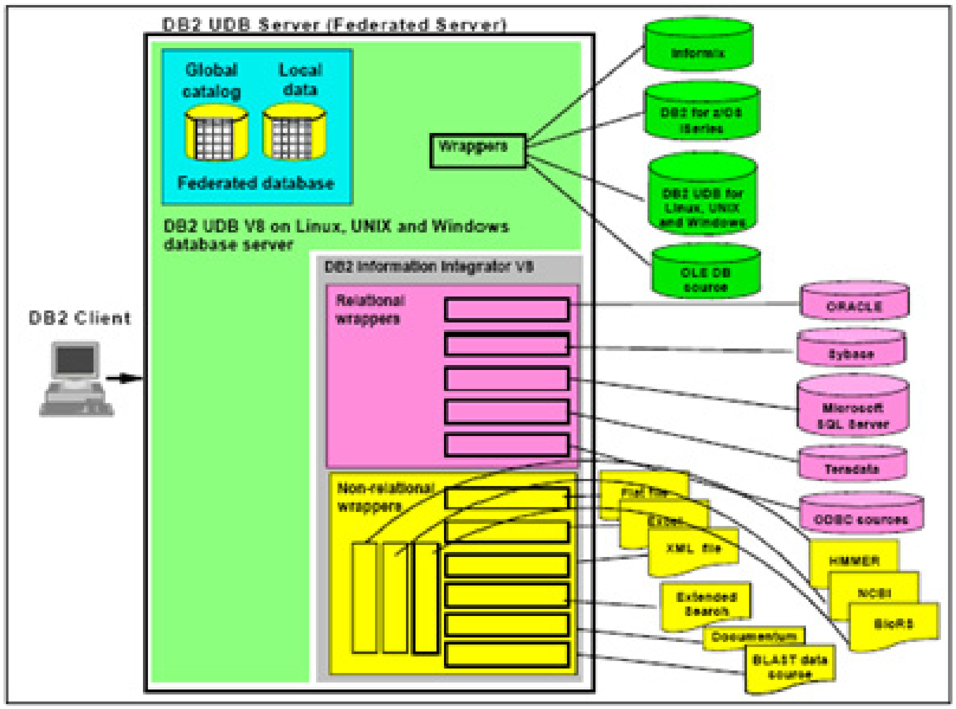


Figure 3.11: WebSphere Information Integrator components [ACD04]

- *Servers* - wrappers provide data access to particular sources store at servers, whereas each server is identified by a DLL statement, which refers to one certain data source.

- *Nicknames* - provide logical abstraction of data sources, mapping the data to local table. Each data source is identified by one unique nickname used by a server for executing SQL queries.

- *User mapping* - user access ID information on server-side is mapped to a password and data source ID used for further connections.

- *Data type mappings* - data types are mapped to DB2 data types in order to query data, whereas the wrappers defines the mapping facilities.

- *Function mappings* - the wrappers implement also special DB2 mapping facilities.

- *Global catalog* - handles the whole federated system metadata information, including the data source information (attributes and operations), metadata information about wrapper module implementations, logical nickname representations, and the information about the data sources and mapping functions.

Federated Server Each DB2 server having a running WebSphere Information Integrator on it represents a *federated server*, which can be implemented on Unix, Linux and Windows operating systems. Client applications have integrated data access across distributed data sources, which are handled as one unified data source, having transparent format, location and executed operations. The WebSphere Information Integrator provides facilities for manipulation of XML sources and documents. The query results can be usual sql statements or XML documents, which can be mapped into XML Schema. Federated Servers can be accessed through usual database or service clients, which if preformed remotely are named pushdown operations.

The data source registration on the WebSphere Information Integrator is provided using following steps [BAB03]:

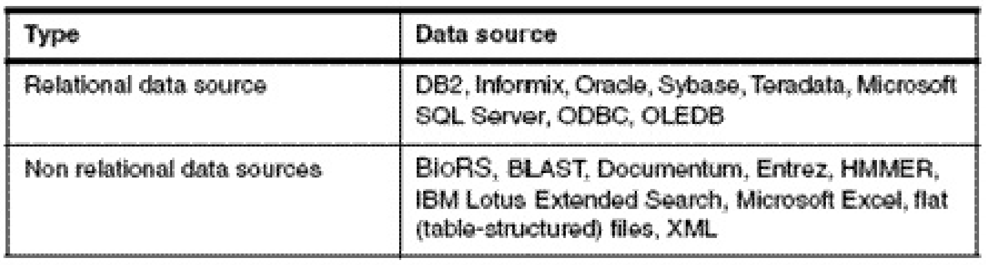
- *registration of the wrapper module* - each wrapper is registered in the database providing the data source access information.
- *data source-server definition* - each data source need to be declared as server of the federated system.
- *the authentication information* - registration of the remote authentication facilities provided by user mappings
- *the federated system connection* - the SQL query statements should be directly executable on each data source.
- *definition of data type mappings* - additional mapping definitions should be provided if required to be applied on particular data sources.
- *nickname and table identification* - each data source refers to a corresponding nickname, which identifies the data sources.

There is additional nickname metadata information, named *column options*, describing data source column objects, which can provide federated servers with further information. This metadata used by wrappers indicates how the column data should be handled.
Each data source contains a set of index information declared as the *index specification*. Each time a new nickname is registered, the metadata global catalog saves information about index specification of data sources adding corresponding table information about the sources.

Wrapper *Wrappers* offer a module library named *wrapper module*, providing access information about data sources which communicate with federated servers. For each date source different wrapper is implemented.
A wrapper consists of following operations [BAB03]:

- *Federated object registration* - the object property information is registered by the a wrapper.
- *Communication with the data source* - the entire wrapper-data source communication information is managed by a wrapper.
- *Services and operations* - a wrapper provides various operation facilities of different types of data sources. These operation might be query statements, update facilities, data manipulation operations, transactions, etc. [ACD04].
- *Data modeling* - wrappers support various mappings of data sources into a table representation managed by the federated server engine [ACD04].

WebSphere Information Integrator provides access to data sources illustrated in Figure 3.12 below.



Type	Data source
Relational data source	DB2, Informix, Oracle, Sybase, Teradata, Microsoft SQL Server, ODBC, OLEDB
Non relational data sources	BioRS, BLAST, Documentum, Entrez, HMMER, IBM Lotus Extended Search, Microsoft Excel, flat (table-structured) files, XML

Figure 3.12: Supported data sources [ACD04]

3.6.4 Replication

With the release of *WebSphere Information Integrator 8.2*, IBM has introduced a function called queue replication, also known as Q replication.
The data transfer across various relational data sources is provided by the replication server, which might be used as a source and as well as destination replication server. Examples for such servers are *DB2, Oracle, SQL Microsoft and Sybase server, etc.* [BAB03].
The replication architecture is shown in Figure 3.13:

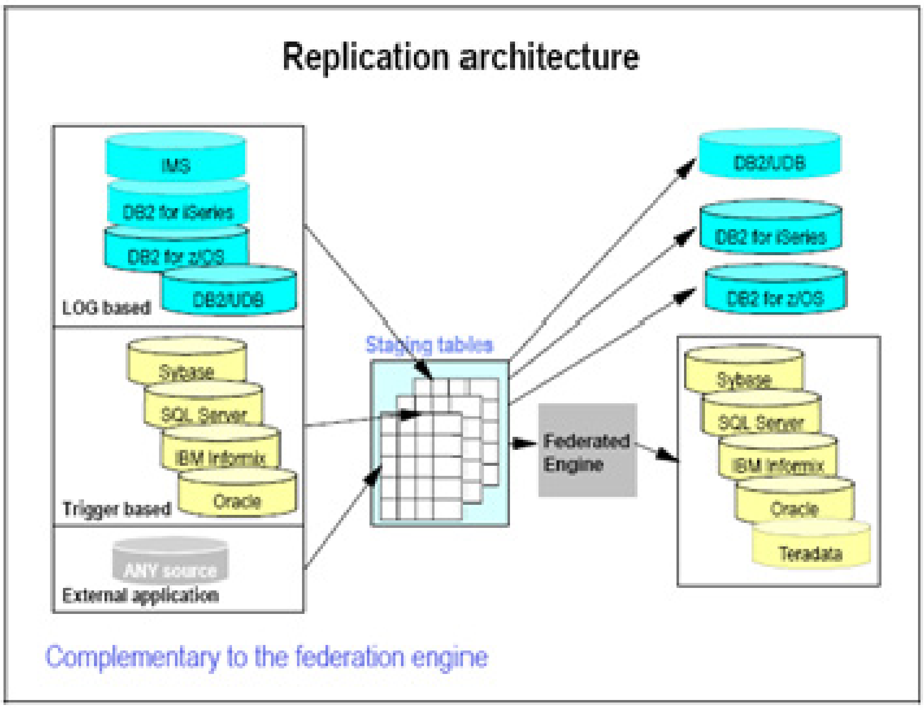


Figure 3.13: Replication architecture [BAB03]

Q replication builds upon message queue mechanism in order to manage replication facilities by transporting data changes across replicated data source. Q replication is made of two components, *Q Capture* and *Q Apply*. The role of Q Capture is to persistently perform database checks on recovery log, and generate messages, which provide performed data changes. Those messages are afterwards loaded into one or more MQ queues, as illustrated in Figure 3.14.

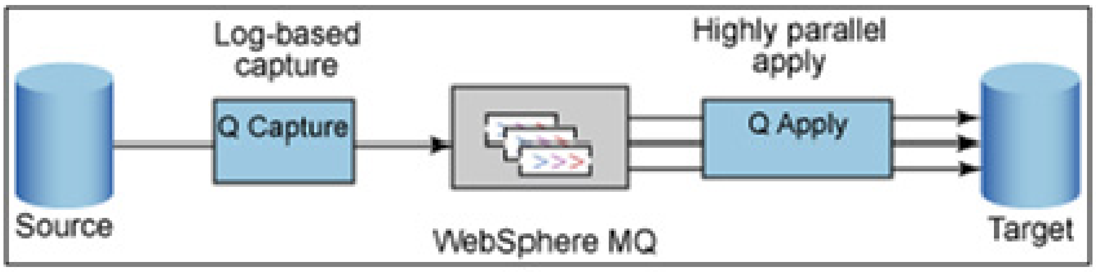


Figure 3.14: Q replication [Asc03]

There are three types of Q Replication [Asc03]:

- *unidirectional replication* - the replication is done across WebSphere MQ queues from source to target table.
- *bidirectional replication* - the replication is performed across changes in table copies on two servers in any direction.

- *peer-to-peer replication* - the replication is performed across changes in table copies on two or more different servers, and is often used for data synchronisation.

3.7 Google Desktop Search

Google desktop search is a freeware desktop search tool propagated by Google. It features a Google-like Web interface that offers an easy way to search for personal information.
The tool provides a keyword search mechanism over a variety of personal resources including emails, file directories, video and music files, photos, viewed Web pages, and more [Inc08] by indexing the supported data types. It generates file and other relevant user data copies every time a user views the data allowing the user to access the stored information afterwards. As a result, a user can access and find needed information even after it has been deleted.
The Google Desktop tool is running a local Web server which listens to port 4664. The application handles only local request in order to provide more security [AAS07]. Figure 3.15 shows the Google desktop search interface.

3.8 Phlat & Windows Desktop Search

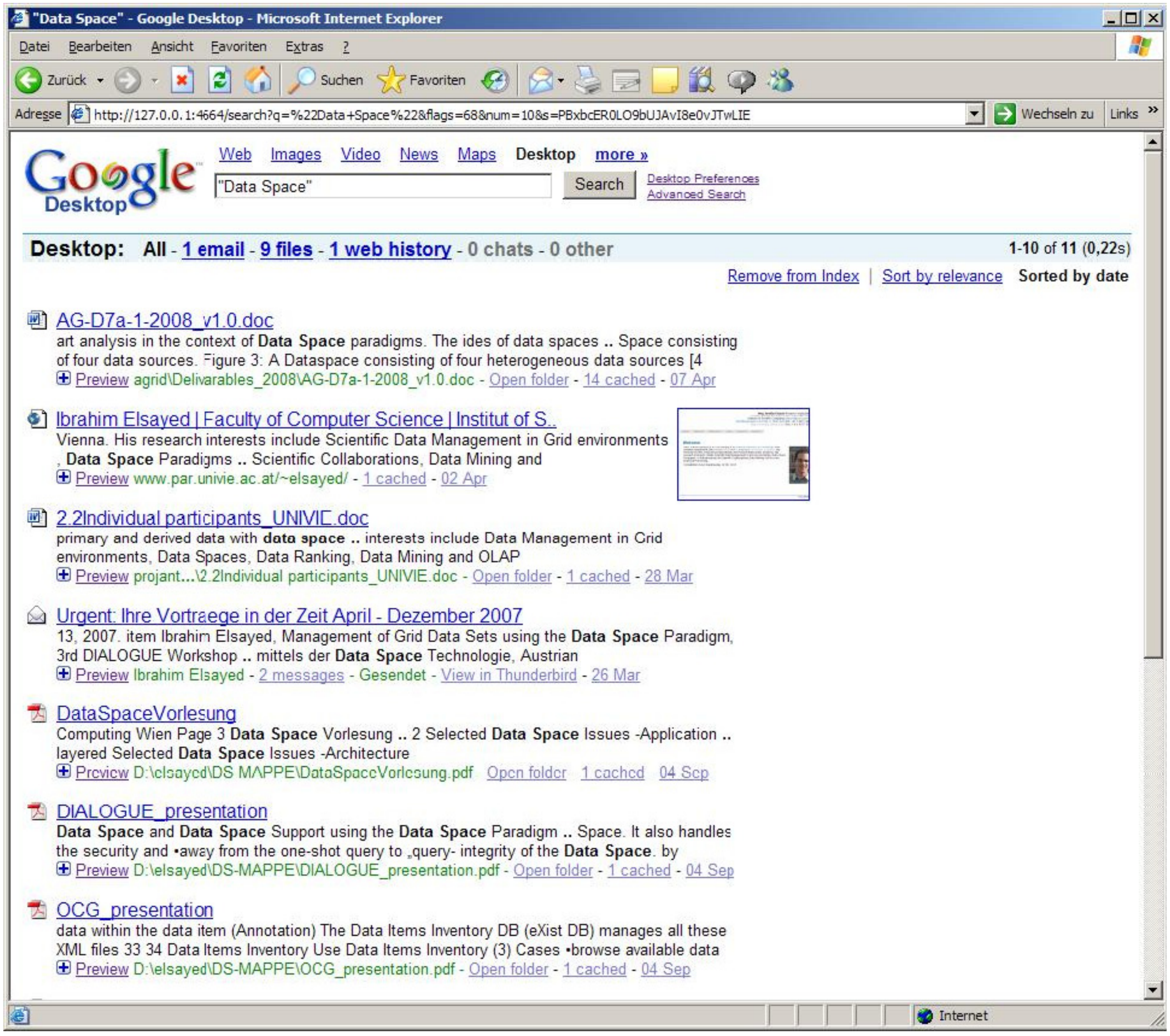
3.8.1 Introduction

Phlat is a search system for personal data providing an user interface that enables a label facilities for personal data management such as file directories, personal audio and video files, email, and more.
A basic keyword search in conjunction with a specification of particular properties of saved personal information provide effective search results. Additionally, users can define personal information metadata, which describes the stored data more in detail and helps achieve better results of desired information.

3.8.2 Design Principles

The Phlat design introduces several core principles [CRDS06].

- *"Unify text entry and filtering"* - a user can perform a query statement by using a filter, a keyword, or both.


"Data Space" - Google Desktop - Microsoft Internet Explorer
Datei Bearbeiten Ansicht Favoriten Extras ?
Zurück Suchen Favoriten
Adresse http://127.0.0.1:4664/search?q=%22Data+Space%22&flags=68&num=10&s=PBxbcER0LO9bUJAvI8e0vJTwLIE
Wechseln zu Links
Google Desktop
Web Images Video News Maps Desktop more »
"Data Space"
Search
Desktop Preferences
Advanced Search
Internet

Desktop: All - 1 email - 9 files - 1 web history - 0 chats - 0 other 1-10 of 11 (0,22s)

Remove from Index | Sort by relevance Sorted by date

AG-D7a-1-2008_v1.0.doc
art analysis in the context of **Data Space** paradigms. The ides of data spaces .. Space consisting of four data sources. Figure 3: A Dataspace consisting of four heterogeneous data sources [4
Preview agrid\Delivarables_2008\AG-D7a-1-2008_v1.0.doc - Open folder - 14 cached - 07 Apr

Ibrahim Elsayed | Faculty of Computer Science | Institut of S..
Vienna. His research interests include Scientific Data Management in Grid environments , **Data Space** Paradigms .. Scientific Collaborations, Data Mining and
Preview www.par.univie.ac.at/~elsayed/ - 1 cached - 02 Apr

2.2Individual participants_UNIVIE.doc
primary and derived data with **data space** .. interests include Data Management in Grid environments, Data Spaces, Data Ranking, Data Mining and OLAP
Preview projant...\2.2Individual participants_UNIVIE.doc - Open folder - 1 cached - 28 Mar

Urgent: Ihre Vortraege in der Zeit April - Dezember 2007
13, 2007. item Ibrahim Elsayed, Management of Grid Data Sets using the **Data Space** Paradigm, 3rd DIALOGUE Workshop .. mittels der **Data Space** Technologie, Austrian
Preview Ibrahim Elsayed - 2 messages - Gesendet - View in Thunderbird - 26 Mar

DataSpaceVorlesung
Computing Wien Page 3 **Data Space** Vorlesung .. 2 Selected **Data Space** Issues -Application .. layered Selected **Data Space** Issues -Architecture
Preview D:\elsayed\DS MAPPE\DataSpaceVorlesung.pdf Open folder 1 cached 04 Sep

DIALOGUE_presentation
Data Space and **Data Space** Support using the **Data Space** Paradigm .. Space. It also handles the security and •away from the one-shot query to „query- integrity of the **Data Space**. by
Preview D:\elsayed\DS-MAPPE\DIALOGUE_presentation.pdf - Open folder - 1 cached - 04 Sep

OCG_presentation
data within the data item (Annotation) The Data Items Inventory DB (eXist DB) manages all these XML files 33 34 Data Items Inventory Use Data Items Inventory (3) Cases •browse available data
Preview D:\elsayed\DS-MAPPE\OCG_presentation.pdf - Open folder - 1 cached - 04 Sep

Figure 3.15: The Google Desktop interface

- *"Current search criteria has to be persistently visible"* - all used filter, keywords and the order of returned search results, while performing searches, has to maintain visible for users at any time.
- *"Provide rapid query iteration"* - effective search results and sophisticated result updates should be provided.
- *"Allow iteration based on recognition"* - the applied results should provide a reuse facilities to explore the result more in detail.
- *"Allow for abstraction across property values"* - a user should be able to perform a unified query on various personal information independent of the underlying data types.
- *"User Interface has to support both tagging and filtering"* - one unified interface design should provide filter and tag facilities.
- *"Integration with file system/email operations"* - common data manipulation facilities such as copy/paste, drag and drop should be supported by the user interface.

3.8.3 Architecture

Phlat is implemented in Microsoft Visual C# and builds upon the Windows Desktop Search engine. All user personal information such as file directories, emails, audio and video files, including the Web related information such as caches are indexed. Further a declaration of user-defined metadata of the stored personal information enriches the index and provide more sophisticated search results.

User Interface The Phlat UI consists of 3 main areas [CRDS06] as illustrated in Figure 3.16:
Query Area
Query Area provides the basic information about the search and query properties, indicating a status and a quality of the results. Various property filters can be attached to any query affecting the displayed results.
Filter Area
Filter Area consists of six filters: *"Tags, Saved Queries, Path, People, Date, and Type"*. Every filter displays its property value, filtering the provided query results.

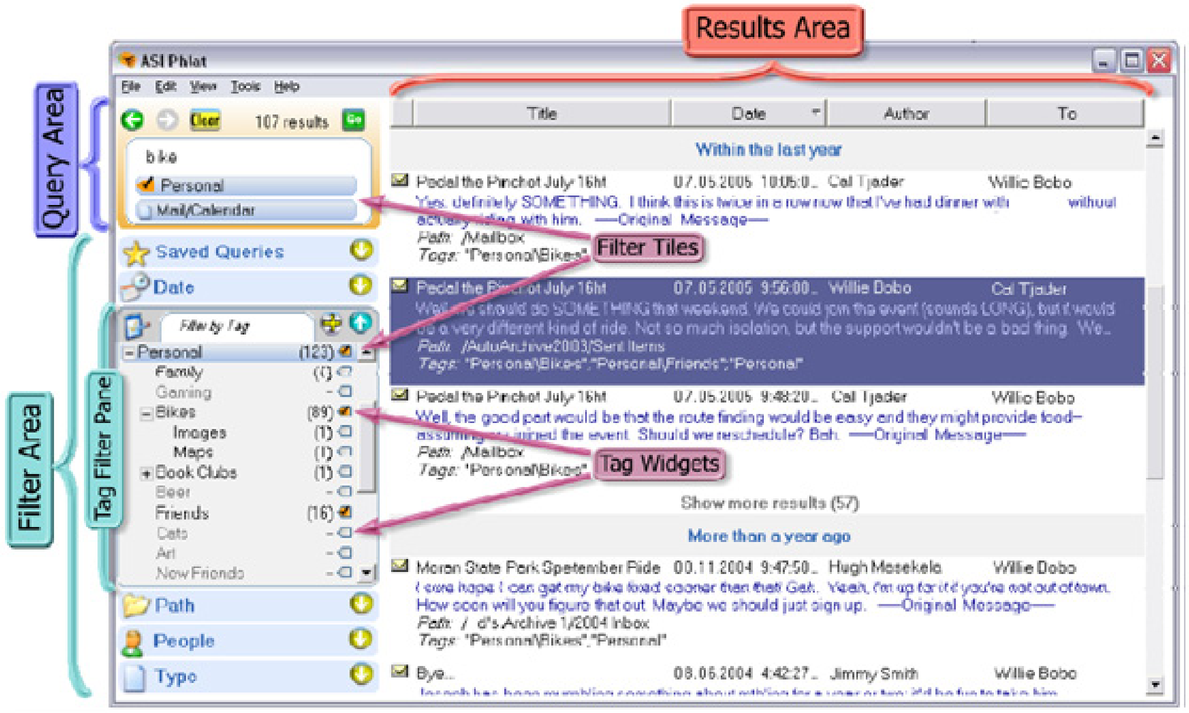


Figure 3.16: The Phlat interface

Filter area provides various types of filter abstractions, which can be associated with particular data elements more sophisticated search results.

