Università degli Studi di Camerino

School of Advanced Studies

Dottorato di Ricerca in Scienze dell'Informazione
e Sistemi Complessi - XXII Ciclo

Settore Scientifico Disciplinare INF/01
Informatica

Dipartimento di Matematica e Informatica

A Semantic Framework for Declarative and Procedural Knowledge

Relatore

Prof. Flavio Corradini

Dottorando

Leonardo Vito

Commissione

Prof. Knut Hinkelmann

Dr. Pietro Lió

Prof. Sauro Longhi

Anno Accademico 2008-2009

Università degli Studi di Camerino

School of Advanced Studies

Doctor of Philosophy in Information Science And Complex Systems - XXII Cycle

Dipartimento di Matematica e Informatica

A Semantic Framework for Declarative and Procedural Knowledge

Advisor

Prof. Flavio Corradini

PhD Candidate

Leonardo Vito

Advisory Board

Prof. Knut Hinkelmann

Dr. Pietro Lió

Prof. Sauro Longhi

Academic Year 2008-2009

for those who work at the quality life improvement

Abstract of the Dissertation

In any scientific domain, the full set of data and programs has reached an *-ome* status, i.e. it has grown massively. The original article on the Semantic Web describes the evolution of a Web of *actionable information*, i.e. information derived from data through a semantic theory for interpreting the symbols. In a Semantic Web, methodologies are studied for describing, managing and analyzing both resources (domain knowledge) and applications (operational knowledge) - without any restriction on what and where they are respectively suitable and available in the Web - as well as for realizing automatic and semantic-driven workflows of Web applications elaborating Web resources.

This thesis attempts to provide a synthesis among Semantic Web technologies, Ontology Research, Knowledge and Workflow Management. Such a synthesis is represented by $\mathcal{R}esourceome$, a Web-based framework consisting of two components which strictly interact with each other: an ontology-based and domain-independent knowledge manager system ($\mathcal{R}esourceome$ KMS) - relying on a knowledge model where resource and operational knowledge are contextualized in *any* domain - and a semantic-driven workflow editor, manager and agent-based execution system ($\mathcal{R}esourceome$ WMS).

The $\mathcal{R}esourceome$ KMS and the $\mathcal{R}esourceome$ WMS are exploited in order to realize semantic-driven formulations of workflows, where activities are semantically linked to any involved resource. In the whole, combining the use of domain ontologies and workflow techniques, $\mathcal{R}esourceome$ provides a flexible domain and operational knowledge organization, a powerful engine for semantic-driven workflow composition, and a distributed, automatic and transparent environment for workflow execution.

Acknowledgements

Writing a thesis is a solitary affair that cannot be done alone. First of all, I wish to express my sincere gratitude to my advisor, Prof. Flavio Corradini, and my promoter, Prof.ssa Emanuela Merelli: this work would not have been possible without them. Their direction and support have been invaluable. Hours and hours of discussions, suggestions and disagreements allowed me to understand the field and the nature of research itself. Their encouragement throughout the years has helped me tremendously.

I would like to thank Dott.ssa Diletta Romana Cacciagrano for being involved in this work as much as she was, being a very active participant in this research all the time. Her suggestions and discussions have had a lot of influence on this work.

I would like to express my gratitude to all the current and former CoSy research group members, which have contributed with their knowledge and expertise to improve this project.

I am very grateful to Formal Methods research group members. In particular, I would like to thank Prof. Jos Baeten, at the University of Eindhoven where I spent seven months of my PhD . . . I cannot never forget their hospitality.

I am deeply grateful to the GRUPPO LOCCIONI for its collaboration and support in carrying out the $\mathcal{R}esourceome$ project.

Most importantly, my gratitude goes to my grandparents, for their love and support, for always being there for me, for always believing in me. Last but not least I would like to thank Cristiana.

List of Publications

[BCMV06] Ezio Bartocci, F. Corradini, E. Merelli, and L. Vito. Model driven design and implementation of activity-based applications in hermes. In F. De Paoli, A. Di Stefano, A. Omicini, and C. Santoro, editors, *WOA*, volume 204 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006.

[CCCTV] D. Cacciagrano, F. Corradini, R. Culmone, L. Tesei, and L. Vito. A model-prover for constrained dynamic conversations. In Gabriele Kotsis, David Taniar, Eric Pardede, and Ismail Khalil Ibrahim, editors, *iiWAS*, pages 630–633. ACM, 2008.

[CCCV] D. Cacciagrano, F. Corradini, R. Culmone, and L. Vito. Dynamic constraint-based invocation of web services. In Mario Bravetti, Manuel Núñez, and Gianluigi Zavattaro, editors, *WS-FM*, volume 4184 of *Lecture Notes in Computer Science*, pages 138–147. Springer, 2006.

[CCCV09] D. Cacciagrano, F. Corradini, Rosario Culmone, and L. Vito. Constraint-based dynamic conversations. *Networking and Services, International conference on*, 0:7–12, 2009.

[CCGLMPV07] N. Cannata, F. Corradini, S. Gabrielli, L. Leoni, E. Merelli, F. Piersigilli, and L. Vito. Intuitive and machine-understandable representation of the bioinformatics domain and of related resources with resourceomes. In *Proceedings of NETTAB 2007. A Semantic Web for Bioinformatics: Goals, Tools, Systems, Applications*, pages 35–46, 2007.

[CCMPV09] N. Cannata, F. Corradini, E. Merelli, F. Piersigilli, and L. Vito. Towards bioinformatics resourceomes. In *Biomedical Data and Applications, Studies in Computational Intelligence, ISBN: 978-3-642-02192-3*, page Chapter 1, 2009.

[CCMPV07] N. Cannata, F. Corradini, F. Piersigilli, E. Merelli, and L. Vito. Semantic resource management in mas. In Matteo Baldoni, Antonio Boccalatte, F. De Paoli, Maurizio Martelli, and Viviana Mascardi, editors, *WOA*, pages 42–47. Seneca Edizioni Torino, 2007.

[CCMV09] D. Cacciagrano, F. Corradini, E. Merelli, G. Romiti, and L. Vito. Resourceome: a multilevel model and a semantic web tool for managing domain and operational knowledge. In *Proceeding of SEMAPRO 09, Malta*, 2009.

[CMPRV09] F. Bartocci, C. Cacciagrano, F. Corradini, E. Merelli, and L. Vito. Ontology-driven automation of in-silico experiments. In *Proceeding of BIOSYSCOM 10, 2010 - Cancun, Mexico*, 2009.

Contents

List of Figures

Part I

Introduction

Chapter 1

Introduction

Today, knowledge is considered to be the most important resource in any domain. It can be defined as all of the information needed by a human being (or a machine), organized to carry out a task. However, how to find specific knowledge from the massive information shared on Internet is a difficult work, even if accessing it is more and more easily. More and more information is available on-line every day: the greater the amount of on-line information, the greater the demand for tools that process and disseminate this information. It is no longer possible just to navigate through gigabytes of data and programs on CD-ROMs and the World Wide Web.

Information searching is also a key issue in the knowledge sharing: correct informations will bring success and the others will cause failure. The possibility to share and reuse knowledge is central for any knowledge-based system and infrastructure. For this purpose, knowledge engineers and system analysts play the role of bringing knowledge forth and making it explicit: they display the implicit knowledge about a domain in a form that programmers can encode in algorithms and data structures.

1.1 Semantics in the Knowledge Representation

Conceptualization offers an abstract simplified view of a reality. Recent advances in knowledge representation, reasoning, computational linguistics, and other related fields have demonstrated the advantages of developing a reference *ontology* or set of component-based reference ontologies for a given domain.

Ontology is the term borrowed from the philosophy where the meaning is

predominantly on the study of being; in the computer science, ontology specifies the representational vocabulary with which knowledge-based systems represent knowledge. According to Guarino, ontology can be understood as an intentional semantic structure used to capture and encode the implicit rules constraining the structure.

Ontologies are aimed at answering the question: *What kinds of objects exist in one or another domain of the real world and how are they interrelated?* [11]. Thus, ontology is used to capture knowledge asset objects in the organization so that they can be reasoned about or be otherwise used and processed.

1.2 Motivations and Objectives

The original article on the Semantic Web [130] describes the evolution of a Web of *actionable* knowledge, i.e. knowledge derived from data through a semantic theory for interpreting the symbols.

There are numerous types of knowledge, but they are generally distinguished between *knowing something* (declarative or domain knowledge) and *knowing how do something* (procedural or operational knowledge): the first refers to our memory for concepts, facts, or episodes, whereas the latter refers to the ability to perform various tasks. Procedural knowledge may also begin as declarative knowledge and then be proceduralized with practice.

In a Semantic Web, methodologies are studied for describing, managing, analyzing and invoking both domain and operational knowledge - without any restriction on what and where it is suitable and available in the Web.

This thesis attempts to provide a synthesis among Semantic Web technologies, Ontology Research, Knowledge and Workflow Management, in according to the idea described in [130]. Such a synthesis is represented by $\mathcal{R}esourceome$, a Web-based framework consisting of two components which strictly interact with each other: an ontology-based and domain-independent knowledge manager system ($\mathcal{R}esourceome$ KMS) - relying on a knowledge model where resource and operational knowledge are contextualized in *any* domain - and a semantic-driven workflow editor, manager and agent-based execution system ($\mathcal{R}esourceome$ WMS). $\mathcal{R}esourceome$ proposes the combined use of domain ontologies and workflow techniques to realize a semantic guide for:

- a flexible knowledge organization, thanks to a well-structured ontologization;

- a semantic-driven workflow definition, thanks to a framework for the

assembly of (semantically) well-formed workflows from (semantically) heterogeneous resources;

- a distributed, automatic and transparent execution of workflow, thanks to an agent-oriented layer implementing a Migrating Workflow Model [200].