Results Area

The result are shown as a table-like view, displaying the properties above the actual results. The results can be ordered by each property represented by a column, consisting of the query result, the path directory of shown results, and additional tags. User may browse through displayed search results and use them for defining new queries.

3.9 Chimera - A Virtual Data System

3.9.1 Introduction

Chimera is a virtual data system, which provides a workflow management tool. It mainly consists of a Virtual Data Catalog, used for derived process organization and a Virtual Data Language Interpreter (VDL), which understands user requests and is able to execute SQL statements on generated data sets. VDL is location transparent, storing the workflow related definitions in the Virtual Data Catalog. The main idea is to provide semantic information about how a data set is derived from various data sources and which operations were performed on such data, uncovering the relationships among these data sets. The virtual data system offers different data management facilities which can be executed on generated data sets, such as data replication, restore or redefine operations over already defined data items. Chimera virtual data catalog represents and manages all processes and operation as well as their characteristics applied on derived data.

3.9.2 Chimera Architecture

As already mentioned the core of chimera architecture consists of two main components, which is illustrated in Figure 3.17:

- *VDC - a Virtual Data Catalog* - *"implements the Chimera virtual data schema."* and
- *VDL - a Virtual Data Language Interpreter* - interprets user requests and application calls into operations VDC operations.

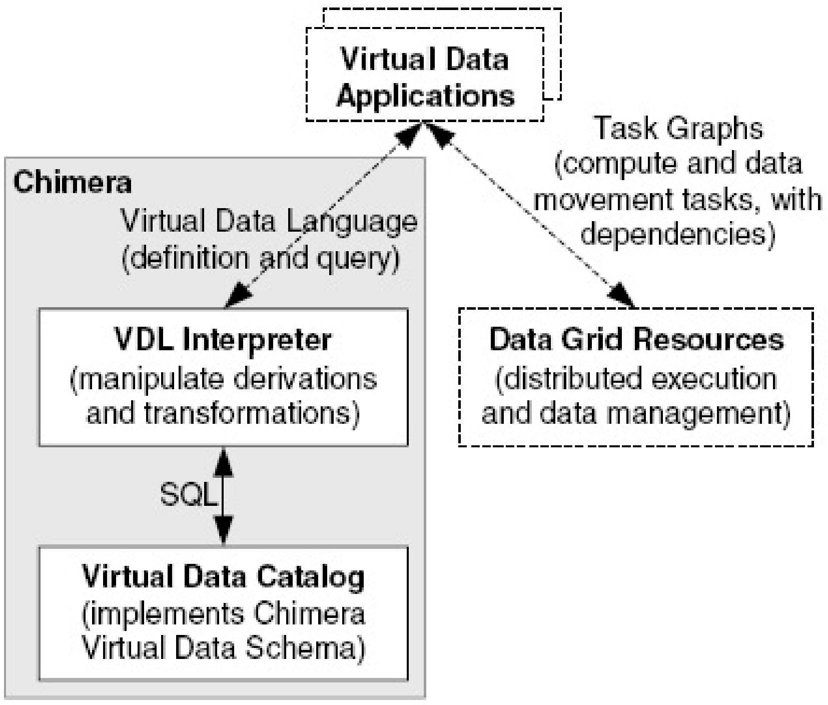

Virtual Data Applications
Task Graphs (compute and data movement tasks, with dependencies)
Chimera
Virtual Data Language (definition and query)
VDL Interpreter (manipulate derivations and transformations)
SQL
Virtual Data Catalog (implements Chimera Virtual Data Schema)
Data Grid Resources (distributed execution and data management)

Figure 3.17: Chimera architecture [FVWZ02]

Virtual Data Language acts as a interface between application calls and chimera operations. It provides facilities for common database definitions and query declarations, which can be performed over databases. Virtual data applications, as illustrated in the Figure can use Chimera information in conjunction with components implemented within a Data Grid in order to deploy application requests.

Query results can be represented in a direct graph structure specifying the virtual relations among data. The most important *"entities of interest"* in Chimera are:

- *a tranformation* - indicates a program execution information by describing the attributes of the execution process and its properties, such as a program name, location, version, etc.
- *a derivation* - indicates a transformation execution information by describing the data set information being related to performed transformations, such as data set name, execution time, property values, etc.
- *a data object* - indicates a item name generated by a derivation process, representing a *logical file name (LFN)* abstracting from a actual location of a file. Any data object is associated with a set of metadata describing the object.

The information about a derivation or transformation process can be provided by a user or generated form different assess interfaces. A logical transformation is identified by its unique name, the namespace defining the range of the name, and a number of the transformation version. It might include several derivations, representing various transformation parameter values.
The VDL defines two data derivation operations: TR (transformation), generates a object and DV (derivation), generates a invocation, which are stored in the virtual data catalog, when executed by the language interpreter.
The virtual data language builds upon SQL langugae and provides SQL query expressions over transformations and derivations by using different metadata describing logical file names, transformation and derivation names, application names, etc.

3.10 myExperiment

3.10.1 Introduction

myExperiment provides a Virtual Research Platform for workflow management in e-Science applications, allowing scientist to easy the collaborative work and exchange their knowledge more efficiently. It enables scientist to view workflows from other scientist or research groups as well as publish their own research results in form of workflows. myExperiment is implmented as a Web site and it is available at *www.myexperiment.org* [GR07]. The two main myExperiment components are users and contributors, representing the workflow objects that can be organized using myEperiment platform. Each user has access to a private section, containing a users personal information and a public section, containing the information user share with other scientists. There are several types of contributions [Cru08]:

- *Workflows* - represent the main object in myExperiment. Statistic information about workflows such as view and download count is recorded.
- *Groups* - users can create and join groups, which have title and description information. All groups are declared as public.
- *Packs* - are a set of objects or contributions.
- *files* - various files can be saved in myExperiment.
- *Experiments* - workflows are invoked within experiments. Every experiment includes a description, a name, and a an executed job, whereas each job can have a different running status.
- *Site announcements* - managed by site administrator.
- *Ownership* - contributions can be shared depending on the type definition, which can be *public* , *friend* or *group*.

myExperiment offers user authentication facilities provided by OAuth. It is generally a protocol service, which allows users to specify and register keys and attach privileges to them.

3.10.2 Basic Concepts

As already mentioned, all object in myExperiment are represented as *contributions*, whereas all relevant metadata is recorded by the system. The object resources itself are also known as *contributables*, such as files, workflows and packs. myExperiment distinguishes between two different levels of object abstraction: *high-level* (some of the contribution metadata is not displayed) and *low-level abstraction* (all recorded metadata can be viewed). It further defines following structural constraints for [Ale08]:

- *contributions* - represents an object and may have one contributable abtraction, providing following resource details: contributor id and type, date, attached permissions, and number of views and downloads.
- *contributables* - unique abstraction name of one contribution. only files, workflows and packs may appear as contributables.

- *contributors* - can own many contributions, and specify permissions and policy definitions.

Contribution tables consists of *types* and *ids* while using *contribution ids* to specify relations between contributables and contributors.
myExperiment also supports resource versioning, recording all resource versions, which indicates how a particular resource (current myExperiment implementation allows only workflow versioning) was deployed over a period of time.
Figure 3.18 illustrates the myExperiment UI. The main page of the Web site provides

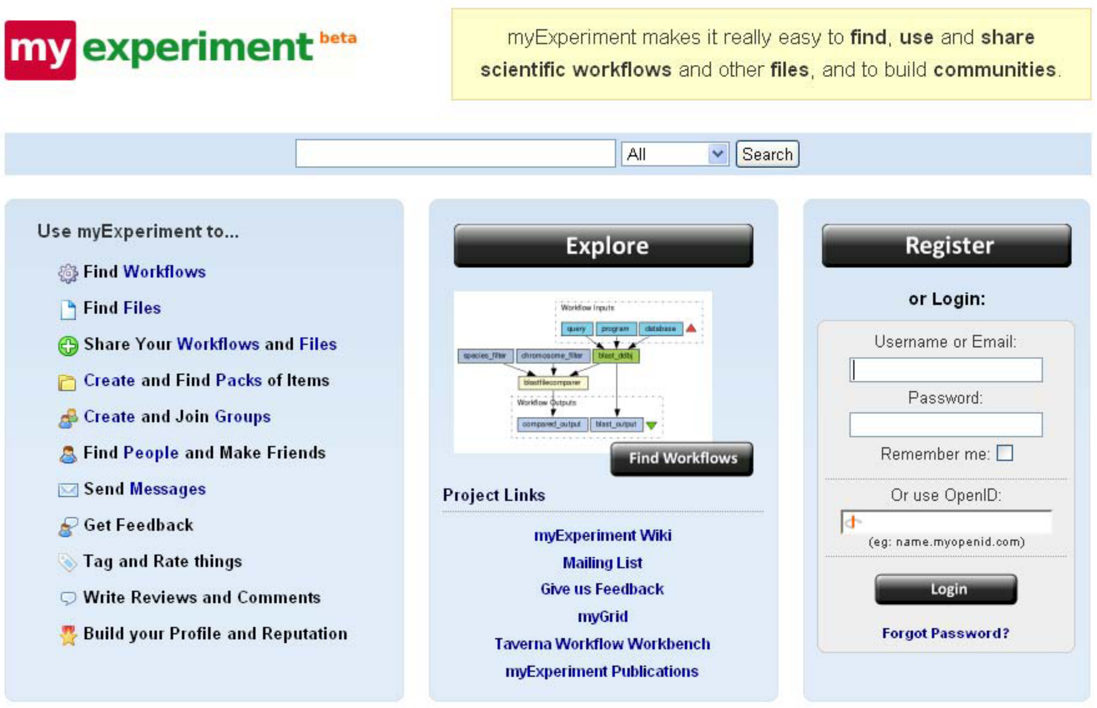

myExperiment makes it really easy to **find**, **use** and **share** **scientific workflows** and other **files**, and to build **communities**.

Figure 3.18: The myExperiment UI [oMoS08]

basic information on which facilities myExeriment offers to its users. In addition, further project and documentation information can be explored.

3.11 Conclusions

So far dataspace paradigms have been mainly considered in terms of personal information management. In [FHM05] the concepts of dataspaces are introduced in a visionary way providing the most important key concepts designing e-Science Life Cycle Ontology with regards to data management of a heterogeneous data sources represented as participants in a dataspace while being described by an intelligent relationship model.

Influenced by this vision a personal dataspace management system with an own data model, abstracting from underlying substructures of data sources, and a new query language is presented in [BDG$^+$07]. However, this system is limited to personal information management only, handling data of an individual while not considering dataspace management facilities for large scale proposes.
Other projects like iRods [RWMS06], SRB [BMRW98], and the IBM's commercial product Websphere Information Integrator [BAB03] have considered some dataspace concepts such as a metadata catalog and a logical name abstraction concepts in their architecture; however the key dataspace paradigm, which is to provide semantically rich relationships among heterogeneous data elements represented as participants, is not taken into consideration. A first approach towards realization of dataspaces regarding the Grid is given in [EBT06].
Chimera [FVWZ02] provides a virtual data system, which offers a workflow management tool. The main idea is to provide semantic information about how a data set is derived from various data sources and which operations were performed on such data, uncovering the relationships among these data sets. This approach, compared to the e-Science Life Cycle Data Model, is more data set oriented managing the workflow information, while the Life Cycle View aims on describing the entire relationship information while performing scientific applications, including information on researchers, research goal specifications, data preparation tasks, data analysis tasks, and produced results which can be published for further discovery and experimentation.
myExperiment [GR07] provides a Virtual Research Platform for workflow management in e-Science applications, allowing scientist to easy the collaborative work and exchange their knowledge more efficiently. It enables scientist to view workflows from other scientist or research groups as well as publish their own research results. However, the main work here is focused on knowledge exchange in form of workflows with no regards of tracking down background data source information associated with scientific experiments.
Both systems, Chimera and myExperiment are targeting to model relationships from primary and derived data through collecting provenance data of executed workflows. However, dataspace search and query features are not tightly focused.

3.12 Summary

The following table shows a comparison matrix of the above described systems and tools with regard to key-Dataspace Paradigms:

DS Pradigms	iMemex	SRB	iRODS	IBMII	GDS	Phlat	Chi	myExp
Semantic Integration	x							
advanced querying	x							x
enriching keyword	x							
property search	x							
Semantic relationships	x							
Browse by relationship								
Update Mechanism	x	x	x	x	x	x	x	x
full control of the data				x	x	x	x	x
Automatically updates								
Schema first								
no schema								
Ranking query results	x				x			
RDF								
OWL								
Meta data Catalog	x	x	x	x			x	x
Own data model	x							
keyword search	x	x	x		x	x	x	x
large scale Dataspaces	x	x	x	x				
security issues	x	x	x	x	x		x	x

Table 3.1: Dataspace paradigms comparison matrix

DS Pradigms	**iMemex**	**SRB**	**iRODS**	**IBMII**	**GDS**	**Phlat**	**Chi**	**myExp**
Managing sub-Dataspaces								
Role management	x	x	x	x				
User groups	x	x	x	x			x	x
Registration wizard							x	x
Data Statistics								
Learning features								
Autonomic features								
Own query language	x				x	x		

Table 3.1: Dataspace paradigms comparison matrix

Chapter 4

e-Science Life Cycle Data Model

4.1 Introduction

A great challenge currently faced by the data management community is to rise up the level of abstraction at which data is managed. Today we have well integrated data base management systems such as the widely used RDMS, which offer rich data management features for a single data source. However there is a lack of something when it comes to the need of managing different heterogeneous and distributed data sources.

The concept of Dataspaces visionary introduced by Franklin et al. [HFM06] gave

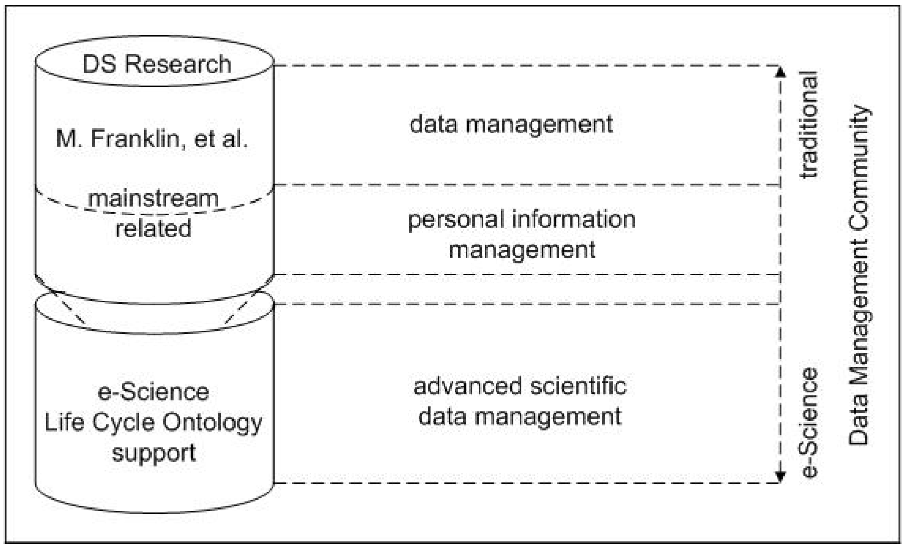


Figure 4.1: Data management community - dataspace research

rise to new data management challenges and influenced various database-oriented data management and personal data management approaches, for example such as

iMemex [BDG$^+$07], however is so far not considered for data management in advanced scientific applications.
The e-Science Life Cycle Ontology can be seen as an application range extension of the common data management applications by introducing the dataspace concepts in the context of e-Science as illustrated in Figure 4.1. It provides an ontology based dataspace model for data being collected in scientific collaborations uncovering the relationships among primary and derived data in scientific experiments.

In order to elaborate how dataspace concepts can support e-Science, we have investigated what happen, or better what should ideally happen to data in e-Science applications. The result of this investigation is an iterative and hierarchical metamodel with five main activities, represented in Figure 4.2, which is defined as following: *The e-Science life cycle - a domain independent ontology-based iterative metamodel, tracing semantics about procedures in e-Science applications. It consists of five main activities grouped into iterations and organized as instances of the e-Science life cycle ontology, which are feeding a dataspace, allowing it to evolve and grow into a valuable, intelligent, and semantically rich space of scientific data.* First we provide an overview of these activities and then in Section 4.2 a more detailed discussion.

At the beginning of the life cycle targeted goals are specified, followed that a data preparation step including pre-processing and integration tasks is fulfilled. Further appropriate data analysis tasks are selected and applied on the prepared dataset of the previous step. Finally achieved results are processed and published, which might provoke further experimentation and consequentially specification of new goals within the next iteration of the life cycle. The outcome of this is a space of primary and derived data with semantically rich relationships among each other providing (a) easy determining of what data exists and where it resides, (b) searching the dataspace for answers to specific questions, (c) discovering interesting new data sets and patterns, and (d) assisted and automated publishing of primary and derived data.

Each activity in the life cycle shown in Figure 4.2 includes a number of tasks that again can contain a couple of subtasks. For instance, the activity *Prepare Data* covers, on a lower level of abstraction, a data integration task gathering data from multiple heterogeneous data resources that are participating within an e-Infrastructure. This task consists of several steps that are organized into a workflow, which again is represented at different levels of abstraction - from a graphical high level abstraction representation down to a more detailed specific workflow language representation, which is further used to enact the workflow.

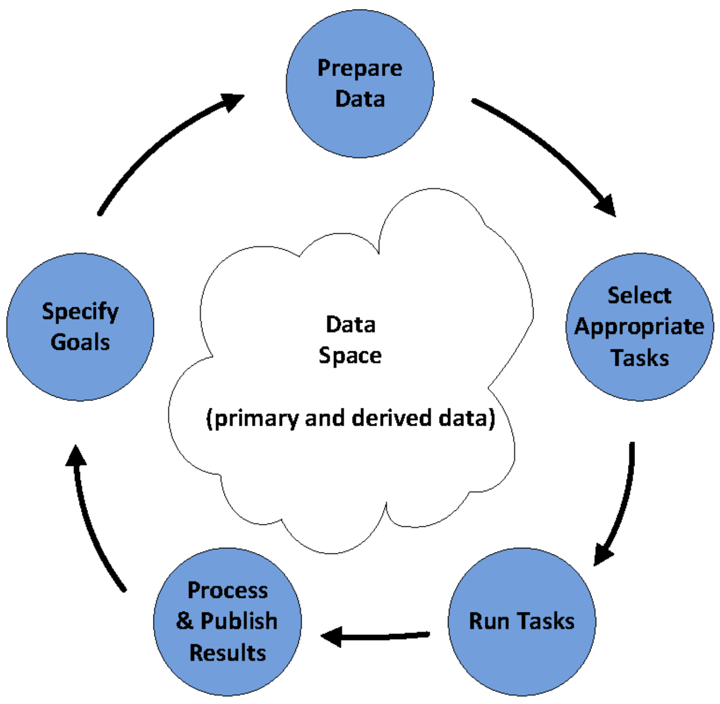


Figure 4.2: The e-Science life cycles

4.2 e-Science Life Cycle Activities

1 *Specify Goals* - Scientists specify their research goals for a concrete experiment, which is one iteration of the entire life cycle. This is the starting activity in the life cycle. A textual description of the objectives, user name, corresponding user group, research domain and other optional fields like a selection of and/or references to an ontology representing the concrete domain is organized by this activity.

2 *Prepare Data* - Once the objectives for this life cycle are either specified or selected from a published life cycle that was executed in the past, the life cycle goes on with the data preparation activity. Here it is specified which data sources are used in this life cycle in order to produce the final input dataset, by the data integration process. For example, the resource URI, name, and a reference to the OGSA-DAI [M. 07] is the de facto standard for data access and integration for relational and xml data as well as file resources. Resource File is recorded. The final dataset as well as the input data sets are

acting as participants in the dataspace and are referenced with an unique id. Additionally, the user specifies a short textual description and optionally some keywords of the produced data set.

3 *Select Appropriate Tasks* - In this activity the data analysis tasks and to be applied on the prepared dataset are selected. In e-Science applications it is mostly the case that various analytical tasks, for instance the widely used data mining techniques, are executed successively. The selected tasks, which are available as Web and Grid services, are organized into workflows. For each service, its name and optionally a reference to an ontology describing the service more precisely is captured. Also for the created workflow, its name, a short textual description, and a reference to the document specifying the workflow are recorded.

4 *Run Tasks* - In this activity the composed workflow will be started, monitored and executed. A report showing a brief summary of the executed services and their output is produced. The output of the analytical services used is represented in the Predictive Model Markup Language (PMML) [Dat08] which is a standard for representing statistical and data mining models. PMML documents represent derived data sets, thus they are managed as participants of the scientific dataspace and considered as resources by this activity.

5 *Process and Publish Results* - This is the most important activity in order to allow the underlying dataspace to evolve and grow into a valuable, powerful, semantically rich space of scientific data. Based on the settings of the user, one automatically publishes the results of the data mining tasks, represented in PMML as well as all semantical information captured in the previous activities. Different publishing modes allow to restrict access to selected collaborations, user groups, or research domains.

4.3 Life Cycle Metamodel

Ontological knowledge is sharable, understandable to machines, and supports the enrichment of data sources and relationships at the semantic level. Therefore we have developed the *e-Science life cycle ontology*, which organizes the concepts and coherences of the above described e-Science life cycle activities. Strong regard was put on considering input (primary) and output (derived) data sets as well as relevant background data (e.g. domain ontologies, data statistics, OGSA-DAI resource files, workflow descriptions, etc.) for modeling an intelligent relationship paradigm.

At first, supported by the ontology, a metamodel independent from the various e-Science domains is set up. Then this metamodel is applied to describe domain-specific iterations of the e-Science life cycle, which describe the relationship among data participating within the scientific dataspace, illustrated as different abstraction layers in Figure 4.3.

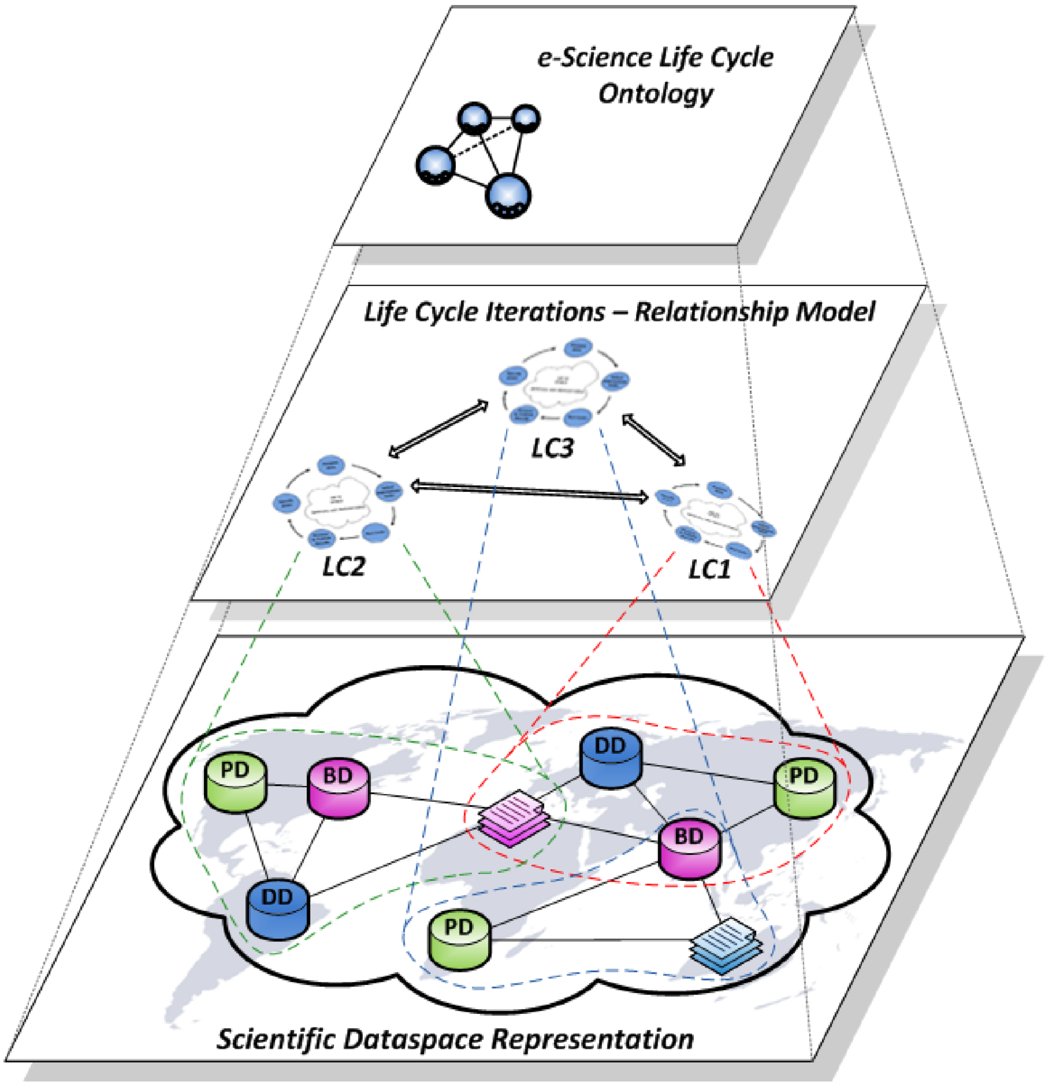

e-Science Life Cycle Ontology
Life Cycle Iterations – Relationship Model
LC3
LC2
LC1
PD
BD
DD
DD
PD
BD
DD
PD
Scientific Dataspace Representation

Figure 4.3: Abstraction layers of scientific dataspaces *(PD - primary data, DD - derived data, BD - background data)*

One iteration of the e-Science life cycle has, in short, a goal specification, a set of input data (primary data), a set of output data (derived data), a set of background data, and a set of activities describing what has been done to the input data sets in order to produce the output data sets. These data sets are populating the scientific dataspace, enriched with semantic relationships among each other, described by its corresponding life cycle iteration. We can see from this, that the dataspace is evolving with an increasing number of life cycles.

This profound knowledge about iterations of the e-Science life cycle, consolidated within instances of the ontology represents an intelligent relationship model for scientific dataspaces, because it provides (a) creation, (b) representation, and (c) searching of semantically rich relationships among dataspace participants. Realization of a scientific dataspace paradigm will highly contribute to the development of high productivity e-Science frameworks.
With the help of the e-Science life cycle ontology, it is made possible for scientists to describe, execute and share their e-Science experiments with others in an efficient manner. Further, it is feasible to search for published instances of the life cycle or even for instances of single activities of the life cycle. In such a way, a scientist could search for all published goal specifications corresponding to his research domain, by searching for a given domain name. The dataspace will then provide not only the published instances of the activity, but also the complete instance of the e-Science life cycle, including the inputs of other activities and its corresponding results. In addition, it will give hints about similar life cycle iterations by using the semantically rich relationships described by the ontology.

With this in mind, it will be easier for research groups to engage collaboration, provide knowledge transfers within collaborations and among different research groups with respect to different research areas. In conclusion, the e-Science life cycle meta-model is likely to unify the process of publishing primary, derived, and background data sets as well as the their interconnection and make it easy for scientists to register, describe and execute new e-Science experiments and for users to find, explore and understand these applied experiments. The e-Science life cycle ontology is available at http://www.gridminer.org/e-sciencelifecycle/.

4.4 Environment of the e-Science Life Cycle

Scientific dataspaces will be set up to serve a special subject, which is on one hand to semantically enrich the relationship of primary and derived data in e-Science applications and on the other hand to integrate e-Science understandings into iterations of the life cycle model allowing scientists to understand the objectives of applied e-Science life cycles. Figure 4.4 shows the environment of e-Science life cycle. In particular there is a set of participants participating to one ore more activities of the e-Science life cycle.

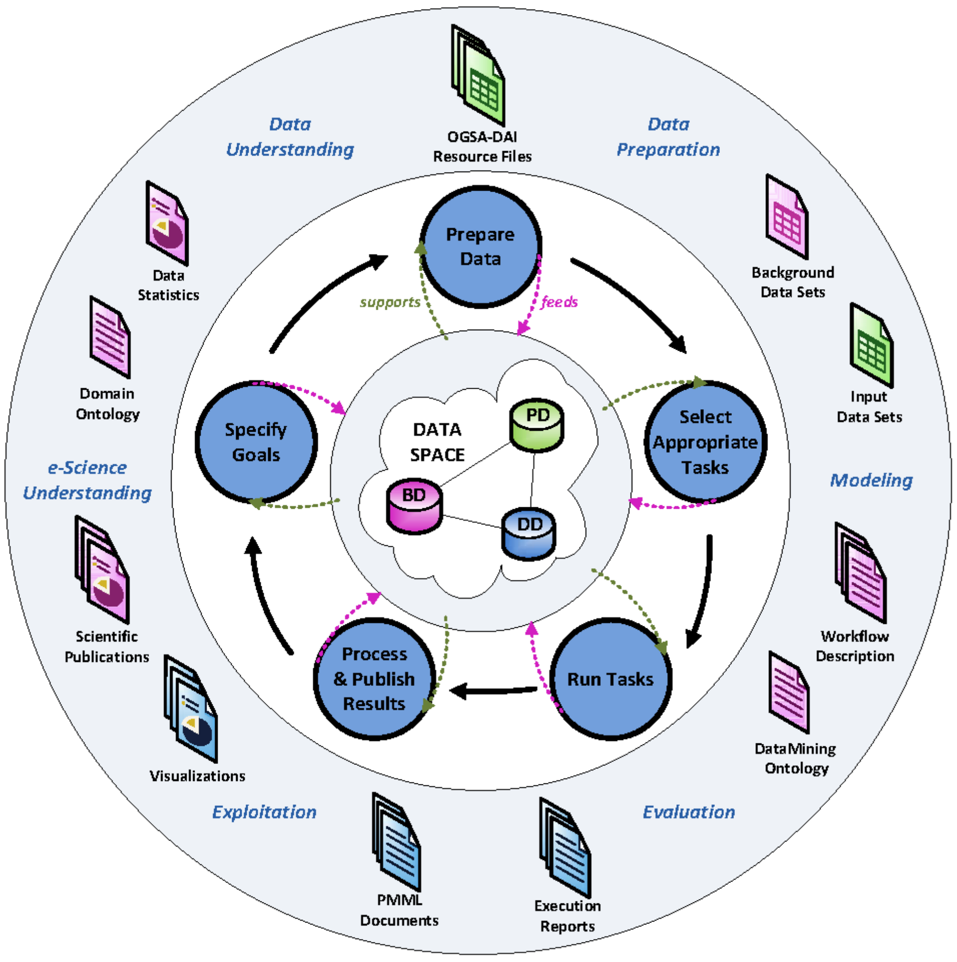


Figure 4.4: Environment of the e-Science life cycle

Each activity feeds the dataspace with new participants, as for example the activity *Specify Goals* adds new domain ontologies, the activity *Prepare Data* adds new final input data sets as well as OGSA-DAI resource files, and the activity *Select Appropriate Tasks* adds new workflow description documents, while the activity *Run Tasks* adds new PMML documents describing the data mining model applied, and finally the activity *Process and Publish Results* adds new documents visualizing the achieved data mining outputs. All these participants belong to at least one or more e-Science life cycles, expressed as instances of the ontology describing its relationship and interconnection to a great extend.

Each iteration of the life cycle metamodel will produce a new instance of the ontology. Based on the publishing mode, set by the scientist who accomplished the life cycle, the whole instance will automatically be published into the dataspace and thus is available to other users of a wider collaboration with respect to other research areas. We distinguish between four publication modes, (1) *free access*, (2) *research domain*, (3) *collaboration*, and (4) *research group*. Users will have access to sets of participants available in the scientific dataspace, depending on their assigned role. By this, the concept of managing sub-dataspaces is realized. A sub-dataspace contains a subset of participants and a subset of relationships of the overall dataspace. There can be sub-dataspaces setup for different domains, then for different research collaborations and even for single research groups. e-Science experiments that were published using the *free access mode*, will participate in the overall dataspace, thus its participants and the life cycle instances are accessible for every one having access to the scientific dataspace. In order to access data of a specific life cycle iteration, that was published using the *research group mode*, it will be necessary to be member of that specific research group, as the data will be participating only in the corresponding sub-dataspace.

4.5 Search and Query Scientific Dataspaces

Based on our unified e-Science metamodel, search and query services can be provided for all the participants of the scientific dataspace. Hence, it is possible to forward a keyword query to all participants, which has the aim to identify relevant data sets. However, each query submitted to the scientific dataspace, will receive not only the matching data but also data of its followed e-Science activities. For instance it will be possible to receive what mining task were applied on a discovered dataset, the concrete workflow, the workflow report, the results presented in PMML and its corresponding visualizations.

Using *SPARQL* query language for RDF [PS08] and semantically rich described e-Science life cycles, consolidated within instances of the ontology, keeping relationships among each other, the dataspace is able to provide answers to specific questions, such as the following:

A *"I have found some interesting data, but I need to know exactly what corrections were applied before I can trust it."*

B *"I have detected a model error and want to know which derived data products need to be recomputed."*

C *"I want to apply a NIGM-analysis on meridian HE GU. If the results already exist, I'll save hours of computation."*

D *"Is there any experiment done on meridian BA XIE"*

Through portals and advanced user interface scientists are supported with the needed tools, which enable users to express search queries visually and in an simple way. The output is simply *SPARQL*, which allows to query instances of an ontology efficiently. This is part of our ongoing work, currently under investigation. However, the basis for intelligent dataspaces for e-Science is developed and have cleared the way towards developing high-productivity e-Science frameworks.

4.6 GridMiner Knowledge Discovery System enhanced by the e-Science Life Cycle Model

GridMiner [TBW$^+$08] is a knowledge discovery system on the grid, which was developed at the University of Vienna. Several different services, integrated within the GridMiner architecture, such as data integration, data selection, data transformation, data mining, pattern evolution and knowledge presentation interact together providing an service-oriented grid application.
The e-Science Life Cycle Ontology can be applied on this knowledge discovery process creating semantic relationships among participating data elements representing the primary and derived data in e-Science applications and integrating e-Science understandings into iterations of the life cycle model.
Knowledge discovery in databases is an interactive process which provides useful and understandable pattern identification of data. GridMiner project defines several different phases of knowledge discovery organized as service-oriented scientific workflows [TBJ08]. Figure 4.5 illustrates the GridMiner knowledge discovery reference model enhanced by the e-Science Life Cycle components. In the following we will discuss how the particular knowledge discovery phases can be supported by the e-Science life Cycle Model.

Scientific problem identification
The Scientists identifies in this initial phase the main goals and requirements for their research problems. In the *Specify Goals* activity of the life cycle model we capture all goal and objective specification related information provided by the user.
Data quality understanding
This phase is concerned with data quality identification issues, introducing the relevant data sets to the user. The interesting data subsets, data statistics and possible

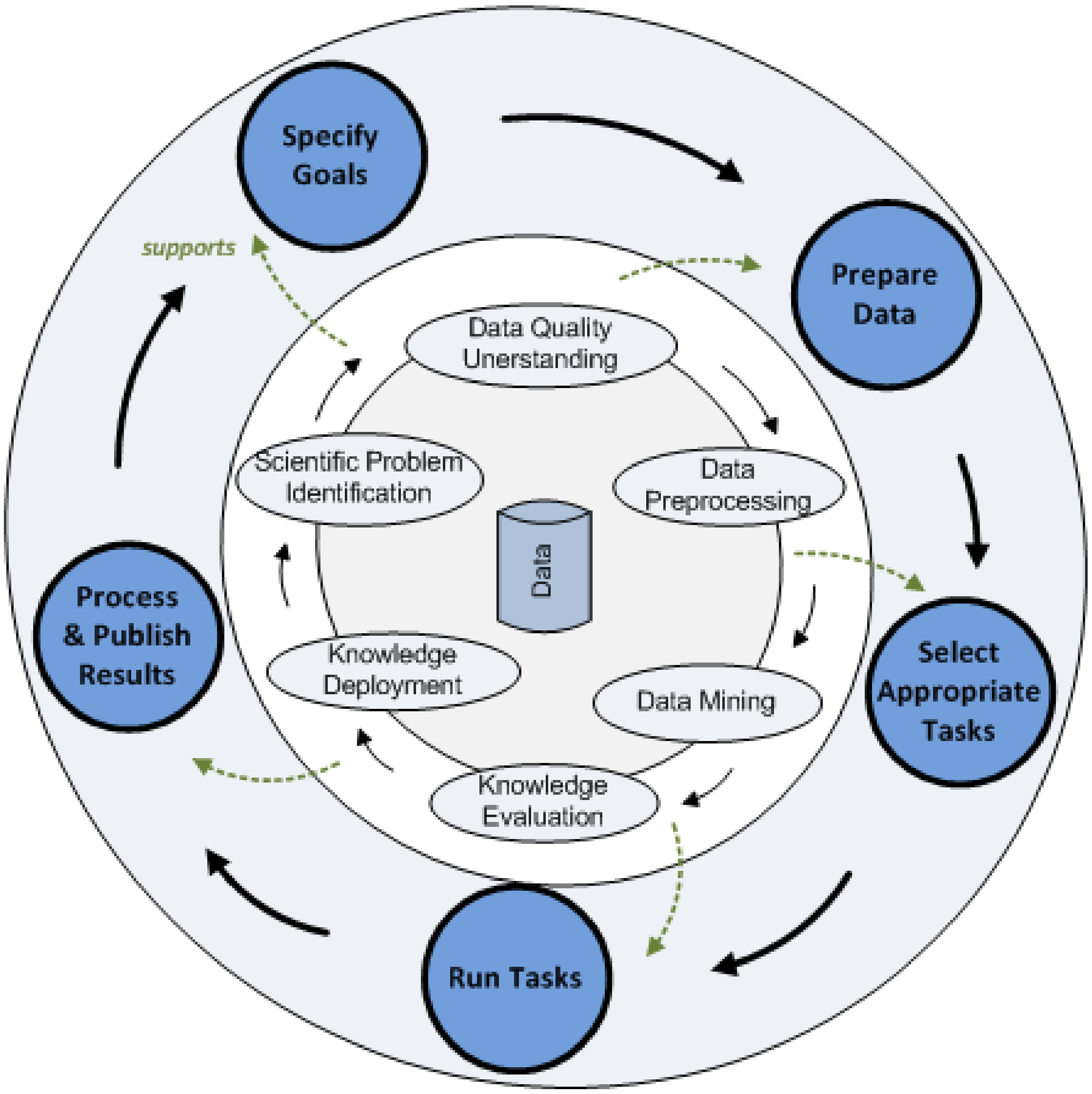


Figure 4.5: GridMiner knowledge discovery system enhanced by the life cycle model

quality problem reports can be organized as background data of the life cycle dataspace model.

Data pre-processing

Data pre-processing represent the main phase of knowledge discovery process. The main goal here is to clean the data producing an improved data set which is recorded by the *Prepare Data* life cycle activity, while also specifying which data sources are used in order to produce the final input dataset of the data integration. For more information about which data source relevant information is recorded in detail we refer to Section 4.2

Data Mining

This phase is concerned with data analysis tasks applied on the prepared dataset, which are organized as workflows. Various data mining methods such as association rules, classification or regression are executed in order to analyze the prepared data sets. The valuable workflow and data mining related information is organized by the *Prepare Data* life cycle activity.

Evaluation

This phase deals with representation of the discovered information. Various visualization techniques are used for representing the results which act as participants in a dataspace. This information is managed by the *Run Task* activity and is referred as the derived data of the life cycle ontology. The output of the analytical services used is represented in the Predictive Model Markup Language (PMML) [Dat08] which is a standard for representing statistical and data mining models.

Knowledge deployment

Idea of analytical data mining processes is to increase the knowledge and further share the achieved results. Finally achieved results are processed and published, which might provoke further experimentation and consequentially specification of new goals within the next iteration of the life cycle.

The following Figure 4.6 illustrates a high-level architecture model of the e-Science Life Cycle Ontology interacting with the GridMiner system. The architecture consists of two main components: the GridMiner System implementing a number of services and the e-Science Ontology which is integrated into a Dataspace Framework. Some of the Ontology activities are supported by the GridMiner Services as described above. A DataSpace UI provides a user environment allowing a user to communicate with a dataspace.

GridMiner implements a number of various services and interfaces which are used in

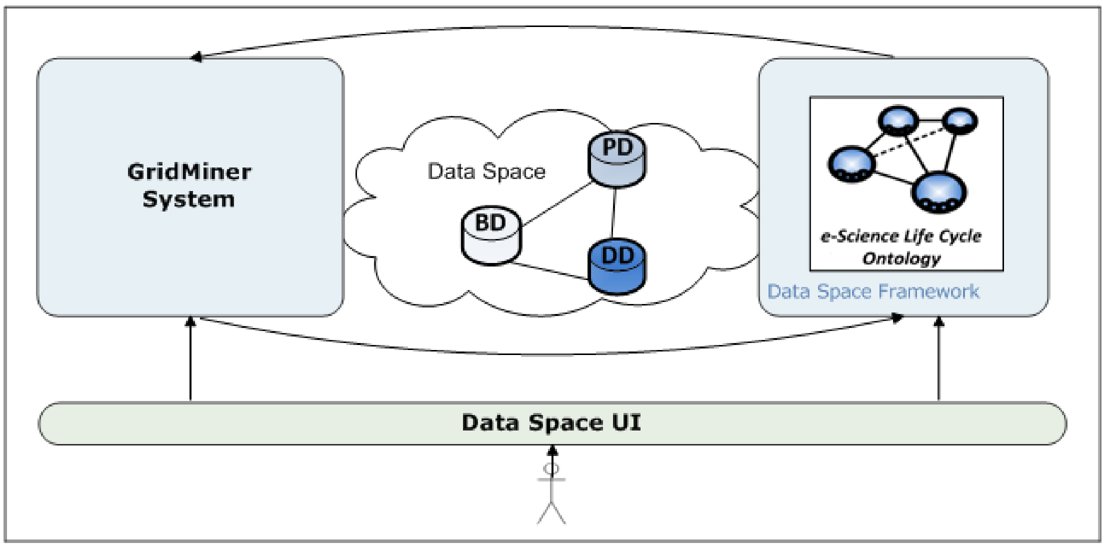


Figure 4.6: High-level e-Science ontology architecture model interacting with the GridMiner system

the Knowledge Discovery Process. Figure 4.7 provides an overview of these services enhanced by the Dataspace components [KHB04], which are briefly describe below:

GridMiner Dynamic Service Control (GMDSCE) This service is concerned with workflow execution for a particular knowledge discovery process.

GridMiner Mediation Service (GMMS) This service provides a unified virtual data source by integrating heterogeneous distributed data sources.

GridMiner Information Service (GMIS) Information services provide information on resources which can be found within a grid system.

GridMiner Resource Broker (GMRB) Resource Broker is responsible for request and resource matchmaking.

GridMiner OLAP/Cube Service (GMCMS) This service generates OLAP Cubes from various data sources.

GridMiner Transformation Service (GMDT) Certain data has to be transformed before appropriate data mining techniques can be applied on it, which is the main task of this service.

GridMiner Preprocessing Service (GMPPS) These services deals with task which are executed before the main data mining processes.

GridMiner DataMining Service (GMDMS) This service represents the main component of the GridMiner system and provides information about all data mining methods and techniques.

GridMiner OLAP Mining Service (GMOMS) OLAP Mining service facilitates data mining services which can be applied on cubes.

Dataspace Components

Dataspace Participants. Dataspace consists of primary, derived and background data, which act as participants of a dataspace.
Dataspace Access Service. Dataspace Access Service manages the user and participant information regarding the dataspace access.
Dataspace Registration Service. This service provides user and participant registration information.
Dataspace Catalog Service Catalog service summarize a dataspace participant meta information. It provides the entire information about all the data sources acting as participants in a dataspace.
Dataspace Relationship Service. This Service manages the information how dataspace participants are related to each other while keeping track of all participant relationships management information, monitoring the evolution of the participant relationships and their creators.
Life Cycle Ontology Repository Service. The Ontology Repository records all life cycle related information that is generated while executing life cycle iterations like life cycle ids, corresponding activities, involved scientists, produced outputs, etc.
Dataspace Presentation Service. The results of scientific experiments performed as life cycle iterations are presented in a dataspace allowing a user to explore the content of published information.
Dataspace SPARQL Service. This service allows SPARQL query execution on executed life cycle iterations organized as ontology instances.
Dataspace Publication Service. The life cycle iteration results and its corresponding participant information is published by a certain publication mode, which is covered by the Publication Service.