On the one hand, $\mathcal{R}esourceome$ KMS supports:

- the splitting of domain and resource concepts into two distinct ontologies – Domain Ontology and Resource Ontology – in such a way to formalize the resource conceptualization and then to contextualize its use in one or more specific domains;

- the definition of a well-structured Task Ontology for modeling the workflow activity space. This feature allows the use of $\mathcal{R}esourceome$ to support a high level, semantic-driven process specification language that is independent of any specific formalism (XPDL, BPMN, etc.).

On the other hand, $\mathcal{R}esourceome$ WMS provides a Web-based environment to manage the knowledge base, to define semantic-driven workflows and to execute them.

The main and innovative contribution of this work is not in the use of ontologies and Semantic Web technologies *on their own*, but *how* ontologies and Semantic Web technologies are used:

- $\mathcal{R}esourceome$ KMS differs from most existing semantic KMSs, each of which is well-typed for a specific domain. $\mathcal{R}esourceome$ knowledge model allows to conceptualize either specific resources or any other arbitrary resource space.

  The flexibility of this knowledge conceptualization is a consequence of keeping, at the same time, a physical separation and a logical interoperation among *domain*, *resource* and *activity* concepts: the knowledge space is partitioned in three ontologies (Domain, Resource and Task Ontologies) and appropriate relationships among their respective (most general) concepts have been defined in order to keep information for each activity about its execution context, the roles that perform it, any involved resource and interchanged information, a possible implementation code, its preconditions and effects.

- Also Web Services, as self-contained software components exposing specific functionalities on the Internet, are indexed simply as *resources* in the Resource Ontology and, for this reason, can be involved in a workflow defined by $\mathcal{R}esourceome$ WMS. In order to allow them to be automatically and properly invoked, $\mathcal{R}esourceome$ KMS has been enriched

with a theoretical framework - namely, two conversation specification language-independent models for *valid* client-server conversations - and a fully XML-based practical environment - namely, a parser associated to XML constrained grammar rules - in order to realizing at runtime *valid* sequences of *valid* Web Service invocations (where *valid* is intended w.r.t. message parameters and transitions among exchanged messages).

- $\mathcal{R}esourceome$ WMS can capture not only the experimental method but also constraints and goals associated to a workflow. Also this feature is a consequence of the $\mathcal{R}esourceome$ knowledge conceptualization flexibility, which can support both search and discovery of services, interoperation between them and, as a natural consequence, the realization of semantic-driven workflows.

  Moreover, $\mathcal{R}esourceome$ WMS also exploits the agent-oriented technology, allowing workflow engines coupled with single user experiment specifications to be dynamically generated. More precisely, the agent-based technology permits to embed application domain features inside the agent knowledge and proactiveness and mobility inside the agent behavior. The resulting workflow engine is a multiagent system, i.e. a distributed, concurrent system, typically open, flexible, adaptive and mobile.

1.2.1 Structure of the Thesis

The rest of the Thesis is organized as follows:

- Chapter 2 introduces the Semantic Web and describes the knowledge conceptualization by ontology, with some remarks about description logic, while Chapter 3 provides a brief introduction about $\mathcal{H}ermes$ middleware as a framework for distributed computation.

- Chapter 4 provides a semantic model for declarative and procedural knowledge representation called $\mathcal{R}esourceome$. The model, originally proposed in "Time to organize the bioinformatics resourceome" [34], is represented here using formal languages and is extended introducing a task ontology.

- Chapter 5 is devoted to discuss the issues related to a distributed workflow engine relying on agent technologies.

- Chapter 6 presents a formal model for describing, verifying and realizing dynamic invocations of Web Services, being Web Services one of the main resources available through Http protocol, as well as one of the newest approaches of distributed computing. In particular, a formal framework for valid client-server communication protocol is proposed and studied.

- Chapter 7 describes the $\mathcal{R}esourceome$ Knowledge Management lifecycle idea, that is on the basis of the $\mathcal{R}esourceome$ Web tool to manage declarative and procedural knowledge.

- Chapter 8 presents a possible application domain of $\mathcal{R}esourceome$, i.e. a biological scenario where domain experts (in this case bioinformatics) can manage biology resources and drive in-silico experiments.

- Finally, Chapter 9 ends the dissertation highlighting the main contributions of the conceptual and implemented framework described in this Thesis.

Part II

Background

Chapter 2

Semantic Web

The innovative value of ontology-based solutions stems from the factors that distinguish ontology-based methods from conventional baseline technologies already available in the traditional Kwonledge Management toolset, ranging from whiteboards, relational databases, expert systems to content classication solutions. In order to crystallize the potential of ontology-based methods, we first need to take a closer look at the fundamentals of ontologies. This will help us to identify the role and the scope of ontology use in KM applications and will lead us to a simple classication of ontology applications, namely the Semantic Web Tent.

2.1 Semantic Web

The Semantic Web is a project for the creation of a universal medium for information exchange by giving meaning (semantics), in a manner understandable by machines, to the content of documents on the Web. The Semantic Web is defined as "an extension of the current Web in which information is given well-defined meaning, better enabling computers and people to work in cooperation" [130]. It is a project aimed to make Web pages understandable by computers, so that they can search Web sites and perform actions in a standardized way.

The Semantic Web allows Web entities like software agents, users and programs, to interoperate, dynamically discovering and using resources and its metadata, extracting knowledge, and solving complex problems.

But as the concept of metadata could not yet be clear, here follows a definition.

Surfing the Web, generally people follow links bringing to resources iden-

tified by a URI. A resource is not an independent entity, but can be accompanied by information describing it. This machine-understandable information, generally called "metadata" can be used by software agents to properly use the resources. As metadata are data, they can be:

- Stored as data, in a resource, that can contain information about its own or another resource. A document metadata can be contained in the document or in another one.
- Described by other metadata, and so on.

Although metadata are needed to annotate and describe Web content and to allow machine-to-machine operation, it is necessary complex automated processing to give semantic meaning to each Web resource. Semantic description can also applied to the processes, for example represented as web services. Thanks to semantic web services can be discovered and automatically composed.

A layered model of the Semantic Web comprises:

- A set of Web resources, with a unique, global identity, described by metadata and with rules for inferring new metadata knowledge through ontologies.
- A set of basic services such as reasoners over metadata and ontologies.
- A set of high-level applications developed through basic services.

In the next section an overview of some of these semantic technologies will be presented.

2.1.1 Semantic Web Technologies

The Semantic Web can be built through the layers of Web technologies and standards. The layers are presented in Figure 2.1:

- The Unicode and Uniform Resource Identifier layers make sure that it is using international characters sets and providing means for identifying the objects in Semantic Web, respectively.
- The XML layer with namespace (NS) and schema definitions make sure the Semantic Web definitions can be integrated with the other XML based standards.

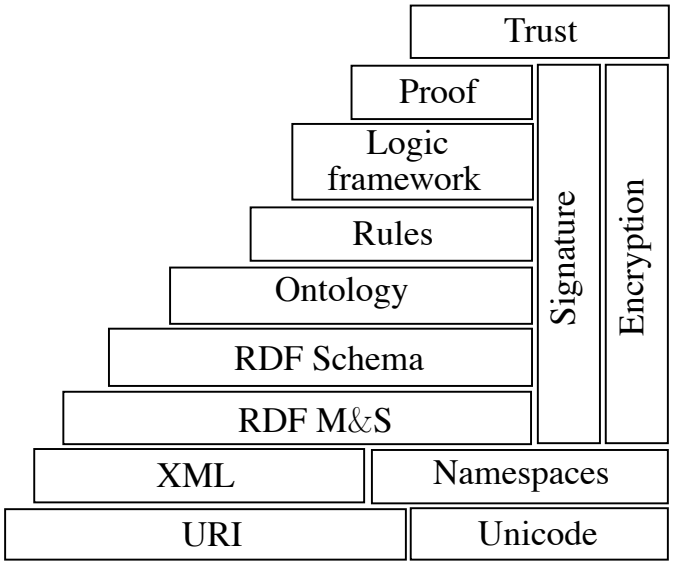

Figure 2.1: Semantic Web layers

- With the Resource Description Framework (RDF) (W3C standard) and RDFSchema (RDFS) it is possible to make statements about objects with URI and define vocabularies that can be referred to by URIs.

- The Ontology layer supports the evolution of vocabularies as it can define relations between different concepts (see also Section 2.2).

- A digital signature is an electronic signature that can be used to authenticate the identity of the sender of a message or the signer of a document. The Digital Signature layer ensures that the original content of the message or document is unaltered.

- The top layers Logic, Proof and Trust, are currently being researched. The Logic layer concerns the writing of rules, while the Proof layer executes the rules and evaluates together with the Trust layer mechanism for applications, whether to trust the given proof or not.

The main intent of the Semantic Web is to give machines autonomous access to information resources in order to support humans.
In the following sections we will introduce the basic technologies to build the Semantic Web.

2.2 Ontology

Knowledge Representation is developed as a branch of artificial intelligent and it applies theory and technique from *Logic* (providing formal structure and role of inference), *Ontology* (defining the kind of things that exist in the application domain) and *Computation* (supporting the application).