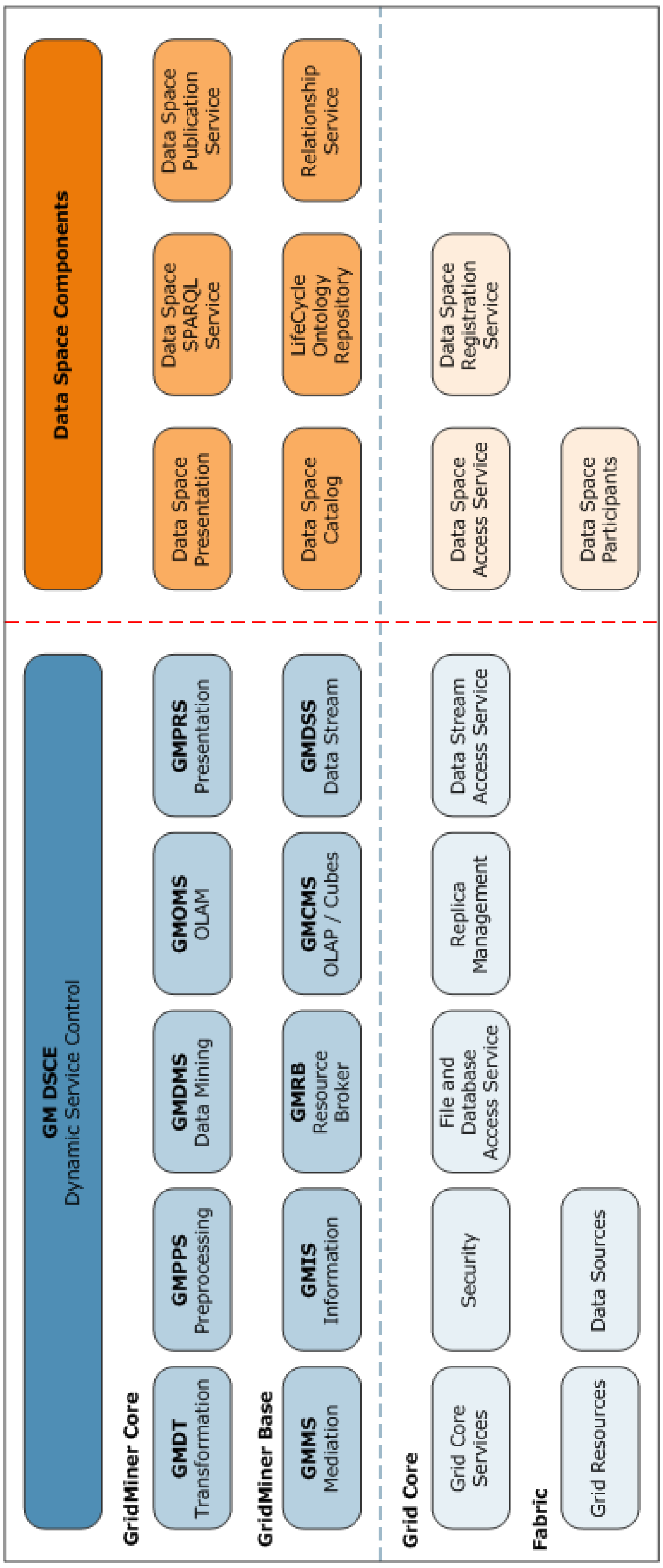


Figure 4.7: The GridMiner components enhanced by the dataspace components

Chapter 5

e-Science Life Cycle Implementation

5.1 Introduction

The most important issue building the *Life Cycle Ontology* was to capture and uncover the e-Science Life Cycle knowledge by identifying the key concepts and relationships described in the previous chapter. The identification of general abstract terms related to defined Life Cycle concepts such as the participants including primary data, background data, and derived data, then life cycle activities, research domain and researcher, was a initial point in defining the classes and their relations to each other as main ontology concepts. The more specific concepts describing for example the life cycle activities more in detail such as data preparation activity, which background data it uses, and what kind of output data it produces, were identified next. Required semantic relationships between identified concepts were declared providing a higher complexity level while appropriate notation indicating the relation role between classes was identified. There was great emphasis on defining authentication concepts based on a publish mode of a particular life cycle provided by a scientist who executed the corresponding life cycle. While going more into detail defining specific ontology concepts, a certain rework was occasionally required in order to guarantee the consistency of the ontology. Finally, ontology use case scenarios and applications were applied to identify further classes and relations providing more knowledge about the ontology. The ontology was created using Protege OWL Plugin 3.3.1 [fBIR08].

5.2 Methodologies for Ontology Creation

The goal of this section is to present the main methodologies for ontology creation. A methodology is defined by IEEE [IEE08] as *"a comprehensive, integrated series of techniques or methods creating a general system theory of how a class of though-intensive work ought be performed"* [GPFLC04]. The ontology development process deals with identifying the most crucial activities which are performed while creating ontologies. There are three main Types of activities [GPFLC04]:

- *Ontology management activities* - task identification, arrangement, and execution time is managed by this activity including a control and quality facilities for performed tasks.
- *Ontology development oriented activities* - describes the most important issues considering ontology development such as the ontology environment, referring to how and where the ontology will be used, the ontology specification, conceptualization, formalization and implementation, addressing the main development steps, and finally the ontology application.
- *Ontology support activities* - includes a knowledge discovery, evaluation and documentation process while building ontologies.

In the following we present some of the most important methodologies building ontologies:

Unschold and King's method
In 1995 Unshold and King introduced the first method for creating ontologies. The main process includes four building steps, which are:

- *Purpose identification* - specification of the relevant application domain terms, goals, ontology purpose and user identification.
- *Ontology building process* - key concepts identification and term relationships of the specified domain, textual descriptions and definitions of classes and relationships is provided. We distinguish here between three different concept strategies in order to build an ontology: (1) *the bottom-up strategy* - first the specific concepts such as a *dataSet* or *workflowDocumenet* are identified and then a more general abstraction is modeled grouping the concepts for example into *derivedData* participants of a dataspace. As a result we receive a more detailed concepts while increasing a inconsistency risk while requiring more rework. (2) *the top-down strategy* - first an abstract concept is modeled and

then a specification of this model. This results in a less consistent model, requiring greater rework and more effort. (3) *the middle-out strategy* - this approach specifies the core concepts and basic classes first which for example are represented trough different participant categories such as *backgroundData*, *primaryData* and *derivedData*, Then, we specify the concepts on the top such as *participant* and concepts on the bottom such as *workflowDocument* representing a particular derived data output document.

- *Evaluation* - the ontology is evaluated using appropriate application environments.
- *Documentation* - the notation, concepts and their relationships should are documented.

Grüninger and Fox's Methodology

in 1995 Grünninger and Fox proposed a formal building and evaluation concept designing ontologies. The core methodology processes are: (1) *scenario identification* - the ontology development is application scenario related, providing solutions and a formal knowledge model of the classes and relationships which will be used while building ontology. (2) *informal competency question elaboration* - represent the informal questions, expressed in natural language, that should be answered using the implemented ontology. (3) *terminology specification* - using the extracted content and knowledge of a ontology, one can identify the terminology indicating the concepts and their relations. (4) *formal specification of competency questions using formal terminology* - the informal questions, expressed in natural language, are formally represented. (5) *axioms specification using first order logic* - term definition of a ontology using axioms, and (6) *completeness theorem specification* - condition definition providing the complete answers to the competency questions.

METHONTOLOGY

Methontology was introduced in 1997 by the Ontology Group at Universidad Politecnia de Madrid, allowing building knowledge level ontologies. The core task processes while creating ontologies using Methonology are: (1) *glossary of terms specification* - term definition of the relevant domain of interest is specified, including concepts, instances, properties representing the relationships among the concepts, textual descriptions, synonyms, etc. (2) *concept taxonomy specification* - after the terms are identified, a concept taxonomy is specified providing a hierarchy model definition. Each of the in *Unschold and King's method* introduced strategies (top-down, bottom-up, middle-out) can be applied here. (3) *ad hoc binary relation diagram specification* - the relationships among concepts are defined specifying the domains and ranges of each relation. (4) *the concept dictionary specification* - property and relationship specification describing the previous generated taxonomy concepts, including all domain related concepts, their instances and relationships. (5) *detailed specification of the ad hoc binary relations* - detailed description of the concept dictionary in form of a relation table representing the Object Properties. (6) *detailed attribute specification* - all attributes specified in the concept dictionary are described in detail, including the name, the value types, domain and ranges, representing the Data Type Properties. (7) *detailed class attribute definition* - all class attributes specified in the concept dictionary are described related to the class they belong to. (8) *detailed constants definition* - each constant specified in the term glossary is described. (9)*formal axiom definition* - formal axiom table, including the logical expressions, description, name, corresponding concepts and attributes, is generated. (10) *rule definition* - ontology rule definition, including the name, description, expression describing the rule, concepts and relations. (11) *instance definition* - instance specification, including the name, the concept the instance belong to and the attribute values.

On-To-Knowledge

The main goal of the *On-To Knowledge* approach is concerned with knowledge management improvement in large and distributed areas. Ontologies created using On-To-Knowledge are strongly application dependent, specifying the ontology goals with respect to usage application. Therefor this particular methodology was used to develop the e-Science Life Cycle Ontology.

In the following we are going to illustrate and describe the main On-To-knowledge methodology processes applied on the e-Science Life Cycle Ontology:

- *Feasibility study* - concept investigation and elaboration process describing the model and application usage. Goals specification, e-Science Life Cycle Data Model design, textual descriptions and specifications. Concept and relationship analysis, users involved, data source environment specification. Solution targeting, usage scenarios analysis.

- *Kickof* - the ontology requirements are specified targeting the main concepts such as the participants, activity outputs, main activity tasks, more detailed domain and goal identification, ontology design models with respect to data sources involved, use cases and user role definitions, including the concepts and relationships identification.

- *Refinement* - the abstract concepts and relationships specified in the previous precesses are modeled and formalized using the Web Ontology Language.
- *Evaluation* - this process demonstrates the usage of the implemented ontology applied on generated instances of the domain of interest. In this case SPARQL query language is used to extract the ontology knowledge.
- *Maintenance* - this process describes the maintenance role management.

5.3 Life Cycle Ontology Classes

First of all we are going to identified all Life Cycle Ontology classes and property restrictions applied one these classes, which provide the most basic abstraction concept in OWL. Every OWL class, defined by a user, is a subclass of a predefined class *owl:Thing* and therefor each individual is member of this class.
Life Cycle represents a central ontology class instance containing five main activities as shown in Figure 5.1 below. The concrete OWL implementation looks as follows:

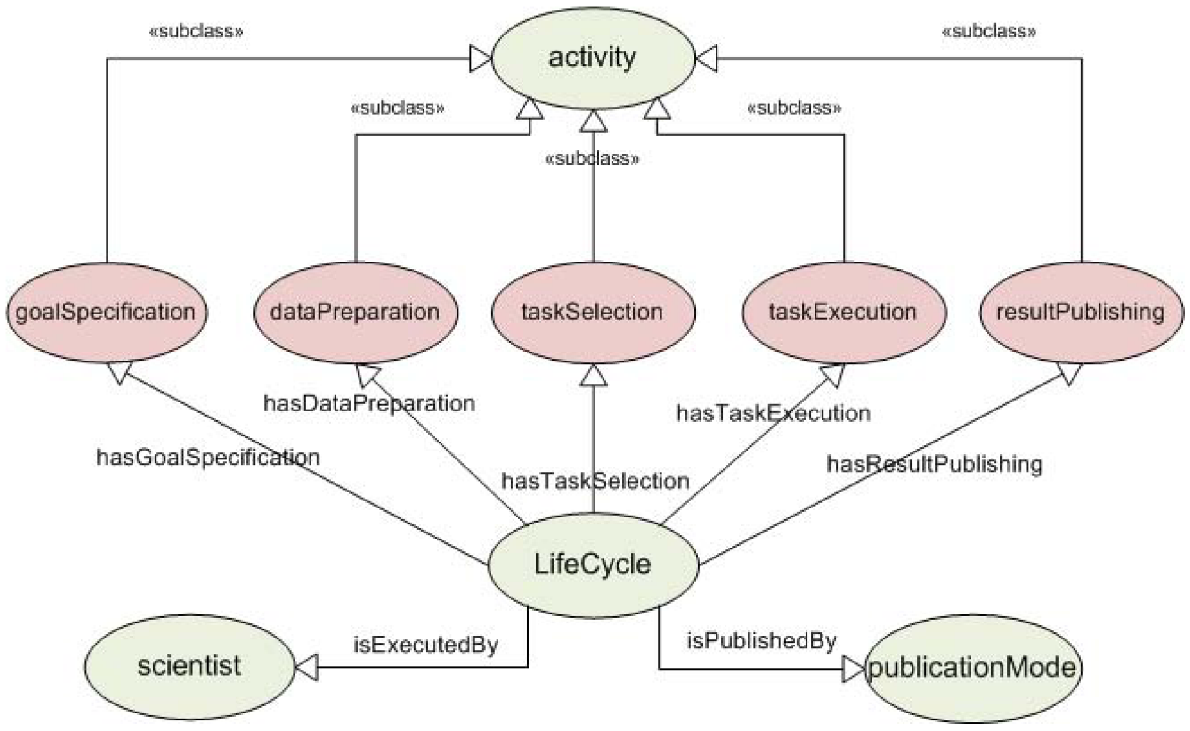


Figure 5.1: e-Science life cycle - activity relation

```
<owl:Class rdf:about="#lifeCycle">
```

```
<rdfs:subClassOf rdf:resource="&owl;Thing"/>
<rdfs:subClassOf>
   <owl:Restriction>
        <owl:onProperty rdf:resource="#hasTaskExecution"/>
        <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">
        1</owl:maxCardinality>
   </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
   <owl:Restriction>
        <owl:onProperty rdf:resource="#hasTaskSelection"/>
        <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">
        1</owl:maxCardinality>
   </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
   <owl:Restriction>
        <owl:onProperty rdf:resource="#hasResultPublishing"/>
        <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">
        1</owl:maxCardinality>
   </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
    <owl:Restriction>
        <owl:onProperty rdf:resource="#hasGoalSpecification"/>
        <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">
        1</owl:maxCardinality>
    </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
    <owl:Restriction>
        <owl:onProperty rdf:resource="#hasDataPreparation"/>
        <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">
        1</owl:maxCardinality>
     </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
```

```
      <owl:Restriction>
          <owl:onProperty rdf:resource="#isExecutedBy"/>
          <owl:allValuesFrom rdf:resource="#Scientist"/>
       </owl:Restriction>
   </rdfs:subClassOf>
   <rdfs:subClassOf>
       <owl:Restriction>
          <owl:onProperty rdf:resource="#isPublishedBy"/>
          <owl:someValuesFrom rdf:resource="#publicationMode"/>
       </owl:Restriction>
   </rdfs:subClassOf>
</owl:Class>
```

We define a class *lifeCycle* as a subclass of a predefined class *owl:Thing*. A constraint *owl:maxCardinality* defines a value, which specifies the number of property values individuals of a class may contain at most, in this case a *lifeCycle* may contain at most one value of each of the declared properties: *hasGoalSpecification, hasDataPreparation, hasTaskExecution, hasTaskSelection, hasResultPublishing*, which refer to the five main life cycle activities. Further we specify that the class *lifeCycle* may only be executed by *scientist*, and that each lifeCycle is published by a particular publication mode. Figure 5.2 gives a more detailed illustration of a publication mode.

A concrete OWL implementation of the publication mode looks as follows:

```
  <owl:Class rdf:about="#publicationMode">
        <rdfs:subClassOf rdf:resource="&owl;Thing"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#hasPublicationMode"/>
                <owl:someValuesFrom rdf:resource="#colllaboration"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#hasPublicationMode"/>
                <owl:someValuesFrom rdf:resource="#researchDomain"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
```

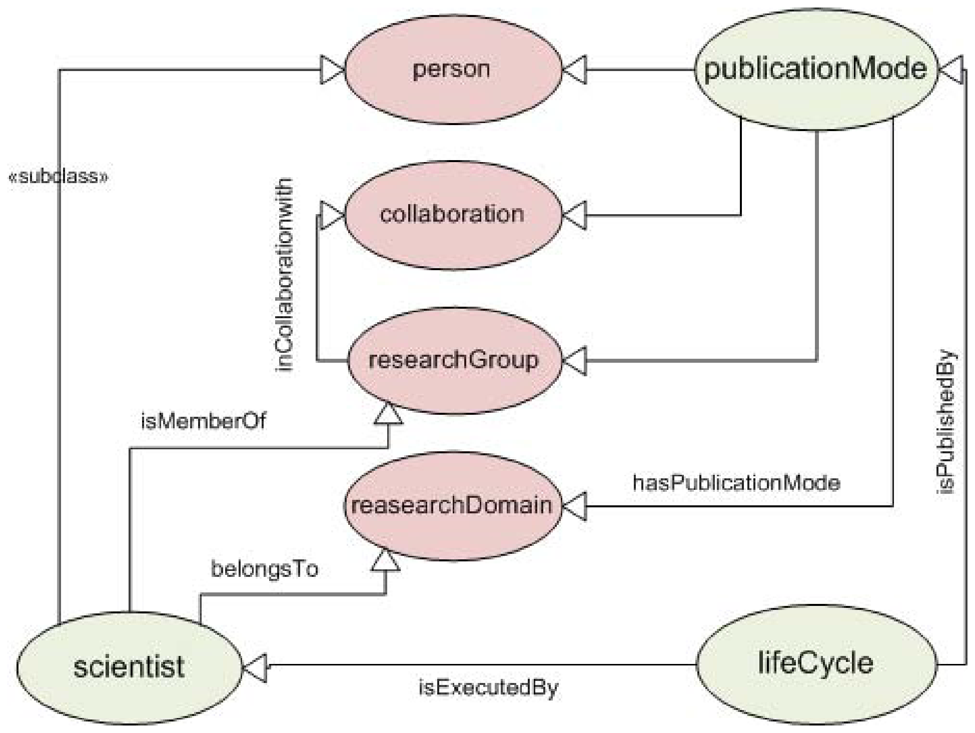


Figure 5.2: e-Science life cycle - publication mode

```
            <owl:Restriction>
                <owl:onProperty rdf:resource="#hasPublicationMode"/>
                <owl:someValuesFrom rdf:resource="#person"/>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#hasPublicationMode"/>
                <owl:someValuesFrom rdf:resource="#researchGroup"/>
            </owl:Restriction>
        </rdfs:subClassOf>
  </owl:Class>
```

Life cycles may be published into the dataspace and thus is available to other users of a wider collaboration with respect to other research areas. We distinguish between four publication modes, (1) *free access*, represented by a class *person*, meaning that anybody may view the published results (2) *research domain*, represented by a *researchDomain* class, while restricting the results to certain domains , (3) *collaboration*, represented by a *collaboration* class with restrictions to particular collaborations, and (4) *research group*, restricting the published life cycles to some particular research groups. Figure 5.2 provides further relationships indicating that a class *scientist* is derived from the class *person* while being member of some research groups and belonging to certain research domains. This information is coded in OWL as shown below:

```
<owl:Class rdf:about="#scientist">
     <rdfs:subClassOf rdf:resource="#person"/>
     <rdfs:subClassOf>
         <owl:Restriction>
             <owl:onProperty rdf:resource="#isMemberOf"/>
             <owl:someValuesFrom rdf:resource="#researchGroup"/>
         </owl:Restriction>
     </rdfs:subClassOf>
     <rdfs:subClassOf>
         <owl:Restriction>
             <owl:onProperty rdf:resource="#execute"/>
             <owl:someValuesFrom rdf:resource="#lifeCycle"/>
         </owl:Restriction>
     </rdfs:subClassOf>
     <rdfs:subClassOf>
         <owl:Restriction>
             <owl:onProperty rdf:resource="#belongsToDomain"/>
             <owl:someValuesFrom rdf:resource="#researchDomain"/>
         </owl:Restriction>
     </rdfs:subClassOf>
</owl:Class>
```

Each activity feeds the dataspace with new participants, which are categorized into tree different participant subgroups: background data, primary data and derived data. The concepts identifying the different participant subgroups are illustrated in Figure 5.3, and defined as subclasses of the class *participant*. All these participants belong to at least one or more e-Science life cycles. The classes *domainOntology, dataStatistics, dataMiningOntology, resourceFile,* and *workflowDescription* are subclasses of a class *backgroundData*, and represent a data which is referenced while executing life cycle iterations. On the right of Figure 5.3 we can see which output is produced while executing which life cycle activities. Each referenced background data and every generated output is added as a participant into a dataspace. For example the activity *goalSpecification* adds new domain ontologies, the activity *dataPreparation* adds new data sets, which are generated using *primaryData* as well as

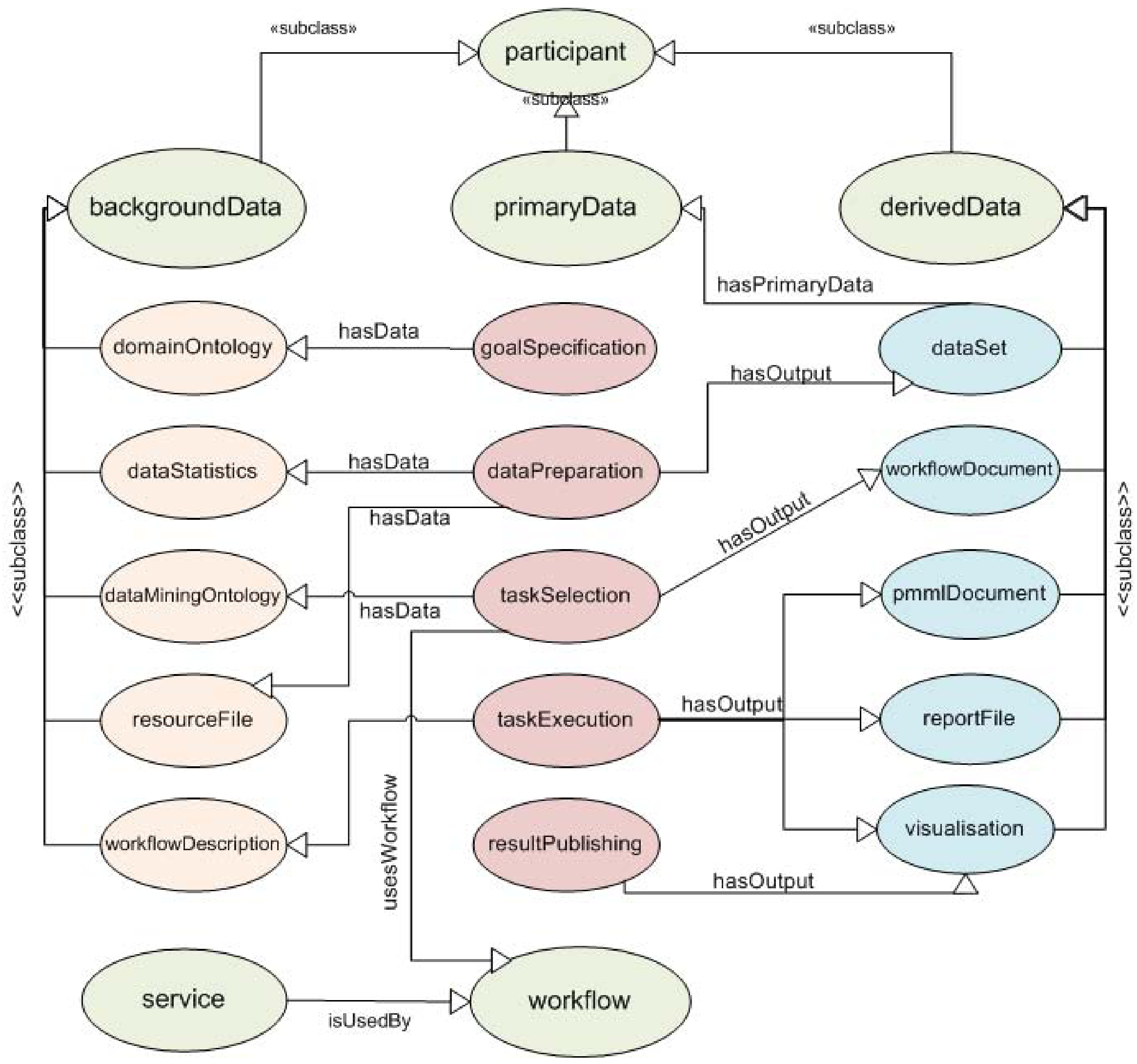


Figure 5.3: e-Science life cycle - participants

OGSA-DAI *resourceFile.*The activity *taskSelection* adds new *workflowDescription* documents, while the activity *taskExecution* adds new *pmmlDocuments* describing the data mining model applied, and finally the activity *resultPublishing* adds new documents visualizing the achieved data mining outputs. As shown in Figure 5.3 the activity *taskSelection* implements a usesWorkflow property linking to a class *workflow* indicating that the activity consists of several steps that are organized as workflows using different data mining services.

A concrete OWL implementation of the *dataPreparation* activity considering the different participants involved is coded as follows:

```
<owl:Class rdf:about="#dataPreparation">
     <rdfs:subClassOf rdf:resource="#activity"/>
     <rdfs:subClassOf>
         <owl:Restriction>
            <owl:onProperty rdf:resource="#hasOGSADaiResourceFile"/>
            <owl:someValuesFrom rdf:resource="#resourceFile"/>
         </owl:Restriction>
     </rdfs:subClassOf>
     <rdfs:subClassOf>
         <owl:Restriction>
            <owl:onProperty rdf:resource="#hasDataSet"/>
            <owl:someValuesFrom rdf:resource="#dataSet"/>
         </owl:Restriction>
     </rdfs:subClassOf>
     <owl:disjointWith rdf:resource="#goalSpecification"/>
     <owl:disjointWith rdf:resource="#resultPublishing"/>
     <owl:disjointWith rdf:resource="#taskExecution"/>
     <owl:disjointWith rdf:resource="#taskSelection"/>
</owl:Class>
```

Additionally we define that a class *dataPreparation* is disjoint from other activity classes using the owl:disjointWith statement, which indicates that these classes do not share any individuals. Figure 5.4 provides a general view on the ontology illustrating how the implemented life cycle concepts are related to each other. We can see the a class *descriptionData* is used to describe some life cycle activities, workflows, services, and research domains. For each class implementing the class *descriptionData* a user can specify some keywords, descriptions and a description name. *Student* is a subclass of a class *person* and is only able to view performed life

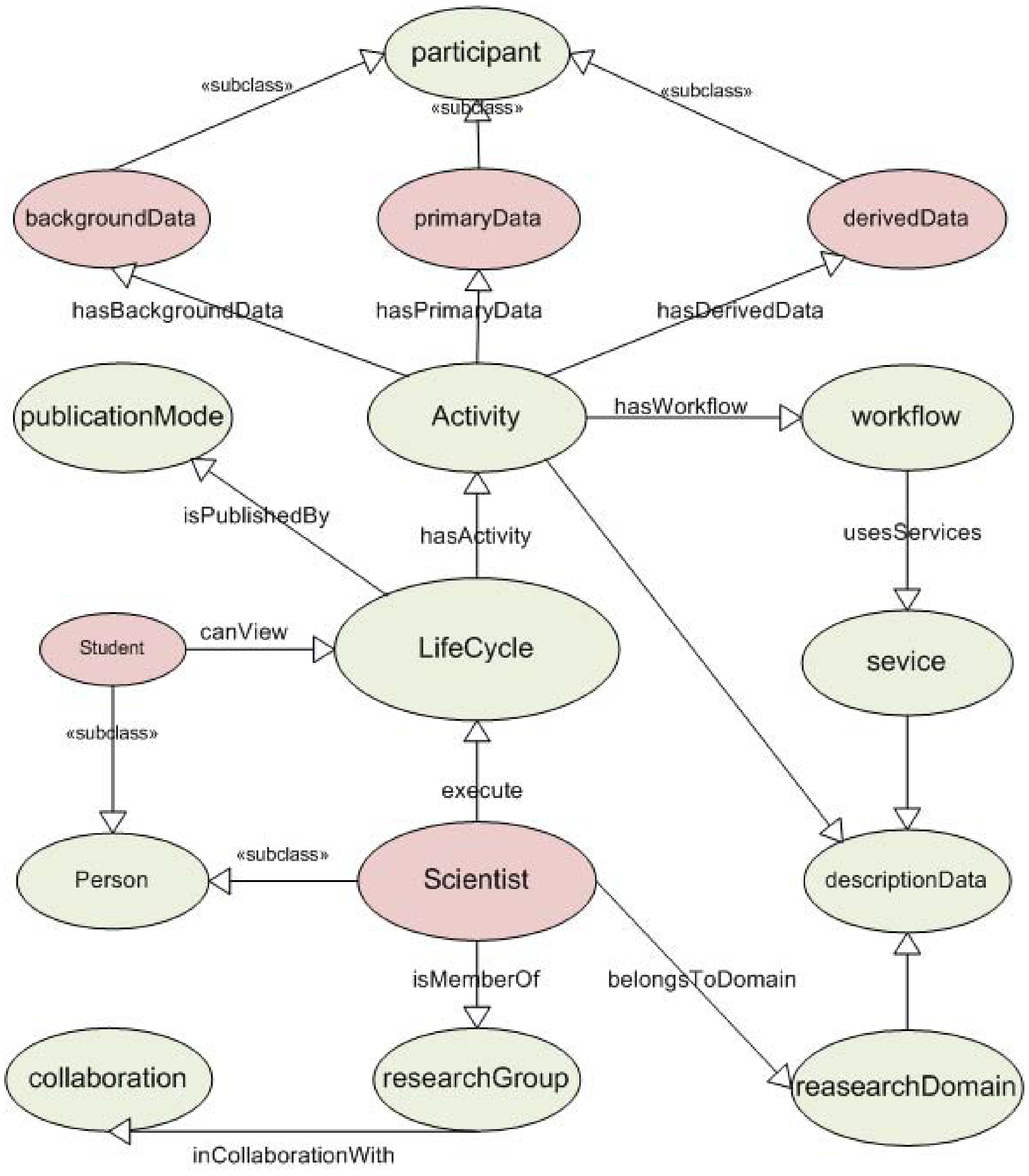


Figure 5.4: e-Science life cycle - general view

cycles, but not to execute them. Different research groups can be in collaboration with each other and so form *collaborations*.

5.4 Life Cycle Ontology Properties

A declaration of a object or data type property alone, does not imply which individuals are related to each other. By the use of *range* and *domain* statements we can define which classes are related to each other. A restriction domain indicates that the subject of such declared property has to belong to related instance of a class. For example a property domain of *execute* is restricted to individuals of the class *scientist*. A *range* statement indicates that the objects of the property range has to belong to a instance of defined class, which means that the property range is restricted to individuals of the class *lifeCycle*, which is illustrated below as a concrete OWL implementation:

```
<owl:ObjectProperty rdf:about="#execute">
    <rdfs:range rdf:resource="#lifeCycle"/>
    <rdfs:domain rdf:resource="#scientist"/>
</owl:ObjectProperty>
```

A definition of a *isExecutedBy* object property includes a ”inverse property” declaration using the already defined *execute* property, interchanging a direction of a range and domain relation, indicating that life cycles are executed by scientists.

```
<owl:ObjectProperty rdf:about="#isExecutedBy">
   <owl:inverseOf rdf:resource="#execute"/>
   <rdfs:domain rdf:resource="#lifeCycle"/>
   <rdfs:range rdf:resource="#scientist"/>
</owl:ObjectProperty>
```

Lets look at a definition of the *hasDescriptionData* object property. It also includes a defined ”inverse properties” *describes*, which interchanges a direction of a range and domain relation of the *hasDescriptionData* property. We use an *owl:unionOf* statement to create a union of instances of the classes listed inside of the statement, restricting the property only to individuals of these classes. The property range is restricted to individuals of a class *descriptionData*.

```
 <owl:ObjectProperty rdf:about="#hasDescriptionData">
        <owl:inverseOf rdf:resource="#describes"/>
        <rdfs:range rdf:resource="#descriptionData"/>
        <rdfs:domain>
            <owl:Class>
                <owl:unionOf rdf:parseType="Collection">
                    <rdf:Description rdf:about="#dataPreparation"/>
                    <rdf:Description rdf:about="#goalSpecification"/>
                    <rdf:Description rdf:about="#researchDomain"/>
                    <rdf:Description rdf:about="#researchGroup"/>
                    <rdf:Description rdf:about="#service"/>
                    <rdf:Description rdf:about="#taskExecution"/>
                    <rdf:Description rdf:about="#workflow"/>
                </owl:unionOf>
            </owl:Class>
        </rdfs:domain>
</owl:ObjectProperty>
```

In the next example we declare a *hasGoalSpecification* object property defining that individuals of a class *lifeCycle* belong to instances of a class *goalSpecification.* Further we declare a *owl:FunctionalProperty* as a subclass of *rdf:property.* This property definition may contain one value for each object at most, which means that one particular life cycle instance may include only one individual of a class *goalSpecification.*

```
<owl:ObjectProperty rdf:about="#hasGoalSpecification">
     <rdf:type rdf:resource="&owl;FunctionalProperty"/>
      <rdfs:range rdf:resource="#goalSpecification"/>
      <rdfs:domain rdf:resource="#lifeCycle"/>
</owl:ObjectProperty>
```

The following table summarizes all defined object properties and their restrictions:

Object Property	**Range**	**Domain**	**inverse Property**
hasPrimaryData	dataResource	dataSet	usedToGenerate
belongsToDomain	researchDomain	scientist	isDomainOf
hasCollaborativeGroup	researchGroup	collaboration	inCollaborationWith
hasDataPreparation	dataPreparation	lifeCycle	
hasDataSet	dataSet	dataPreparation	isDataSetOf
execute	scientist	lifeCycle	isExecutedBy
hasGoalSpecification	goalSpecification	lifeCycle	
hasMembers	scientist	researchGroup	isMemberOf

Table 5.1: Object properties

Object Property	**Range**	**Domain**	**inverse Property**
hasOGSADaiResourceFile	resourceFile	dataPreparation	isResourceFileOf
hasPMMLDocument	pmmlDocument	taskExecution	isPMMLDocumentOf
hasReferenceTo	domainOntology	goalSpecification	
hasReportFile	reportFile	taskExecution	isReportFileOf
hasResultPublishing	resultPublishing	lifeCycle	
hasServiceOntology	serviceOntology	service	isServiceOntologyOf
hasTaskExecution	taskExecution	lifeCycle	
hasTaskSelection	taskSelection	lifeCycle	
hasVisualisation	visualisation	resultPublishing	isVisualisationOf
hasVisualisation	visualisation	taskExecution	isVisualisationOf
hasWorkflow	workflow	taskSelection	isWorkflowOf
hasWorkflowDocument	workflowDokument	taskSelection	isWorkflowDocumentOf
isPublishedBy	publicationMode	lifeCycle	isPublicationModeOf
isUsedFor	workflow	service	usesService
isResourceFileOf	dataPreparation	resourceFile	
usedToGenerate	dataSet	dataResource	

Table 5.1: Object properties

The following example illustrates some of the data type properties which are restricted to individuals of the class *person*. The concrete OWL implementation of the *firstName, homepage, email, and age* object properties looks as follows:

```
<owl:DatatypeProperty rdf:about="#firstName">
       <rdfs:domain rdf:resource="#person"/>
       <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="#homepage">
       <rdfs:domain rdf:resource="#person"/>
       <rdfs:range rdf:resource="&xsd;anyURI"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="#email">
       <rdfs:domain rdf:resource="#person"/>
       <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:about="#age">
       <rdfs:domain rdf:resource="#person"/>
```

```
        <rdfs:range rdf:resource="&xsd;integer"/>
 </owl:DatatypeProperty>
```

The following table summarizes all defined data type properties and their restrictions:

Data Type Property	**Domain**	**Range**
firstName	person	&xsd;string
lastName	person	&xsd;string
age	person	&xsd;integer
homepage	scientist	&xsd;string
birthDate	person	&xsd;date
country	person	&xsd;string
state	person	&xsd;string
phone	scientist	&xsd;string
email	scientist	&xsd;string
title	scientist	&xsd;string
keywords	descriptionData	&xsd;string
description	descriptionData	&xsd;string
name	descriptionData	&xsd;string
pmmlReference	pmmlDocument	&xsd;anyURI
reportURL	reportFile	&xsd;anyURI
resourceURI	dataResource	&xsd;anyURI
serviceOntologyReference	serviceOntology	&xsd;anyURI
visualisationReference	visualisation	&xsd;anyURI
visualisationType	visualisation	&xsd;anyURI
resourceFileDescription	resourcefile	&xsd;string
resourceFileReference	resourceFile	&xsd;anyURI
workflowReference	workflow	&xsd;anyURI
workflowName	workflow	&xsd;string

Table 5.2: Data type properties

Chapter 6

e-Science Life Cycle Concept Evaluation

6.1 Introduction

This chapter deals with the evaluation of the e-Science Life Cycle concept. First of all we are going to look at project related dataspaces and their executed life cycle iterations within a generic application scenario. We will focus on how a project state evolves within a dataspace system and illustrate each state within a project state diagram. Then, we will concentrate on a concrete application scenario located in the field of Traditional Chinese Medicine (TCM) and evaluate the application results using the SPARQL query language.

6.2 Generic Application Scenario

Let's assume we have a dataspace (DS) setup for a specific domain with different subdataspaces defined for various research studies and each representing a separate project. Each of these projects has a number of executed life cycle iterations represented as instances of these life cycles and creating a project related dataspace. Once a certain project dataspace is initialized, specifying a general setup model for life cycle iterations, each additional life cycle is defined as an instance of this generic life cycle.

Suppose a *Project 1* is setup by a certain research group executing one life cycle iteration and so defining an initial generic life cycle instance. All further scientific experiments executed within this particular project are based upon this life cycle setup for all participating project members. Each performed life cycle activity including the corresponding primary and derived data are related to this particular

life cycle instance. These instances are organized using the life cycle ontology where all project related relationships are captured and saved.
The members of each project can highly profit from this kind of organized relationship information allowing an easy reuse of life cycle activities and an intelligent discovery of already generated datasets and their corresponding published results. Through an increasing number of executed life cycles, the dataspace is able to grow into a large space of well-organized and inter-connected information, which can easily be queried and reused.
Life cycles, which are created within *Project 2* can for example include different activities of executed life cycles from *Project 1* and so create research collaborations which are based upon overlapping reuse of activities and their corresponding primary and derived data as illustrated in Figure 6.1. These research collaborations can contain various number of research groups which can be involved in different projects while working in related domain applications thus operating on the same primary data. In order to reuse already executed life cycles, their produced data sets and derived data, such as for example workflow and PMML documents, report files or applied visualizations, the data has to be published within a dataspace and further be accessible for a particular research member or research group.

Assume we have a set of existing Projects, $P = (P1,P2,..,Pn)$ where $Pi \in DS$ and Projects are entering and leaving this set. For each Project Pi we define several basic constraints:

- each Project Pi has at least one generic Life Cycle LCg that is executed by one certain scientist.
- each Project Pi contains a set of primary and derived data acting as participants in a dataspace.
- each Project Pi has at least one research group working on it.
- each Project Pi that has overlapping reuse of activities must be published in a dataspace.

After a project is generated and its results are published in a dataspace, the project owner can anytime reset the publication mode applied on the executed project and even remove the project from the dataspace. In this case, all executed life cycle iterations and their corresponding dataspace participants are deleted.

Figure 6.2 shows a dataspace project state diagram describing how a project state

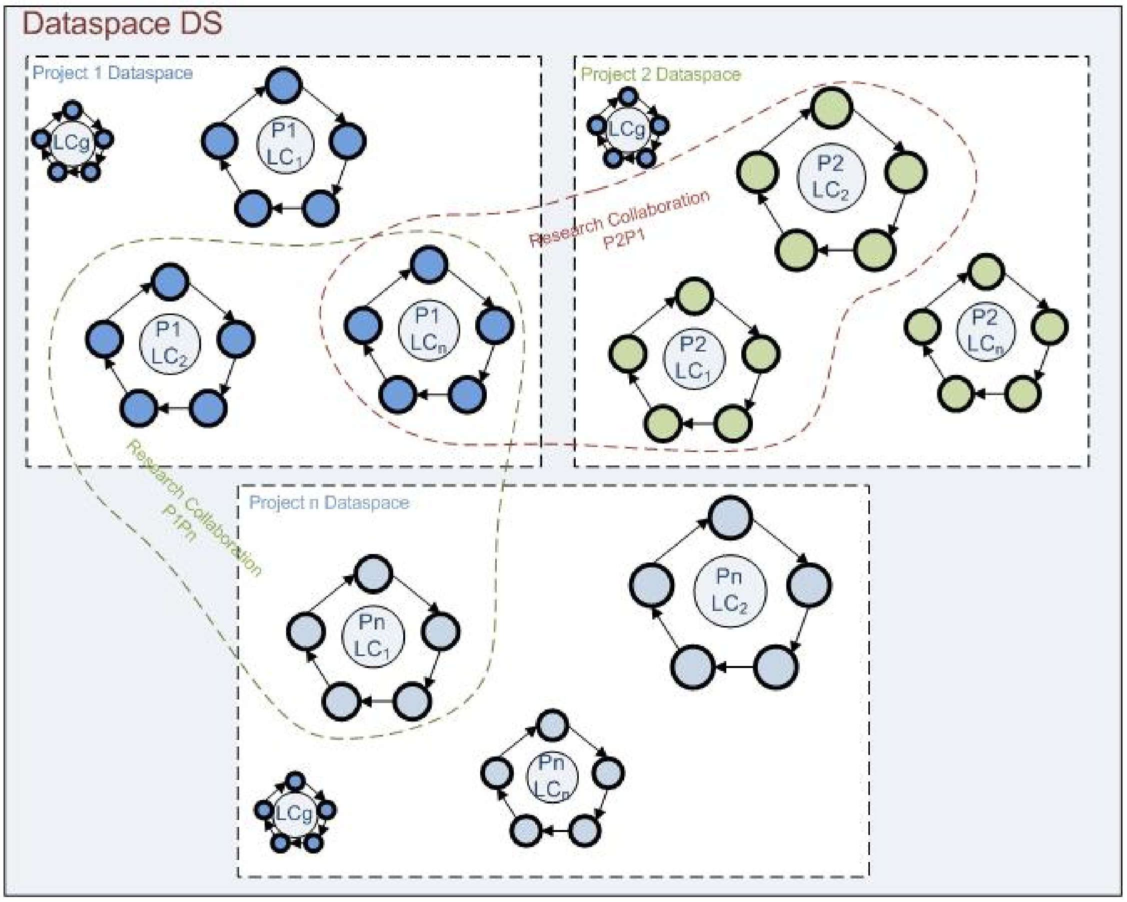
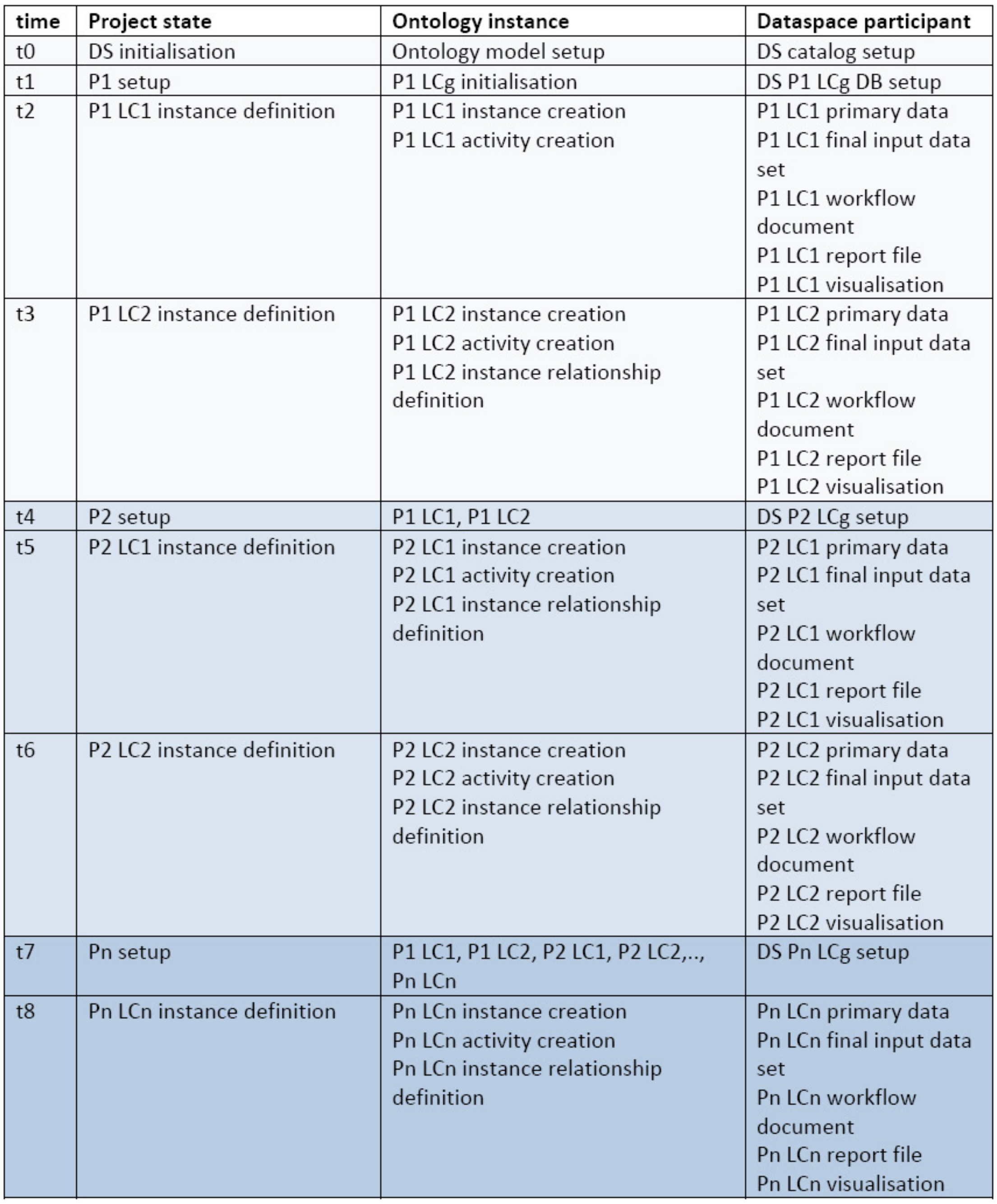


Figure 6.1: Project dataspaces - life cycle reuse

evolves within a dataspace system representing the life cycle instances modeled in the ontology and data elements acting as participants in a dataspace.