2.2.1 Philosophical Background

Ontology is the study of existence, of all entities that make up the world. Ontology as a branch of philosophy is the science of what is, of the kinds and structures of objects, properties, events, processes and relations in every area of reality. Philosophical ontology is what is standardly called *descriptive* ontology. It seeks not explanation but rather a description of reality in terms of a classification of entities that is exhaustive in the sense that it can serve as an answer to such questions as: (i)What classes of entities are needed for a complete description and explanation of all the goings-on in the universe?(ii)What classes of entities are needed to facilitate the making of predictions about the future? Sometimes a division is made between formal and material (or local) ontology. Formal ontology is domain-neutral; it deals with those aspects of reality (for example parthood and identity) which are shared in common by all material regions. Material ontology deals with those features which are specific to given domains.

2.2.2 Ontology in Information Science

The ontology-based approach has gained currency in recent years in the field of computer and information science; in particular, it has become popular especially in domains such as knowledge engineering, natural language processing, cooperative information systems, intelligent information integration, and knowledge management. In the world of information systems, an ontology is a software (or formal language) artefact designed with a specific set of uses and computational environments in mind. An ontology is often something that is ordered by a specific client in a specific context and in relation to specific practical needs and resources.

2.2.3 A formal approach

Definition 2.2.1 *An ontology is a tuple* $\mathscr{O} = (C, R, \leq, \perp, |, \sigma)$ *where:*

1. *C is a finite set of concept symbols;*

2. *R is a finite set of relation symbols;*

3. *$\leq$ is a reflexive, transitive and anti-symmetric relation on C (a partial order);*

4. *$\perp$ is a symmetric and irreflexive relation on C (disjointness);*

5. *$|$ is a symmetric relation on C (coverage); and*

6. *$\sigma : R \to C^+$ is the function assigning to each relation symbol its arity; the functor $(-)^+$ sends a set C to the set of finite tuples whose elements are in C.*

When an ontology $\mathscr{O} = (C, R, \leq, \perp, |, \sigma)$ is used in a particular application domain, we need to populate it with instances (or individuals). First, we will have to classify objects of a set X according to the concept symbols in C by defining a binary classification relation $\models_C$.This will determine a classification $C = (X, C, \models_C)$. Next, we will have to specify over which instances the relations represented by the symbols in R are to hold, thus classifying finite tuples of objects of X to the relation symbols in R by defining a binary classification relation $\models_R$. This will determine a classification $R = (X^+, R, \models_R)$. Both classifications will have to be defined in such a way that the partial order $\leq$, the disjoints $\perp$, the coverage $|$, and the arity function σ are respected:

Definition 2.2.2 *A populated ontology is a tuple $\widetilde{\mathscr{O}} = (C, R, \leq, \perp, |, \sigma)$ such that $C = (X, C, \models_C)$ and $R = (X^+, R, \models_R)$ are classifications and $\mathscr{O} = (C, R, \leq, \perp, |, \sigma)$ is an ontology.*

We say the ontology is correct when, for all $x, x_1, x_2, \ldots, x_n \in X, c, d \in C, r \in R$, and $\sigma(r) = \langle c_1, c_2, \ldots, x_n \rangle$

1. *if $x \models_C c$ and $c \leq d$, then $x \models_C d$;*

2. *if $x \models_C c$ and $c \perp d$, then $x \not\models_C d$;*

3. *if $c \mid d$, then $x \models_C c$ or $x \models_C d$;*

4. *if $\langle x_1, \ldots, x_n \rangle \models_R r$ then $x_i \models_R c_i$, for all $i = 1, \ldots, n$.*

2.2.4 OWL Ontology Web Language

Ontologies are a critical piece of the Semantic Web puzzle and are already used in various forms to capture knowledge in a machine understandable language. The Web Ontology Language (OWL) is a W3C standard for the ontology layer of the Semantic Web Framework [12]. It is being accepted

as the standard language for building ontologies and hence chosen for the development of the ontology in development phase.

The OWL specification provides three levels of expressiveness with increasing complexity:

1. **OWL Lite** supports classification hierarchies and only simple constraints on relations. It is easy to process but not very expressive. For example, while OWL Lite supports cardinality constraints, it only permits cardinality values of 0 or 1.

2. **OWL DL** is based on Description Logics and hence is more expressive. It includes all OWL language constructs with restrictions such as type separation (a class cannot be also an individual or property, a property cannot be also an individual or class). OWL DL was designed to support the existing Description Logic business segment and has desirable computational properties for reasoning systems.

3. **OWL Full** provides maximum expressiveness with the syntactic freedom of RDF, but provides no computational guarantees. For example, a class can be treated simultaneously as a collection of individuals and as an individual in its own right. Another significant difference from OWL DL is that an `owl:DatatypeProperty` can be marked as an `owl:InverseFunctionalProperty`. It is unlikely that any reasoning software will be able to support every feature of OWL Full.

Each of these sublanguages is an extension of its simpler predecessor, both in what can be legally expressed and in what can be validly concluded. The following set of relations hold. Their inverses do not:

- 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.