In the following we will discuss each Project state at a given time instance t_j:

- t0 - a dataspace is initialized, indicating the initial state of the ontology while having no life cycle instances recorded yet.
- t1 - project *P1* is setup and the generic Life Cycle *LCg* for P1 is initialized.
- t2 - first LC instance of the project *P1* is defined and created executing the life cycle activities and producing several derived data outputs acting as participants in a dataspace.
- t3 - second LC instance of the project *P1* is defined and created. Additional life cycle results are recorded and published as new participants of a dataspace. Further instance relationships are defined indicating possible relations among published results.
- t4 - project *P2* is setup and the generic Life Cycle *LCg* for *P2* is initialized. As we can see in Figure, the Ontology already contains two executed LC instances which can be queried and reused for purposes of project *P2* experiments.
- t5 - first LC instance of the project *P2* is defined and created. New dataspace participants are generated by *LC1* of *P2* and inserted into the *P2DS*.
- t6 - second LC instance of the project *P2* is defined and created. Again new dataspace participants are generated by *LC2* of *P2* and inserted into the *P2DS*. If e.g. *LC2* of *P2* reuses some activities (and consequently its corresponding DS participants) of previously executed LC then these shared participants are linked.
- t7 - after a number of created Projects including several executed life cycles a dataspace has evolved into a space of primary and derived data with semantically rich relationships among each other which can easily be queried for answers to specific questions.
- t8 - for any life cycle instance *Pn LCn* definition of Project *Pn* we capture its activities as instances of the e-Science Life Cycle Ontology and publish the experiment results as participants of a dataspace.



time	Project state	Ontology instance	Dataspace participant
t0	DS initialisation	Ontology model setup	DS catalog setup
t1	P1 setup	P1 LCg initialisation	DS P1 LCg DB setup
t2	P1 LC1 instance definition	P1 LC1 instance creation P1 LC1 activity creation	P1 LC1 primary data P1 LC1 final input data set P1 LC1 workflow document P1 LC1 report file P1 LC1 visualisation
t3	P1 LC2 instance definition	P1 LC2 instance creation P1 LC2 activity creation P1 LC2 instance relationship definition	P1 LC2 primary data P1 LC2 final input data set P1 LC2 workflow document P1 LC2 report file P1 LC2 visualisation
t4	P2 setup	P1 LC1, P1 LC2	DS P2 LCg setup
t5	P2 LC1 instance definition	P2 LC1 instance creation P2 LC1 activity creation P2 LC1 instance relationship definition	P2 LC1 primary data P2 LC1 final input data set P2 LC1 workflow document P2 LC1 report file P2 LC1 visualisation
t6	P2 LC2 instance definition	P2 LC2 instance creation P2 LC2 activity creation P2 LC2 instance relationship definition	P2 LC2 primary data P2 LC2 final input data set P2 LC2 workflow document P2 LC2 report file P2 LC2 visualisation
t7	Pn setup	P1 LC1, P1 LC2, P2 LC1, P2 LC2,.., Pn LCn	DS Pn LCg setup
t8	Pn LCn instance definition	Pn LCn instance creation Pn LCn activity creation Pn LCn instance relationship definition	Pn LCn primary data Pn LCn final input data set Pn LCn workflow document Pn LCn report file Pn LCn visualisation

Figure 6.2: Project dataspaces state diagram

6.3 Concrete Use-Case Scenario

6.3.1 Introduction

A first application highly profiting from the above described life cycle of scientific data is located in the field of Traditional Chinese Medicine (TCM). According to the basic TCM theory, the human body has 14 acupuncture meridians, which are a secret to our biological and medical knowledge. The China-Austria Data Grid project [CAD07], which includes nine different research groups in China and Austria is devoted to meridian measurements using various measurement techniques while collecting huge amount of meridian data. In order to use this large amount of valuable information, it is necessary to make available a space of data with semantically rich relationships accessible for other research groups targeting different research areas.
In the following we will exemplify the relevant data elements and resources identified while performing e-Science life cycle iterations on the the above introduced application. Additionally, the SPARQL query results will be evaluated using *Protege Open SPARQL Query Panel*.

6.3.2 Life Cycle Ontology Individuals (A Case Study)

1 *Specify Goals Example* - The following information is recorded by this activity for the sample iteration. For each instance we show how the information is modeled and represented in the e-Science Life Cycle Ontology.

- Goal specification name: *goalSpecification_NIGM*

```
<hasGoalSpecification rdf:resource="#goalSpecification_NIGM"/>
```

- a scientist username: *CADGrid-researcher3*

```
<scientist rdf:about="#CADGrid-researcher3"></scientist>
```

- a research group he or she is member of: *CADGrid-buct.edu.cn*

```
<researchGroup rdf:about="#CADGrid-buct.edu.cn">
   <hasMembers rdf:resource="#CADGrid-researcher3"/>
</researchGroup>
```

- description of objectives, including a name of the description data *descriptionData_gS_NIGM*

- keywords such as *The Non-Invasive Blood Glucose Measurement (NIGM)* and *NIGM measurement techniques*
- a textual description: *The Non-Invasive Blood Glucose Measurement (NIGM) method offers a novel promising non-invasive technology for measuring patient's data. In order to be accepted by general medical community, the quality of provided results will have to be verified in extensive clinical trials following well defined protocol. The main aim of this experiment is to specify medical conditions under which values of blood glucose obtained through the NIGM method are as reliable as data provided by the invasive measurement techniques that are currently the only standard used and accepted by western medicine* [EHL+08].

```
<descriptionData rdf:about="#descriptionData_gS_NIGM">
   <keywords xml:lang="en">NIGM</keywords>
   <Name xml:lang="en">
        The Non-Invasive Blood Glucose Measurement (NIGM)
   </Name>
   <keywords xml:lang="en">blood glucose</keywords>
   <description xml:lang="en">
        he Non-Invasive Blood Glucose Measurement (NIGM)
        method
   </description>
   <keywords xml:lang="en"
        >measurement techniques</keywords>
   <describes rdf:resource="#goalSpecification_NIGM"/>
</descriptionData>
```

- Additionally, we save a name of the executed life cycle *NIGM_CADGrid_001*.

```
<scientist rdf:about="#CADGrid-researcher3">
   <execute rdf:resource="#NIGM_CADGrid_001"/>
</scientist>
```

We further save

2 *Prepare Data Example* - In this experiment there is no need for data integration, thus the final input data set is simply the deployed OGSA-DAI resource. The following information is recorded by this activity.

- Data preparation name: *dataPreparation_NIGM*

```
<hasDataPreparation rdf:resource="#dataPreparation_NIGM"/>
```

– Resource name: CADGrid-MMDB1

```
<resourceFile rdf:about="#CADGrid-MMDB1">
   <isResourceFileOf rdf:resource="#dataPreparation_NIGM"/>
</resourceFile>
```

– OGSA-DAI resource file location: http://.../CADGrid-MMDB1.ResourceFile

```
<resourceFile rdf:about="#CADGrid-MMDB1">
  <resourceFileReference rdf:datatype="&xsd;anyURI">
      http://.../CADGrid-MMDB1
   </resourceFileReference>
</resourceFile>
```

– Resource description: Within various experiments on diabetes patients the CADGrid research team has collected large amount of meridian measurement data and corresponding blood glucose values. The experiments were applied on 300 diabetic patients, conducted in the Dong Fang hospital, Beijing, China. For each patient 50 meridian measurements and equally many measures were applied using the conventional invasive blood glucose method. Meridian measurements are represented as value pairs, the input signal value, and the output signal value. One meridian measurement generates around 1000 to 10000 value pairs [EHL+08].

```
<resourceFile rdf:about="#CADGrid-MMDB1">
   <resourceFileDescription rdf:datatype="&xsd;string"
       >The experiments were applied on 300 diabetic patients..
   </resourceFileDescription>
</resourceFile>
```

Background data resources participating in this example application are providing information about the patient's health condition and the meridian on which the measurement was performed. They are considered as background data resources within a dataspace.

3 *Select Appropriate Tasks Example* - In this example selected tasks are organized within the NIGM workflow, which consists of the following algorithms, deployed as CADGrid services: (1) System Identification, (2) Kalman Filtering, (3) Wavelet Transformation, (4) Fast Fourier Transformation, (5) Combination Service, and finally (6) Back Propagation Neural Network. The Following information is captured.

– Task selection name: NIGM_Inst23

```
<hasTaskSelection rdf:resource="#taskSelection_NIGM"/>
```

– Workflow name: NIGM_Inst23 and the above mentioned CADGrid services.

```
<workflow rdf:about="#NIGM_workflow">
   <usesService rdf:resource="#CADGrid_service_S_Identif"/>
   <usesService rdf:resource="#CADGrid_service_Kalman_F"/>
   <usesService rdf:resource="#CADGrid_service_Wavelet_T"/>
   <usesService rdf:resource="#CADGrid_service_F_Fourier"/>
   <usesService rdf:resource="#CADGrid_service_C_Service"/>
   <usesService rdf:resource="#CADGrid_service_NN"/>
   <isWorkflowOf rdf:resource="#taskSelection_NIGM"/>
</workflow>
```

– Workflow description: The NIGM workflow processes a compute intensive signal phase and followed that sets up a neural network, which can be used for computing humans blood glucose values. Input data sets are on one side meridian measurements, collected using a specially developed instrument and on the other side blood glucose values measured using the conventional method.

```
<workflow rdf:about="#NIGM_workflow">
  <hasDescriptionData rdf:resource="#Workflow_NIGM_Inst23"/>
</workflow>

<descriptionData rdf:about="#Workflow_NIGM_Inst23">
   <Name rdf:datatype="&xsd;string">NIGM_Inst23</Name>
   <keywords rdf:datatype="&xsd;string"
      >blood glucose values</keywords>
   <keywords rdf:datatype="&xsd;string"
      >intensive signal processing</keywords>
   <keywords rdf:datatype="&xsd;string"
      >meridian measurements</keywords>
   <description xml:lang="en"
      >Runs a compute intensive signal processing phase...
   <usesService rdf:resource="#CADGrid_service_001"/>
   <describes rdf:resource="#NIGM_workflow"/>
</descriptionData>
```

– Workflow reference: http://.../NIGM_Inst23.gwa

```
<workflow rdf:about="#NIGM_workflow">
   <workflowReference rdf:datatype="&xsd;anyURI">
       http://.../NIGM_Inst23.gwa
   </workflowReference>
</workflow>
```

4 *Run Tasks Example* - We record the PMML document representing the output of the neural network model set up by the NIGM workflow and its corresponding visualization document, all considered as derived data. The information is captured as follows:

– Task execution name: taskExecution_NIGM

```
<hasTaskExecution rdf:resource="#taskExecution_NIGM"/>
```

– PMML Document: pmmlDocument_NIGM

```
<pmmlDocument rdf:about="#pmmlDocument_NIGM">
   <pmmlReference rdf:datatype="&xsd;anyURI"></pmmlReference>
   <isPMMLDocumentOf rdf:resource="#taskExecution_NIGM"/>
</pmmlDocument>
```

– NIGM visualisation: NIGM_visualisation

```
<visualisation rdf:about="#NIGM_visualisation">
   <visualisationType rdf:datatype="&xsd;string">
       PMLL Visualisation</visualisationType>
   <visualisationReference rdf:datatype="&xsd;anyURI">
       http://lela.par.univie.ac.at/cadgrid/
       CADGrid_OutputModel_NN.svg
   </visualisationReference>
   <isVisualisationOf rdf:resource="#taskExecution_NIGM"/>
</visualisation>
```

5 *Process and Publish Results Example* - A publish mode is dedicated to each output, based on which the outputs are published into a dataspace. Finally a publication mode of the executed Life Cycle is generated, restricting the published results to one particular research group *CADGrid-buct.edu.cn.* We capture the information as follows:

– Task execution name: resultPublishing_NIGM

```
<hasResultPublishing rdf:resource="#resultPublishing_NIGM"/>
```

– Publication mode: NIGM_publicationMode

```
<publicationMode rdf:about="#NIGM_publicationMode">
   <hasPublicationMode rdf:resource="#CADGrid-buct.edu.cn"/>
   <isPublicationModeOf rdf:resource="#NIGM_CADGrid_001"/>
</publicationMode>
```

The information captured by this life cycle represents the relationship among its participating data resources (primary, derived, and background data). It also covers the objectives specified for this experiment. Finally the data is published in the dataspace, thus is available for further data mining studies aiming at further improvements in diabetic care and meridian theory resulting in higher patient comfort.

6.3.3 Querying the e-Science Life Cycle Ontology

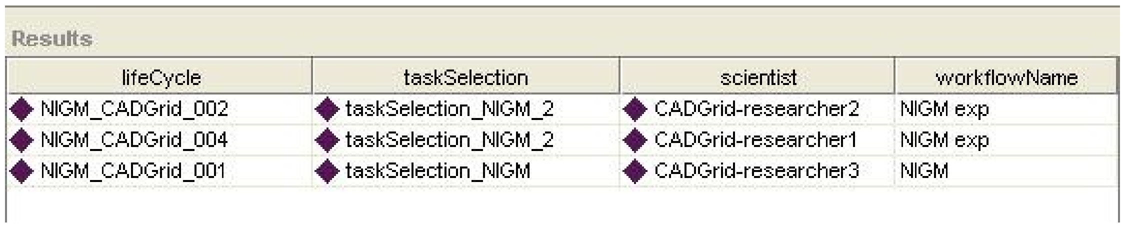
Using *SPARQL* query language and semantically rich described e-Science life cycles we can explore the knowledge captured in the e-Science Life Cycle Ontology. Search and query services can be provided for all the participating data elements of the scientific dataspace identifying relevant data. Each query submitted to the scientific dataspace, will receive not only the matching data but also data of its corresponding e-science activities allowing a user to explore the query results more in detail, discovering which mining task were applied on a particular dataset, the concrete workflow, the workflow report, the results presented in PMML and its corresponding visualizations.
In the following we will illustrate some concrete SPARQL query examples:

SPARQL Query Example 1
Let's assume a user would like to list all life cycles having NIGM workflows applied on them while performing task selection activity and the scientists they were executed by. A concrete SPARQL query might look as follows:

```
PREFIX owl: <http://localhost/LifeCycleOntology.owl#>

SELECT ?lifeCycle ?taskSelection ?scientist ?workflowName
WHERE {
            ?lifeCycle owl:hasTaskSelection ?taskSelection.
            ?lifeCycle owl:isExecutedBy ?scientist.
            ?taskSelection owl:hasWorkflow ?workflow.
            ?workflow owl:workflowName ?workflowName.
            FILTER regex(?workflowName, "nigm", "i")
}
```

A list of results is displayed in Figure 6.3 above:



Results

lifeCycle	taskSelection	scientist	workflowName
◆ NIGM_CADGrid_002	◆ taskSelection_NIGM_2	◆ CADGrid-researcher2	NIGM exp
◆ NIGM_CADGrid_004	◆ taskSelection_NIGM_2	◆ CADGrid-researcher1	NIGM exp
◆ NIGM_CADGrid_001	◆ taskSelection_NIGM	◆ CADGrid-researcher3	NIGM

Figure 6.3: Protege open SPARQL query panel results example 1

The *select* statement consists of *?lifeCycle ?taskSelection ?scientist ?workflowName* data variables that are returned in the query result. The SPARQL results are represented in a table form, whereas every row represents one query answer and each variable used in the select statement represents a column in the result table. We match all workflow names containing the case-sensitive expression *NIGM*, indicated by *"i"*.

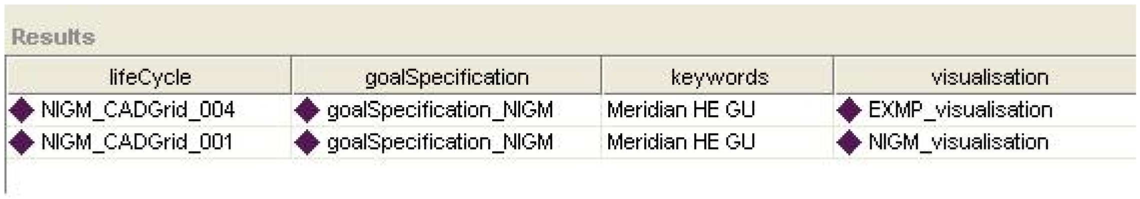
SPARQL Query Example 2

Further, let's say a user wants to apply a NIGM-analysis on meridian HE GU. If the results already exist, he or she could save hours of computation. We might specify a following query to provide a answer to this particular question:

```
PREFIX owl: <http://localhost/LifeCycleOntology.owl#>

SELECT ?lifeCycle ?goalSpecification  ?keywords ?visualisation
WHERE {
            ?lifeCycle owl:hasGoalSpecification ?goalSpecification.
            ?goalSpecification owl:hasDescriptionData ?descriptionData.
            ?lifeCycle owl:hasTaskExecution ?taskExecution.
            ?visualisation owl:isVisualisationOf ?taskExecution.
            ?descriptionData owl:keywords ?keywords.
            FILTER regex(?keywords, "meridian HE GU", "i")
}
```

The result of the above specified query are displayed in Figure 6.4:



Results

lifeCycle	goalSpecification	keywords	visualisation
◆ NIGM_CADGrid_004	◆ goalSpecification_NIGM	Meridian HE GU	◆ EXMP_visualisation
◆ NIGM_CADGrid_001	◆ goalSpecification_NIGM	Meridian HE GU	◆ NIGM_visualisation

Figure 6.4: Protege open SPARQL query panel results example 2

As we can see a user has found some already published visualizations on *meridian HE GU*, which can be now explored more in detail discovering the corresponding life cycle iteration, workflows, applied services, produced outputs, etc.

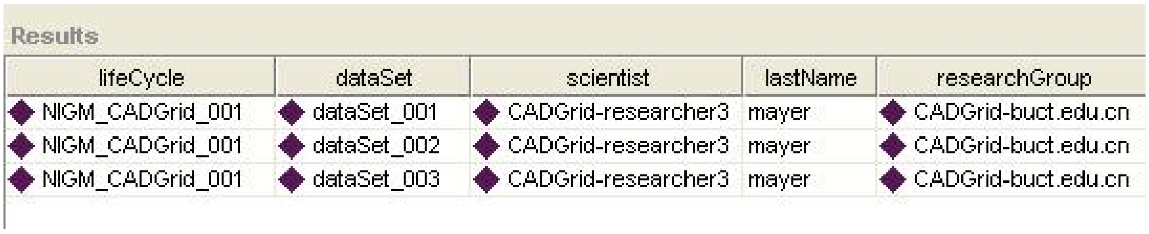
SPARQL Query Example 3

Suppose we need to know which date sets were used while performing particular life cycle iterations executed by a scientist called *Mayer*, and which research groups he is member of. A corresponding SPARQL statement may look as follows:

```
PREFIX owl: <http://localhost/LifeCycleOntology.owl#>

SELECT ?lifeCycle ?dataSet  ?scientist ?lastName ?researchGroup
WHERE {
            ?lifeCycle owl:isExecutedBy ?scientist.
            ?dataSet owl:isDataSetOf ?dataPreperation.
            ?scientist owl:isMemberOf ?researchGroup.
            ?lifeCycle owl:hasTaskExecution ?taskExecution.
            ?scientist owl:lastName ?lastName.
            FILTER regex(?lastName, "mayer", "i")
}
```

The results of the above defined query are displayed in Figure 6.5:



Results

lifeCycle	dataSet	scientist	lastName	researchGroup
◆ NIGM_CADGrid_001	◆ dataSet_001	◆ CADGrid-researcher3	mayer	◆ CADGrid-buct.edu.cn
◆ NIGM_CADGrid_001	◆ dataSet_002	◆ CADGrid-researcher3	mayer	◆ CADGrid-buct.edu.cn
◆ NIGM_CADGrid_001	◆ dataSet_003	◆ CADGrid-researcher3	mayer	◆ CADGrid-buct.edu.cn

Figure 6.5: Protege open SPARQL query panel results example 3

A shown in the figure the query results provide information about particular data sets executed by a certain scientist. Additionally, we display a scientist unique id, last name and the scientists research group. Having found the needed information, a user can for instance further explore what mining task were applied on a discovered dataset, the concrete workflow, the workflow report, the results presented in PMML and its corresponding visualizations.

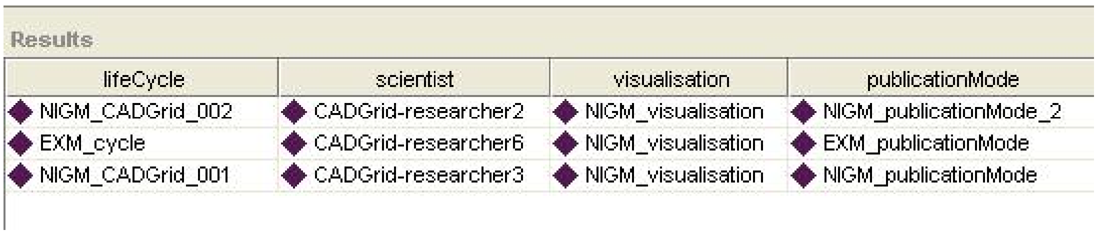
SPARQL Query Example 4

Let's assume we have found some interesting results published in a dataspace but have no access to the published data so as to explore the details of the executed life cycles. We can search for publication mode information of published results in order to receive a access permission:

```
PREFIX owl: <http://localhost/LifeCycleOntology.owl#>

SELECT ?lifeCycle ?scientist ?visualisation ?publicationMode
WHERE {
            ?lifeCycle owl:isExecutedBy ?scientist.
            ?taskExecution owl:hasVisualisation ?visualisation.
            ?lifeCycle owl:isPublishedBy ?publicationMode.
            ?visualisation owl:visualisationName ?visualisationName.
            FILTER regex(?visualisationName, "nigm", "i")
}
```

The results of the above defined query are displayed in Figure 6.6:


Results

lifeCycle	scientist	visualisation	publicationMode
◆ NIGM_CADGrid_002	◆ CADGrid-researcher2	◆ NIGM_visualisation	◆ NIGM_publicationMode_2
◆ EXM_cycle	◆ CADGrid-researcher6	◆ NIGM_visualisation	◆ EXM_publicationMode
◆ NIGM_CADGrid_001	◆ CADGrid-researcher3	◆ NIGM_visualisation	◆ NIGM_publicationMode

Figure 6.6: Protege open SPARQL query panel results example 4

Knowing the particular life cycle iteration information and the corresponding publication mode, a user can either ask for permission or become a member of the particular research group, the life cycle is restricted to, in order to receive access to needed results, which act as participants in a data space.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

The e-Science Life Cycle Dataspace model represents a novel methodology and associated informatics to support the interaction among specific research groups by the means of advanced scientific data management for e-Science applications.

Key contributions are: (1) a hierarchical and iterative metamodel providing a life cycle view of scientific data showing what ideally should happen to data in e-Science applications, while they are processed, is presented generally and by the means of one pilot e-Science application. (2) The e-Science life cycle ontology, organizing the concepts and coherences of e-Science life cycle activities as classes and properties, is developed. (3) The dataspace paradigm presented in [FHM05] is further developed by considering its major research challenge "managing relationships among participants" in order to explicitly support the existing relationship among primary and derived data in scientific collaborations.

The intelligence of the proposed e-Science life cycle model lies in its capability as customizable relationship model for scientific dataspaces, as it covers the creation, representation and searching of semantically rich relationships among participants of a dataspace. It enables researchers to find not only relevant primary data in connection with its derived data, but also lot of semantics about what was initially done with the data, such as which data preprocessing methods have been applied, which data mining and analysis models have been used, which result visualizations are available etc. Further it points to relevant background and ontological data, such as descriptions of applied services, models, research domains etc.

All these information is meant to be the semantically rich relationship among primary and derived data described by the e-Science life cycle ontology. Additionally scientists will retrieve information about the goals specified, which domain it corresponds, and whom to contact in case of interest for engaging collaborations, in short, users will understand for what reason a specific e-Science life cycle was applied.

7.2 Future Work

The development of advanced scientific portals and interfaces collecting the needed information modeled as Life Cycle Ontology is the main part of the ongoing future work, which is currently under investigation. A Scientific Dataspace Manager should be provided in order to manage the relevant research information given by a scientist while creating new life cycle iterations. Therefor, we identify several different user interfaces such as a Dataspace Registration Tool, managing all user registration information, then a Scientific Life Cycle Manager, which we believe will allow creation, execution, and modification facilities of defined life cycle iterations. Further, we think it will be necessary to provide an e-Science Dataspace Life Cycle Explorer, which should enable users to express search queries visually by browsing the dataspace while supported by required interfaces.
However, the basis for intelligent dataspaces for e-Science is developed and has cleared the way towards developing high-productivity e-Science frameworks.

Appendix A

The e-Science Life Cycle Ontology

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:p3="http://localhost/LifeCycleOntology.owl.owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns="http://localhost/LifeCycleOntology.owl#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
  xml:base="http://localhost/LifeCycleOntology.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="visualisation">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="derivedData"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="workflowDokument">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:someValuesFrom>
          <owl:Class rdf:ID="taskSelection"/>
        </owl:someValuesFrom>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="isWorkflowDocumentOf"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Class rdf:ID="backgroundData"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="participant"/>
  <owl:Class rdf:about="#backgroundData">
    <rdfs:subClassOf rdf:resource="#participant"/>
  </owl:Class>
  <owl:Class rdf:ID="resourceFile">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="isResourceFileOf"/>
        </owl:onProperty>
        <owl:someValuesFrom>
          <owl:Class rdf:ID="dataPreparation"/>
        </owl:someValuesFrom>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="#backgroundData"/>
  </owl:Class>
  <owl:Class rdf:ID="domainOntology">
    <rdfs:subClassOf rdf:resource="#backgroundData"/>
  </owl:Class>
  <owl:Class rdf:ID="subWorkflow">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="workflow"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="dataMiningOntology">
    <rdfs:subClassOf rdf:resource="#backgroundData"/>
  </owl:Class>
  <owl:Class rdf:ID="student">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="person"/>
    </rdfs:subClassOf>
  </owl:Class>
```

```
<owl:Class rdf:ID="goalSpecification">
  <owl:disjointWith>
    <owl:Class rdf:about="#dataPreparation"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:ID="resultPublishing"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:ID="taskExecution"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:about="#taskSelection"/>
  </owl:disjointWith>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="hasDescriptionData"/>
      </owl:onProperty>
      <owl:someValuesFrom>
        <owl:Class rdf:ID="descriptionData"/>
      </owl:someValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="hasReferenceTo"/>
      </owl:onProperty>
      <owl:someValuesFrom rdf:resource="#domainOntology"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Class rdf:ID="activity"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#taskSelection">
  <rdfs:subClassOf>
```

```
      <owl:Restriction>
        <owl:someValuesFrom>
          <owl:Class rdf:about="#workflow"/>
        </owl:someValuesFrom>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="hasWorkflow"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="hasWorkflowDocument"/>
        </owl:onProperty>
        <owl:someValuesFrom rdf:resource="#workflowDokument"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="#activity"/>
    <owl:disjointWith>
      <owl:Class rdf:about="#dataPreparation"/>
    </owl:disjointWith>
    <owl:disjointWith rdf:resource="#goalSpecification"/>
    <owl:disjointWith>
      <owl:Class rdf:about="#resultPublishing"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="#taskExecution"/>
    </owl:disjointWith>
  </owl:Class>
  <owl:Class rdf:ID="lifeCycle">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:FunctionalProperty rdf:ID="hasGoalSpecification"/>
        </owl:onProperty>
        <owl:maxCardinality rdf:datatype="
        http://www.w3.org/2001/XMLSchema#int"
        >1</owl:maxCardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:maxCardinality rdf:datatype="
        http://www.w3.org/2001/XMLSchema#int"
        >1</owl:maxCardinality>
        <owl:onProperty>
          <owl:FunctionalProperty rdf:ID="hasDataPreparation"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:FunctionalProperty rdf:ID="hasTaskSelection"/>
        </owl:onProperty>
        <owl:maxCardinality rdf:datatype="
        http://www.w3.org/2001/XMLSchema#int"
        >1</owl:maxCardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:FunctionalProperty rdf:ID="hasTaskExecution"/>
        </owl:onProperty>
        <owl:maxCardinality rdf:datatype="
        http://www.w3.org/2001/XMLSchema#int"
        >1</owl:maxCardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:FunctionalProperty rdf:ID="hasResultPublishing"/>
        </owl:onProperty>
        <owl:maxCardinality rdf:datatype="
        http://www.w3.org/2001/XMLSchema#int"
        >1</owl:maxCardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="
      http://www.w3.org/2002/07/owl#Thing"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:allValuesFrom>
          <owl:Class rdf:ID="scientist"/>
        </owl:allValuesFrom>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="isExecutedBy"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:someValuesFrom>
          <owl:Class rdf:ID="publicationMode"/>
        </owl:someValuesFrom>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="isPublishedBy"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="researchDomain">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="#hasDescriptionData"/>
        </owl:onProperty>
        <owl:someValuesFrom>
          <owl:Class rdf:about="#descriptionData"/>
        </owl:someValuesFrom>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="
      http://www.w3.org/2002/07/owl#Thing"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:someValuesFrom rdf:resource="#domainOntology"/>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="hasDomainOntologyReference"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:someValuesFrom>
          <owl:Class rdf:about="#scientist"/>
        </owl:someValuesFrom>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="isDomainOf"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="serviceType">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="service"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="#taskExecution">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:someValuesFrom>
          <owl:Class rdf:ID="pmmlDocument"/>
        </owl:someValuesFrom>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="hasPMMLDocument"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:someValuesFrom rdf:resource="#visualisation"/>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="hasVisualisation"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <owl:disjointWith rdf:resource="#goalSpecification"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:someValuesFrom>
          <owl:Class rdf:ID="reportFile"/>
        </owl:someValuesFrom>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="hasReportFile"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:someValuesFrom>
          <owl:Class rdf:about="#descriptionData"/>
        </owl:someValuesFrom>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="#hasDescriptionData"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <owl:disjointWith>
      <owl:Class rdf:about="#resultPublishing"/>
    </owl:disjointWith>
    <rdfs:subClassOf rdf:resource="#activity"/>
    <owl:disjointWith rdf:resource="#taskSelection"/>
    <owl:disjointWith>
      <owl:Class rdf:about="#dataPreparation"/>
    </owl:disjointWith>
  </owl:Class>
  <owl:Class rdf:about="#reportFile">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="isReportFileOf"/>
        </owl:onProperty>
        <owl:someValuesFrom rdf:resource="#taskExecution"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Class rdf:about="#derivedData"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="#pmmlDocument">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:someValuesFrom rdf:resource="#taskExecution"/>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="isPMMLDocumentOf"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Class rdf:about="#derivedData"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="#derivedData">
    <rdfs:subClassOf rdf:resource="#participant"/>
  </owl:Class>
  <owl:Class rdf:ID="dataSet">
    <rdfs:subClassOf rdf:resource="#derivedData"/>
    <rdfs:subClassOf>
      <owl:Restriction>
```

```
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="hasPrimaryData"/>
      </owl:onProperty>
      <owl:someValuesFrom>
        <owl:Class rdf:ID="dataResource"/>
      </owl:someValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="isDataSetOf"/>
      </owl:onProperty>
      <owl:someValuesFrom>
        <owl:Class rdf:about="#dataPreparation"/>
      </owl:someValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#dataResource">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="usedToGenerate"/>
      </owl:onProperty>
      <owl:someValuesFrom rdf:resource="#dataSet"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf rdf:resource="#participant"/>
</owl:Class>
<owl:Class rdf:ID="dataStatictics">
  <rdfs:subClassOf rdf:resource="#backgroundData"/>
</owl:Class>
<owl:Class rdf:about="#descriptionData">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:someValuesFrom rdf:resource="#taskExecution"/>
```

```
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="describes"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="#describes"/>
        </owl:onProperty>
        <owl:someValuesFrom>
          <owl:Class rdf:about="#service"/>
        </owl:someValuesFrom>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:someValuesFrom>
          <owl:Class rdf:ID="researchGroup"/>
        </owl:someValuesFrom>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="#describes"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="#describes"/>
        </owl:onProperty>
        <owl:someValuesFrom>
          <owl:Class rdf:about="#workflow"/>
        </owl:someValuesFrom>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
```

```
        <owl:someValuesFrom>
          <owl:Class rdf:about="#dataPreparation"/>
        </owl:someValuesFrom>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="#describes"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="#describes"/>
        </owl:onProperty>
        <owl:someValuesFrom rdf:resource="#goalSpecification"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:someValuesFrom rdf:resource="#researchDomain"/>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="#describes"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="
      http://www.w3.org/2002/07/owl#Thing"/>
  </owl:Class>
  <owl:Class rdf:about="#publicationMode">
    <rdfs:subClassOf rdf:resource="
      http://www.w3.org/2002/07/owl#Thing"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="hasPublicationMode"/>
        </owl:onProperty>
        <owl:someValuesFrom>
          <owl:Class rdf:about="#researchGroup"/>
        </owl:someValuesFrom>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="#hasPublicationMode"/>
        </owl:onProperty>
        <owl:someValuesFrom>
          <owl:Class rdf:ID="colllaboration"/>
        </owl:someValuesFrom>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="#hasPublicationMode"/>
        </owl:onProperty>
        <owl:someValuesFrom rdf:resource="#researchDomain"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:someValuesFrom rdf:resource="#person"/>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="#hasPublicationMode"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="#scientist">
    <rdfs:subClassOf rdf:resource="#person"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:someValuesFrom>
          <owl:Class rdf:about="#researchGroup"/>
        </owl:someValuesFrom>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="isMemberOf"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:someValuesFrom rdf:resource="#lifeCycle"/>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="execute"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="belongsToDomain"/>
        </owl:onProperty>
        <owl:someValuesFrom rdf:resource="#researchDomain"/>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="serviceOntology">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="isServiceOntologyOf"/>
        </owl:onProperty>
        <owl:someValuesFrom>
          <owl:Class rdf:about="#service"/>
        </owl:someValuesFrom>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="#backgroundData"/>
  </owl:Class>
  <owl:Class rdf:about="#workflow">
    <rdfs:subClassOf>
```

```
      <owl:Restriction>
        <owl:someValuesFrom rdf:resource="#taskSelection"/>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="isWorkflowOf"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:someValuesFrom rdf:resource="#descriptionData"/>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="#hasDescriptionData"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="
      http://www.w3.org/2002/07/owl#Thing"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="usesService"/>
        </owl:onProperty>
        <owl:someValuesFrom>
          <owl:Class rdf:about="#service"/>
        </owl:someValuesFrom>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="#dataPreparation">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="#hasDescriptionData"/>
        </owl:onProperty>
        <owl:someValuesFrom rdf:resource="#descriptionData"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="hasOGSADaiResourceFile"/>
        </owl:onProperty>
        <owl:someValuesFrom rdf:resource="#resourceFile"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:someValuesFrom rdf:resource="#dataSet"/>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="hasDataSet"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="#activity"/>
    <owl:disjointWith rdf:resource="#goalSpecification"/>
    <owl:disjointWith>
      <owl:Class rdf:about="#resultPublishing"/>
    </owl:disjointWith>
    <owl:disjointWith rdf:resource="#taskExecution"/>
    <owl:disjointWith rdf:resource="#taskSelection"/>
  </owl:Class>
  <owl:Class rdf:about="#resultPublishing">
    <owl:disjointWith rdf:resource="#dataPreparation"/>
    <owl:disjointWith rdf:resource="#goalSpecification"/>
    <owl:disjointWith rdf:resource="#taskExecution"/>
    <owl:disjointWith rdf:resource="#taskSelection"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:someValuesFrom rdf:resource="#visualisation"/>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="#hasVisualisation"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="#activity"/>
  </owl:Class>
  <owl:Class rdf:about="#colllaboration">
    <rdfs:subClassOf rdf:resource="
      http://www.w3.org/2002/07/owl#Thing"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="hasCollaborativeGroup"/>
        </owl:onProperty>
        <owl:someValuesFrom>
          <owl:Class rdf:about="#researchGroup"/>
        </owl:someValuesFrom>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="#service">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="#hasDescriptionData"/>
        </owl:onProperty>
        <owl:someValuesFrom rdf:resource="#descriptionData"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="
      http://www.w3.org/2002/07/owl#Thing"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:someValuesFrom rdf:resource="#workflow"/>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="isUsedFor"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="#researchGroup">
```

```
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="#hasDescriptionData"/>
        </owl:onProperty>
        <owl:someValuesFrom rdf:resource="#descriptionData"/>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="
      http://www.w3.org/2002/07/owl#Thing"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:someValuesFrom rdf:resource="#colllaboration"/>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="inCollaborationWith"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="subDomain">
    <rdfs:subClassOf rdf:resource="#researchDomain"/>
  </owl:Class>
  <owl:ObjectProperty rdf:ID="isUsedBy"/>
  <owl:ObjectProperty rdf:ID="hasOutput"/>
  <owl:ObjectProperty rdf:about="#hasWorkflow">
    <rdfs:domain rdf:resource="#taskSelection"/>
    <owl:inverseOf>
      <owl:ObjectProperty rdf:about="#isWorkflowOf"/>
    </owl:inverseOf>
    <rdfs:range rdf:resource="#workflow"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="hasMembers">
    <rdfs:range rdf:resource="#scientist"/>
    <rdfs:domain rdf:resource="#researchGroup"/>
    <owl:inverseOf>
      <owl:ObjectProperty rdf:about="#isMemberOf"/>
    </owl:inverseOf>
```

```
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#isServiceOntologyOf">
  <owl:inverseOf>
    <owl:ObjectProperty rdf:ID="hasServiceOntology"/>
  </owl:inverseOf>
  <rdfs:range rdf:resource="#service"/>
  <rdfs:domain rdf:resource="#serviceOntology"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="isDerivedFrom"/>
<owl:ObjectProperty rdf:ID="containsActivity"/>
<owl:ObjectProperty rdf:ID="usesPublishMode">
  <owl:inverseOf>
    <owl:ObjectProperty rdf:ID="hasPublishMode"/>
  </owl:inverseOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#usesService">
  <rdfs:range rdf:resource="#service"/>
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#isUsedFor"/>
  </owl:inverseOf>
  <rdfs:domain rdf:resource="#workflow"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasReferenceTo">
  <rdfs:range rdf:resource="#domainOntology"/>
  <rdfs:domain rdf:resource="#goalSpecification"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#isMemberOf">
  <rdfs:domain rdf:resource="#scientist"/>
  <rdfs:range rdf:resource="#researchGroup"/>
  <owl:inverseOf rdf:resource="#hasMembers"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasPublishMode">
  <owl:inverseOf rdf:resource="#usesPublishMode"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#isResourceFileOf">
  <rdfs:range rdf:resource="#dataPreparation"/>
  <owl:inverseOf>
```

```
    <owl:ObjectProperty rdf:about="#hasOGSADaiResourceFile"/>
  </owl:inverseOf>
  <rdfs:domain rdf:resource="#resourceFile"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasVisualisation">
  <owl:inverseOf>
    <owl:ObjectProperty rdf:ID="isVisualisationOf"/>
  </owl:inverseOf>
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#resultPublishing"/>
        <owl:Class rdf:about="#taskExecution"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdfs:range rdf:resource="#visualisation"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#execute">
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#isExecutedBy"/>
  </owl:inverseOf>
  <rdfs:domain rdf:resource="#scientist"/>
  <rdfs:range rdf:resource="#lifeCycle"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#isReportFileOf">
  <rdfs:domain rdf:resource="#reportFile"/>
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#hasReportFile"/>
  </owl:inverseOf>
  <rdfs:range rdf:resource="#taskExecution"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#isPublishedBy">
  <rdfs:domain rdf:resource="#lifeCycle"/>
  <rdfs:range rdf:resource="#publicationMode"/>
  <owl:inverseOf>
    <owl:ObjectProperty rdf:ID="isPublicationModeOf"/>
  </owl:inverseOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasPublicationMode">
  <rdfs:domain rdf:resource="#publicationMode"/>
  <rdfs:range>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#researchDomain"/>
        <owl:Class rdf:about="#researchGroup"/>
        <owl:Class rdf:about="#colllaboration"/>
        <owl:Class rdf:about="#person"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:range>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#usedToGenerate">
  <rdfs:range rdf:resource="#dataSet"/>
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#hasPrimaryData"/>
  </owl:inverseOf>
  <rdfs:domain rdf:resource="#dataResource"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="isOutputOf"/>
<owl:ObjectProperty rdf:about="#isWorkflowOf">
  <rdfs:domain rdf:resource="#workflow"/>
  <owl:inverseOf rdf:resource="#hasWorkflow"/>
  <rdfs:range rdf:resource="#taskSelection"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="setsRestriction"/>
<owl:ObjectProperty rdf:about="#isDataSetOf">
  <rdfs:range rdf:resource="#dataPreparation"/>
  <rdfs:domain rdf:resource="#dataSet"/>
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#hasDataSet"/>
  </owl:inverseOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#isVisualisationOf">
  <rdfs:domain rdf:resource="#visualisation"/>
  <rdfs:range>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#resultPublishing"/>
        <owl:Class rdf:about="#taskExecution"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:range>
  <owl:inverseOf rdf:resource="#hasVisualisation"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="include"/>
<owl:ObjectProperty rdf:about="#isExecutedBy">
  <rdfs:range rdf:resource="#scientist"/>
  <owl:inverseOf rdf:resource="#execute"/>
  <rdfs:domain rdf:resource="#lifeCycle"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasResearchDomain"/>
<owl:ObjectProperty rdf:ID="belongsToActivity"/>
<owl:ObjectProperty rdf:about="#isUsedFor">
  <rdfs:domain rdf:resource="#service"/>
  <owl:inverseOf rdf:resource="#usesService"/>
  <rdfs:range rdf:resource="#workflow"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasCollaborativeGroup">
  <rdfs:range rdf:resource="#researchGroup"/>
  <rdfs:domain rdf:resource="#colllaboration"/>
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#inCollaborationWith"/>
  </owl:inverseOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasDataSet">
  <rdfs:domain rdf:resource="#dataPreparation"/>
  <rdfs:range rdf:resource="#dataSet"/>
  <owl:inverseOf rdf:resource="#isDataSetOf"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasWorkflowDocument">
```