In the beginning the first languages were used to describe and create ontologies and tools for specific users community, particularly in the science and e-commerce domains. They were not defined to be compliant with the World Wide Web architecture in general, and with the Semantic Web in particular.
OWL resolves this drawback using URI to identify resources and the RDF linking to add the following characteristics to ontologies:

- Ability to be distributed among more systems.
- Scalability for the Web necessities.
- Compatibility with the Web standards as it regards the accessibility and the internationalization.
- Opening and extension possibility.

OWL is a language for the definition of structured Web-based ontologies allowing greater data integration and interoperability between communities describing their knowledge domain.
OWL is based on "RDF Model and Schema" and adds a greater vocabulary to describe classes and properties: relationship between classes (disjunction for example), cardinality (for example, "only one"), equality, and so on.

OWL Basic Elements

Most of the elements of an OWL ontology concern classes, properties, instances of classes, and relationships between these instances. This section presents the basic language components.

Simple Classes

Many uses of an ontology will depend on the ability to reason about individuals. In order to do this in a useful manner it is necessary a mechanism to describe the classes individuals belong to and the properties they inherit. Sometimes the distinction between a class and an individual is emphasized, viewing the individual as an object and the class as a set containing elements. The set of individuals that are members of a class will be called *extension* of the class.

owl:Class, rdfs:subClassOf

The most basic concepts in a domain should correspond to classes that are the roots of various taxonomic trees. Every individual in the OWL world is a member of the class `owl:Thing`, thus each user-defined class is implicitly a subclass of it. Domain specific root classes are defined by simply declaring a named class. OWL also defines the empty class, `owl:Nothing`. For example, let *Winery*, *Region*, and *ConsumableThing* be three root classes:

```
<owl:Class rdf:ID="Winery"/>
```

```
<owl:Class rdf:ID="Region"/>
<owl:Classrdf:ID="ConsumableThing"/>
```

Note that it has been only said that there exist classes to which have been given these names, indicated by the "`rdf:ID=`" syntax. Formally, almost nothing is known about these classes other than their existence, despite the use of familiar English terms as labels. It is important to remember that definitions may be incremental and distributed. In particular, it will be said more about Winery later. Within this document the Region class for example can be referred to using "`#Region`": `rdf:resource="#Region"`. Another form of reference uses the syntax `rdf:about="#Region"` to extend the definition of a resource. This use of the `rdf:about="&ont;#x"` syntax is a critical element in the creation of a distributed ontology. It permits the extension of the imported definition of `x` without modifying the original document and supports the incremental construction of a larger ontology.

The fundamental taxonomic constructor for classes is `rdfs:subClassOf`. It relates a more specific class to a more general class. If `X` is a subclass of `Y`, then every instance of `X` is also an instance of `Y`. The `rdfs:subClassOf` relation is transitive. If `X` is a subclass of `Y` and `Y` a subclass of `Z` then `X` is a subclass of `Z`.

```
<owl:Class rdf:ID="PotableLiquid">
    <rdfs:subClassOf rdf:resource="#ConsumableThing" />
        ...    ...
</owl:Class>
```

`PotableLiquid` now is a subclass of `ConsumableThing` as the labels make deduce.

A class definition has two parts: a name introduction or reference and a list of restrictions. Each of the contained expressions in the class definition further restricts the instances of the defined class. Instances of the class belong to the intersection of the restrictions. Here is a simple but incomplete definition for the class `Wine`.

```
<owl:Class rdf:ID="Wine">
  <rdfs:subClassOf rdf:resource="PotableLiquid"/>
  <rdfs:label xml:lang="en">wine</rdfs:label>
  <rdfs:label xml:lang="fr">vin</rdfs:label>
      ... ...
</owl:Class>
```

The `rdfs:label` entry provides an optional human readable name for this class. Presentation tools can make use of it. The "`lang`" attribute provides support for multiple languages. A label is like a comment and gives no contribute to the logical interpretation of an ontology.

Individuals

In addition to classes, there is the need to be able to describe their members: instances. Normally they are thought as individuals in our universe of things. An individual is minimally introduced by declaring it to be a member of a class.

```
<Region rdf:ID="CentralCoastRegion"/>
```

It has to be noted that the following is identical in meaning to the example above.

```
<owl:Thing rdf:ID="CentralCoastRegion"/>
<owl:Thing rdf:about="#CentralCoastRegion">
   <rdf:type rdf:resource="#Region"/>
</owl:Thing>
```

`rdf:type` is an RDF property that ties an individual to a class of which it is a member. There are a couple of points to be clarified here. First, it has been decided that `CentralCoastRegion` (a specific area) is member of `Region`, the class containing all geographical regions. Second, there is no requirement that the elements need to be adjacent, or even in the same file; it has been designed Web ontologies to be distributed. They can be imported and augmented, creating derived ontologies. If for example the `PotableLiquid` concept was defined in the liquid ontology elsewhere in the network, adding the following code had allowed to define the class `WhiteWine`:

```
<rdf:RDF
xmlns:liquid=http://www.w3.org/TR/2004/
                                REC-owl-guide-0421/liquid#>
  ...
  <owl:Class rdf:ID="WhiteWine">
  <rdfs:subClassOf rdf:resource="&liquid;PotableLiquid"/>
</owl:Class>
```