```
  <rdfs:domain rdf:resource="#taskSelection"/>
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#isWorkflowDocumentOf"/>
  </owl:inverseOf>
  <rdfs:range rdf:resource="#workflowDokument"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasReport"/>
<owl:ObjectProperty rdf:ID="isSetBy"/>
<owl:ObjectProperty rdf:ID="hasPmmlReference"/>
<owl:ObjectProperty rdf:about="#hasReportFile">
  <owl:inverseOf rdf:resource="#isReportFileOf"/>
  <rdfs:domain rdf:resource="#taskExecution"/>
  <rdfs:range rdf:resource="#reportFile"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasServiceOntology">
  <owl:inverseOf rdf:resource="#isServiceOntologyOf"/>
  <rdfs:domain rdf:resource="#service"/>
  <rdfs:range rdf:resource="#serviceOntology"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasPrimaryData">
  <rdfs:domain rdf:resource="#dataSet"/>
  <rdfs:range rdf:resource="#dataResource"/>
  <owl:inverseOf rdf:resource="#usedToGenerate"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#isWorkflowDocumentOf">
  <rdfs:domain rdf:resource="#workflowDokument"/>
  <rdfs:range rdf:resource="#taskSelection"/>
  <owl:inverseOf rdf:resource="#hasWorkflowDocument"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasPMMLDocument">
  <rdfs:range rdf:resource="#pmmlDocument"/>
  <rdfs:domain rdf:resource="#taskExecution"/>
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#isPMMLDocumentOf"/>
  </owl:inverseOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasOGSADaiResourceFile">
```

```
    <rdfs:range rdf:resource="#resourceFile"/>
    <rdfs:domain rdf:resource="#dataPreparation"/>
    <owl:inverseOf rdf:resource="#isResourceFileOf"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="hasBackgroundData"/>
  <owl:ObjectProperty rdf:ID="hasRestriction"/>
  <owl:ObjectProperty rdf:about="#isDomainOf">
    <owl:inverseOf>
      <owl:ObjectProperty rdf:about="#belongsToDomain"/>
    </owl:inverseOf>
    <rdfs:range rdf:resource="#scientist"/>
    <rdfs:domain rdf:resource="#researchDomain"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#belongsToDomain">
    <rdfs:domain rdf:resource="#scientist"/>
    <owl:inverseOf rdf:resource="#isDomainOf"/>
    <rdfs:range rdf:resource="#researchDomain"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#isPublicationModeOf">
    <rdfs:range rdf:resource="#lifeCycle"/>
    <rdfs:domain rdf:resource="#publicationMode"/>
    <owl:inverseOf rdf:resource="#isPublishedBy"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#isPMMLDocumentOf">
    <owl:inverseOf rdf:resource="#hasPMMLDocument"/>
    <rdfs:range rdf:resource="#taskExecution"/>
    <rdfs:domain rdf:resource="#pmmlDocument"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#describes">
    <rdfs:domain rdf:resource="#descriptionData"/>
    <owl:inverseOf>
      <owl:ObjectProperty rdf:about="#hasDescriptionData"/>
    </owl:inverseOf>
    <rdfs:range>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#dataPreparation"/>
          <owl:Class rdf:about="#researchDomain"/>
          <owl:Class rdf:about="#goalSpecification"/>
          <owl:Class rdf:about="#workflow"/>
          <owl:Class rdf:about="#service"/>
          <owl:Class rdf:about="#researchGroup"/>
          <owl:Class rdf:about="#taskExecution"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:range>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#inCollaborationWith">
    <rdfs:domain rdf:resource="#researchGroup"/>
    <owl:inverseOf rdf:resource="#hasCollaborativeGroup"/>
    <rdfs:range rdf:resource="#colllaboration"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="worksIn"/>
  <owl:ObjectProperty rdf:about="#hasDescriptionData">
    <rdfs:range rdf:resource="#descriptionData"/>
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#goalSpecification"/>
          <owl:Class rdf:about="#researchDomain"/>
          <owl:Class rdf:about="#dataPreparation"/>
          <owl:Class rdf:about="#workflow"/>
          <owl:Class rdf:about="#service"/>
          <owl:Class rdf:about="#researchGroup"/>
          <owl:Class rdf:about="#taskExecution"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
    <owl:inverseOf rdf:resource="#describes"/>
  </owl:ObjectProperty>
  <owl:DatatypeProperty rdf:ID="description">
    <rdfs:domain rdf:resource="#descriptionData"/>
    <rdfs:range rdf:resource="
    http://www.w3.org/2001/XMLSchema#string"/>
```

```
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="dataSourceName">
  <rdfs:domain rdf:resource="#dataResource"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="serviceOntologyReference">
  <rdfs:domain rdf:resource="#serviceOntology"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="state">
  <rdfs:domain rdf:resource="#person"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="resourceFileReference">
  <rdfs:domain rdf:resource="#resourceFile"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="lastName">
  <rdfs:domain rdf:resource="#person"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="reportURL">
  <rdfs:domain rdf:resource="#reportFile"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="visualisationName">
  <rdfs:domain rdf:resource="#visualisation"/>
  <rdfs:range rdf:resource="
   http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="workflowReference">
  <rdfs:domain rdf:resource="#workflow"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="age">
  <rdfs:domain rdf:resource="#person"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="pmmlReference">
  <rdfs:domain rdf:resource="#pmmlDocument"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="phone">
  <rdfs:domain rdf:resource="#scientist"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="resourceURI">
```

```
  <rdfs:domain rdf:resource="#dataResource"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="resourceFileDescription">
  <rdfs:domain rdf:resource="#resourceFile"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="birthDate">
  <rdfs:domain rdf:resource="#person"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="firstName">
  <rdfs:domain rdf:resource="#person"/>
  <rdfs:range rdf:resource="
   http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="homepage">
  <rdfs:domain rdf:resource="#person"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="title">
  <rdfs:domain rdf:resource="#person"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="email">
  <rdfs:domain rdf:resource="#person"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="visualisationReference">
  <rdfs:domain rdf:resource="#visualisation"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="Name">
  <rdfs:domain rdf:resource="#descriptionData"/>
  <rdfs:range rdf:resource="
  http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="keywords">
  <rdfs:range rdf:resource="
  http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#descriptionData"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="workflowName">
  <rdfs:range rdf:resource="
  http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#workflow"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="visualisationType">
  <rdfs:domain rdf:resource="#visualisation"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="country"/>
<owl:FunctionalProperty rdf:about="#hasResultPublishing">
  <rdfs:domain rdf:resource="#lifeCycle"/>
  <rdf:type rdf:resource="
  http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:range rdf:resource="#resultPublishing"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:about="#hasTaskExecution">
  <rdfs:range rdf:resource="#taskExecution"/>
  <rdfs:domain rdf:resource="#lifeCycle"/>
  <rdf:type rdf:resource="
  http://www.w3.org/2002/07/owl#ObjectProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:about="#hasGoalSpecification">
  <rdfs:range rdf:resource="#goalSpecification"/>
  <rdf:type rdf:resource="
  http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:domain rdf:resource="#lifeCycle"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:about="#hasTaskSelection">
  <rdfs:range rdf:resource="#taskSelection"/>
  <rdfs:domain rdf:resource="#lifeCycle"/>
  <rdf:type rdf:resource="
  http://www.w3.org/2002/07/owl#ObjectProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:about="#hasDataPreparation">
  <rdf:type rdf:resource="
  http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:domain rdf:resource="#lifeCycle"/>
  <rdfs:range rdf:resource="#dataPreparation"/>
</owl:FunctionalProperty>
```

```
</rdf:RDF>
```

Appendix B

Zusammenfassung

Wissenschaftliche Daten stammen aus verschiedenen Wissens- und Forschungsgebieten, wobei durch die Sammlung dieser Daten große Datenbestände entstehen. Diese oft heterogenen und geographisch verteilten Datenmengen befinden sich auf verschiedenen Serversystemen und werden im Forschungsbereich der e-Science unter dem Begriff *primary data* zusammengefasst.
Forscher aus der ganzen Welt verwenden diese rohen Daten im Zuge vieler wissenschaftlicher Applikationen für ihre Forschungsexperimente und Analysen um signifikantes Wissen aus erzielten Forschungsergebnissen zu gewinnen. Der Output dieser Datenanalysen und Experimente, auch *derived data* genannt, ist das Ergebnis vieler rechenintensiver Prozesse und Aktivitäten, deren Organisation, für eine kooperative Weiterforschung innerhalb der beteiligten Forschungsgruppen und das Wiederverwenden der erzielten Ergebnisse, von enormer Bedeutung ist.
Das Hauptziel dieser Arbeit ist es, die Beziehungen zwischen den oben genannten *primary* und *derived data* in Form einer Ontologie zu beschreiben und somit ein intelligentes Datenmodell, welches ein typisches Forschungsscenario in der wissenschaftlichen Wissensgewinnung darstellt, definiert. Dieses Datenmodell repräsentiert einen sogenannten Dataspace, welcher aus einer Menge von forschungsrelevanten Datenelementen (Participants) und ihren Beziehung (Relationships) besteht und somit eine einheitliche Sicht auf diese Menge von heterogenen Daten ermöglicht.
Die Grundlage dieser Ontologie, welche in OWL (Ontology Web Language) implementiert wurde, ist das sogenannte *e-Science Life Cycle Dataspace Model*, welches aus fünf verschiedenen Aktivitäten, dargestellt in Form eines e-Science Lebenszyklus, besteht. Dieser Lebenszyklus beschreibt den Datenerfassungsprozess innerhalb eines wissenschaftlichen Forschungsscenarios, ausgehend von der einfachen Zielsetzung bis hin zur Datenintegration und Ergebnispublikation.

Appendix C

Lebenslauf

Persönliche Daten:

Name:	Muslimovic Adnan
Geburtsdatum:	17.09.1977
Geburtsort:	Prijedor, Bosnien und Herzegowina
Adresse:	1160 Wien, Blumberggasse 20/13
Staatsangehörigkeit:	Bosnien
Familienstand:	ledig
Tel:	0650/9903450
E-mail:	a9903450@unet.univie.ac.at

Ausbildung:

1984-92	Volks-/Hauptschule, Prijedor
seit Okt.1992 in Österreich	
1992-93	HS 5, Linz
1993-99	BORG, Honauerstraße 24, 4020 Linz
	Informatikzweig, Reifeprüfung (15.06.99)
1999-08	Wirtschaftsinformatikstudium an der Universität Wien

Besondere Kenntnisse und Fähigkeiten:
Deutsch: fließend
Bosnisch, Kroatisch, Serbisch: fließend
Englisch: in Wort und Schrift
Programmiersprachen: C++, JAVA, Delphi
EDV bzw. Betriebssysteme: MS Office, Windows, Linux

Bibliography

[AAS07] Yair Amit, Danny Allan, and Adi Sharabani, *Overtaking google desktop - a security analysis*, Watchfire Whitepaper, 2007. 56

[ACD04] Nagrai Alur, YunJung Chang, and Barry Devlin, *Information aggregation and data integration with db2 information integrator*, IBM Redbook, 2004. 50, 51, 52, 54

[Ale08] S. Aleksejevs, *Contribution model*, http://wiki.myexperiment.org/index.php/Developer:Contribution_Model, 2008. 62

[Ame01] The Scientific American, *The semantic web*, http://www.sciam.com/article.cfm?id=the-semantic-web&page=3, 2001. 6

[Asc03] John Aschoff, *Websphere information integrator q replication*, IBM Technical Article, 2003. 54, 55

[AvH04] G. Antoniou and F. van Harmelen., *A semantic web primer*, The MIT Press Cambridge, 2004. 4, 15, 17

[BAB03] Paolo Bruni, Francis Arnaudies, and Amanda Bennett, *Data federation with ibm db2 information integrator v8.1*, IBM Redbook, 2003. 50, 52, 53, 54, 55, 64

[BB08] D. Beckett and J. Broekstra, *Sparql query results xml format*, http://www.w3.org/TR/rdf-sparql-XMLres/, 2008. 26

[BDG$^+$07] Lukas Blunschi, Jens-Peter Dittrich, Olivier René Girard, Shant Kirakos Karakashian, and Marcos Antonio Vaz Salles, *A dataspace odyssey: The imemex personal dataspace management system*, CIDR 2007, Third Biennial Conference on Innovative Data Systems Research, January 2007, pp. 114–119. 32, 64, 68

[Bec04] Dave Beckett, *Rdf/xml syntax specification (revised)*, http://www.w3.org/TR/rdf-syntax-grammar/, 2004. 5, 11, 12

[BG04] D. Brickley and R.V. Guha, *Rdf vocabulary description language 1.0: Rdf schema*, http://www.w3.org/TR/rdf-schema/, 2004. 5, 13

[BHLT06] T. Bray, D. Hollander, A. Layman, and R. Tobin, *Namespaces in xml 1.0*, http://www.w3.org/TR/REC-xml-names/, 2006. 8

[BL00] Tim Berners-Lee, *A layered approach of semantic web*, http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html, 2000. 6

[BMRW98] Chaitanya Baru, Reagan Moore, Arcot Rajasekar, and Michael Wan, *The sdsc storage resource broker*, Proceedings of the 1998 conference of the Centre for Advanced Studies on Collaborative research (CASCON '98), 1998, p. 5. 38, 39, 40, 41, 64

[BPM04] P. V. Biron, K. Permanente, and A. Malhotra, *Xml schema part 2: Datatypes second edition*, http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/datatypes.html, 2004. 10

[BPSM$^+$06] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau, *Extensible markup language (xml) 1.0 (fourth edition)*, http://www.w3.org/TR/xml/, 2006. 5, 7, 9

[Bus45] Vannevar Bush, *As we may think*, Atlantic Monthly, http://www.theatlantic.com/doc/194507/bush/, July 1945. 32

[BvHH$^+$04] S. Bechhofer, F. van Harmelen, J. Hendlerand, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein, *Owl web ontology language reference*, http://www.w3.org/TR/owl-ref/, 2004. 5, 17, 18, 20, 23

[CAD07] CADGrid, *The China-Austria Data Grid project*, http://www.par. univie.ac.at/project/cadgrid, 2007. 102

[CFT08] K. G. Clark, L. Feigenbaum, and E. Torres, *Sparql protocol for rdf*, http://www.w3.org/TR/rdf-sparql-protocol/, 2008. 26

[CRDS06] E. Cutrell, D.C. Robbins, S.T Dumais, and R Sarin, *Fast, flexible filtering with phlat - personal search and organization made easy*, In proceedings of international conference for human-computer interaction, CHI 2006, 2006. 58

[Cru08] D. Cruickshank, *myexperiment entities*, http://wiki.myexperiment.org, 2008. 62

[Dat08] Data Mining Group, *The Predictive Model Markup Language (PMML)*, http://www.dmg.org/v3-2/, July 2008. 70, 77

[Dit06] Jens-Peter Dittrich, *imemex: A platform for personal dataspace management*, In Proceedings of the 2nd SIGIR Workshop on Personal Information Management (PIM 2006), August 2006. 32, 35

[Dod05] L. Dodds, *Introducing sparql: Querying the semantic web*, www.xml.com/pub/a/2005/11/16/introducing-sparql-querying-semantic-web-tutorial.html, 2005. 28

[DS06] Jens-Peter Dittrich and Marcos Antonio Vaz Salles, *idm: a unified and versatile data model for personal dataspace management*, VLDB '06: Proceedings of the 32nd international conference on Very large data bases, 2006, pp. 367–378. 33, 35, 36

[EBT06] Ibrahim Elsayed, Peter Brezany, and A Min Tjoa, *Towards realization of dataspaces*, Proceedings of International Conference on Database and Expert Systems Applications (DEXA), 2006. 30, 64

[EHL$^+$08] I. Elsayed, J. Han, T. Liu, A. Wöhrer, F. A. Khan, and P. Brezany, *Grid-enabled non-invasive blood glucose measurement*, Springer Berlin, 2008. 103, 104

[Erf05] Farnaz Erfan, *Maintain federated data using websphere information integrator autonomic monitoring tools*, IBM, 2005. 49

[fBIR08] Stanford Center for Biomedical Informatics Research, *Protege owl plugin 3.3.1*, http://protege.stanford.edu/, 2008. 81

[fEe07a] National Institute for Environmental eScience, *The niees wiki for grid information*, http://gridinfo.niees.ac.uk/index.php/ Storage_Resource_Brocker, 2007. 44

[fEe07b] ———, *The niees wiki for grid information*, http://gridinfo.niees.ac.uk/index.php/InQ, 2007. 43

[FHM05] M. Franklin, A. Halevy, and D. Maier, *From databases to dataspaces: A new abstraction for information management*, ACM SIGMOD, December 2005. 30, 31, 33, 36, 63, 111

[FHvH$^+$01] D. Fensel, I. Horrocks, F. van Harmelen, D. Mcguiness, P., and Patel-Schneider, *Oil: An ontology infrastructure for semantic web*, IEEE Inteligent Systems, 2001. 5, 6

[FVWZ02] I. Foster, J. Voeckler, M. Wilde, and Y. Zhao, *Chimera: Avirtual data system for representing, querying, and automating data derivation*, 14th International Conference on Scientific and Statistical Database Management, 2002. 60, 64

[FW04] D. C. Fallside and P. Walmsley, *Xml schema part 0: Primer second edition*, http://www.w3.org/TR/xmlschema-0/, 2004. 5

[GPFLC04] A. Gomez-Perez, M. Fernandez-Lopez, and O. Corcho, *Ontological engineering*, Springer, 2004. 82

[GR07] C. Goble and D. De Roure, *myexperiment: Social networking for workflow-using e-scientists*, Proceedings of the 2nd workshop on Workflows in support of large-scale science, Monterey, California, USA., 2007. 62, 64

[Gru08] Tom Gruber, *What is an ontology?*, http://www-ksl.stanford.edu/kst/what-is-an-ontology.html, 2008. 4

[Hal05] Alon Halevy, *Why your data won't mix*, Queue **3** (2005), 50–58. 30

[HFM06] A. Halevy, M. Franklin, and D. Maier, *Principles of dataspace systems*, Proceedings of the ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS), December 2006. 30, 67

[IEE08] IEEE, *The world's leading professional association for the advancement of technology*, http://www.ieee.org, 2008. 82

[Inc08] Google Inc., *Google desktop features*, http://desktop.google.com/en/features.html, 2008. 56

[iRG06a] iRODS Research Group, *The irods rule system*, https://www.irods.org/ index.php/The_iRODS_Rule_System, 2006. 47

[iRG06b] ______, *Rule-oriented programming*, https://www.irods.org/, 2006. 46

[iRG08a] ______, *Introduction to irods*, https://www.irods.org/index.php/Introduction_to_iRODS, 2008. 46

[iRG08b] ______, *irods v1.0*, https://www.irods.org/, 2008. 45, 48, 49

[KHB04] G. Kickinger, J. Hofer, and P. Brezany, *Grid knowledge discovery system processes and an architecture for their composition*, IASTED Conference, 2004. 78

[KM01] Marja-Riitta Koivunen and Eric Miller, *W3c semantic web activity*, http://www.w3.org/2001/12/semweb-fin/w3csw, 2001. 7

[M. 07] M. Antonioletti et al, *OGSA-DAI 3.0 - the whats and the whys*, Proceedings of the UK e-Science All Hands Meeting, September 2007. 69

[MM04] F. Manola and E. Miller, *Rdf primer*, http://www.w3.org/TR/rdf-primer/, 2004. 10

[MvH04] D. L. McGuinness and F. van Harmelen, *Owl web ontology language overview*, http://www.w3.org/TR/owl-features/, 2004. 15, 16

[nw08] n2 wiki, *Sparql*, http://n2.talis.com/wiki/SPARQL_intro, 2008. 26

[oMoS08] The University of Manchester and University of Southampton, *myexperiment main page*, http://www.myexperiment.org/, 2008. 63

[Pal01] Sean B. Palmer, *The semantic web, an introduction*, http:// infomesh.net/2001/swintro/, 2001. 6

[PS08] E. Prud'hommeaux and A. Seaborne, *Sparql query language for rdf*, http://www.w3.org/TR/rdf-sparql-query/, 2008. 25, 26, 74

[RMC07] Joseph Rosenzweig, Rada Mihalcea, and Andras Csomai, *A machine-readable thesaurus and semantic network*, http://lit.csci.unt.edu/ wordnet/, 2007. 37

[RWM03a] A. Rajasekar, M. Wan, and R.W. Moore, *Mysrb and srb - components of a data grid*, San Diego Supercomputer Center, University of California at San Diego, 2003. 39

[RWM+03b] A. Rajasekar, M. Wan, R.W. Moore, W. Schroeder, G. Kremenek, A. Jagatheesan, C. Cowart, B. Zhu, S.Y Chen, and R. Olschanowsky, *Storage resource broker - managing distributed data in a grid*, Computer Society of India Journal, 2003. 42, 43, 44

[RWMS06] A. Rajasekar, M. Wan, R. Moore, and W. Schroeder, *A prototype rule-based distributed data management system*, HPDC workshop on "Next Generation Distributed Data Management, 2006. 45, 47, 48, 64

[SDK+07] Marcos Antonio Vaz Salles, Jens-Peter Dittrich, Shant Kirakos Karakashian, Olivier René Girard, and Lukas Blunschi, *itrails: pay-as-you-go information integration in dataspaces*, Proceedings of the 33rd international conference on Very large data bases (VLDB 2007), 2007, pp. 663–674. 36, 37

[SWM04] M. K. Smith, C. Welty, and D. L. McGuinness, *Owl web ontology language guide*, http://www.w3.org/TR/owl-guide/, 2004. 18

[TBJ08] A Min Tjoa, P. Brezany, and I. Janciak, *Gridminer: An advanced support for e-science analytics*, http://www.gridminer.org, 2008. 75

[TBMM04] H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn, *Xml schema part 1: Structures second edition*, http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/ structures.html, 2004. 10

[TBW+08] A Min Tjoa, P. Brezany, A. Wöhrer, I. Janciak, and I. Elsayed, *Gridminer - intelligent grid solutions*, http://www.gridminer.org/ index.php, 2008. 75

[Web08] W3C Semantic Web, *W3c semantic web faq*, http://www.w3.org/RDF/FAQ, 2008. 6

[Wik08a] Wikipedia, *Ontology (information science)*, http://en.wikipedia.org/wiki/Ontology, 2008. 4

[Wik08b] ______, *Ontology language*, http://en.wikipedia.org/wiki/Ontology_language, 2008. 4

[Wik08c] ______, *Owl - web ontology language*, http://en.wikipedia.org/wiki/Web_Ontology_Language, 2008. 14

[Wik08d] ______, *Xml*, http://en.wikipedia.org/wiki/XML, 2008. 7