Now will be created a new class, `Grape`, with an individual denoting the Cabernet Sauvignon grape variety. Grapes are defined in the food ontology:

```
<owl:Class rdf:ID="Grape">
  <rdfs:subClassOf rdf:resource="ConsumableThing"/>
</owl:Class>
```

and

```
<owl:Class rdf:ID="WineGrape">
  <rdfs:subClassOf rdf:resource="Grape"/>
</owl:Class>
<WineGrape rdf:ID="CabernetSauvignonGrape"/>
```

There are important issues regarding the distinction between a class and an individual in OWL. A class is simply a name and a collection of properties which describes a set of individuals. Individuals are the members of those sets. These classes should correspond to naturally occurring sets of things in a domain of discourse, and individuals should correspond to actual entities. OWL Full permits such an expressivity that allows to treat an instance as a class and vice versa. This allows for example to state relations between an instance and a concept; for instance it can be said that the SWISS-Prot (an instance of the 'Database' concept) concerns 'Protein' domain (a concept).

2.2.5 Description Logic

Description Logics (DLs) are formal languages for knowledge representation and represent the core of OWL-DL language.

DLs classify knowledge in two parts: the T-Box and the A-Box. The T-Box contains terminological information which is general (good for representing background knowledge). The A-Box contains assertions which are specific (good for representing sentences). Another way to see the division between these two kinds of information is to regard the T-Box as rules which govern our world (e.g., laws from physics, chemistry, biology, etc), and the A-Box as depicting the world's individuals (e.g., a table, a chair, a man, etc).

Description Logics employ the notions of concept, role and individual. Concepts are classes of elements and are interpreted as a subset of a given universe. Roles are links between elements and are interpreted as binary relations of a given universe. Individuals are the elements of a given universe. Using the letters C and D for atomic concepts, the letter R for atomic roles, and the symbols $\top$ and $\bot$ for representing the *universal* and *bottom* concepts respectively, we give the following definition of basic description language $\mathcal{ALC}$

Definition 2.2.3 *($\mathcal{ALC}$ syntax). Let N_C be a set of concept names and N_R be a set of role names. The set of $\mathcal{ALC}$-concept descriptions is the smallest set such that:*

1. *$\top$, $\bot$, and every concept name $A \in N_C$ is an $\mathcal{ALC}$-concept description,*
2. *if C and D are $\mathcal{ALC}$-concept descriptions and $r \in N_R$, then $C \sqcap D, C \sqcup D, \neg C, \forall r.C$, and $\exists r.C$ are $\mathcal{ALC}$-concept descriptions.*

The semantics of $\mathcal{ALC}$ is given in terms of *interpretations*.

Definition 2.2.4 *($\mathcal{ALC}$ semantics). An interpretation $\mathcal{I} = (\Delta_{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a nonempty set $\Delta^{\mathcal{I}}$, called the domain of $\mathcal{I}$, and a function $\cdot^{\mathcal{I}}$ that maps every $\mathcal{ALC}$-concept to a subset of $\Delta^{\mathcal{I}}$, and every role name to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ such that, for all $\mathcal{ALC}$-concepts C, D and all role names r,*

$$\top^{\mathcal{I}} = \Delta^{\mathcal{I}}, \quad \bot^{\mathcal{I}} = \emptyset,$$

$$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}, \quad (C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}, \quad \neg C^{\mathcal{I}} = \Delta^{\mathcal{I}} \backslash C^{\mathcal{I}},$$

$$(\exists r.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} |\ \textit{there is same } y \in \Delta^{\mathcal{I}} \textit{ with } \langle x, y\rangle \in r^{\mathcal{I}} \textit{ and } y \in C^{\mathcal{I}}\},$$

$$(\forall r.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} |\ \textit{for all } y \in \Delta^{\mathcal{I}}, \textit{ if } \langle x, y\rangle \in r^{\mathcal{I}}, \textit{ then } y \in C^{\mathcal{I}}\}.$$

We say that $C^{\mathcal{I}}$ ($r^{\mathcal{I}}$) is the extension of the concept C (role name r) in the interpretation $\mathcal{I}$. If $x \in C^{\mathcal{I}}$, then we say that x is an instance of C in $\mathcal{I}$.

A knowledge base Σ is a pair $\langle T, A\rangle$, where T is the T-Box and A is the A-Box:

- A T-Box T is a finite set of expressions called General Concept Inclusions (CGI) with shape $C_1 \sqsubseteq C_2$, where C_1,C_2 are concepts. The intended meaning of $C_1 \sqsubseteq C_2$ is that the set of individuals in C_1 is included in the set of individuals in C_2. $C_1 \doteq C_2$ is a notation for $C_1 \sqsubseteq C_2$ and $C_2 \sqsubseteq C_1$. Formulas of T are also called *terminological axioms*.
- An A-Box A is a finite set of expressions with shape a:C or (a, b):R, where C is a concept, R a role and a,b two individuals. The first expression means that the individual a belongs to the set of individuals satisfying C. The second expression means that the relation R holds between a and b. Formulas of A are called *assertions*.

2.2.6 The family of AL-languages

$\mathcal{ALC}$ has been extended with several features that are important in an ontology language, including (qualified) number restrictions, nominals, inverse roles, transitive roles, sub-roles and concrete domains. With *number restrictions* (indicated by the letter $\mathcal{N}$), it is possible to describe the number of relationships of a particular type that individuals can participate in. The *nominal* constructor allows the possibility to use individual names also within concept descriptions: if a is an individual name, then $\{a\}$ is a concept, called a nominal, which is interpreted by a singleton set. To supporting the modeling of concrete properties of abstract objects such as the age, the weight, or the name of a person, and the comparison of these concrete properties, DL is integrate with concrete sets such as the real numbers, integers, or strings, as well as concrete predicates defined on these sets, such as numerical, string and constants comparisons. This features is called *concrete domains*(indicated by the letter D).

The name given to a particular DL usually reflects its expressive power. There is a relation between letters and expressivity that the constructors provided. The letter $\mathcal{S}$ [1] is used for indicate "basic" DL consisting of $\mathcal{ALC}$ extended with transitive roles:

- The letter $\mathcal{H}$ represents subroles (role $\mathcal{H}$ierarchies),
- $\mathcal{O}$ represents nominals (n$\mathcal{O}$minals),
- $\mathcal{I}$ represents inverse roles ($\mathcal{I}$nverse),
- $\mathcal{N}$ represents number restrictions ($\mathcal{N}$umber), and
- $\mathcal{Q}$ represents qualified number restrictions ($\mathcal{Q}$ualified).

The integration of a concrete domain/datatype is indicated by appending its name in parenthesis, but sometimes a "generic" D is used to express that some concrete domain/datatype has been integrated. The DL corresponding to the OWL DL ontology language includes all of these constructors and is therefore called $\mathcal{SHOIN}(D)$.

[1] In order to avoid very long names for expressive DLs the abbreviation letter $\mathcal{S}$ is often used as an abbreviation for the "basic" DL consisting of $\mathcal{ALC}$ extended with transitive roles (which in the $\mathcal{AL}$ naming scheme would be called $\mathcal{ALC}_{R^+}$).

2.2.7 SKOS: Simple Knowledge Organization System

In the previous section we have introduced OWL-DL and we have briefly shown its potential based on description logic. However, who have no practice with logic can find hard to work with OWL-DL. For this and other reasons Semantic Web, but also Knowledge Representation, needs tools that are expressive and easy to be managed. SKOS play this role.

SKOS defines an ontology which allows to express the basic structure and contents of concept diagrams, including thesauruses, thematic lists, heading lists, taxonomies, terminologies, glossaries, and other kinds of controlled dictionaries. The specification is divided into three parts:

- SKOS-Core [1] [2]: it defines basic concepts and relations which enable to develop concepts and relations between them;
- SKOS-Mapping[3]: it introduces relations which allow to describe similarities between concepts created in different ontologies;
- SKOS-Extensions[4]: it introduces extentions of the intensity of hierarchical relations from SKOS-Core.

The fundamental element of the SKOS vocabulary is the concept. SKOS introduces the class *skos:Concept*, which allows implementors to assert that a given resource (*individuals*) is a concept. In SKOS semantic relations play a crucial role for defining concepts.

SKOS Semantic Relationships

To assert that one concept is broader in meaning (i.e. more general) than another, the *skos:broader* property is used, while *skos:narrower* property is used to assert the inverse, namely when one concept is narrower in meaning (i.e. more specific) than another. Declaring both narrower and broader properties transitive as well as not transitive imply different reasoning behaviours. Another important semantic relation in SKOS-Core is *skos:related*. *skos:related* enables the representation of associative (non-hierarchical) links between resources (concepts, individuals).

SKOS Concepts Schema

In SKOS a concept can be defined either in relation to other concepts (as part of an internally coherent concept scheme), or as a stand-alone resource. The SKOS Guide defines a concept scheme as “a set of concepts, optionally including statements about semantic relationships between those concepts”,

and provides the class *skos:ConceptScheme* for authors to assert "that a resource is a concept". Using the *skos:hasTopConcept*, concepts can be linked to their top level concepts. It is also possible to assert that a concept belongs to a particular concept scheme, by using the *skos:inScheme* property.

Combining SKOS with OWL

In SKOS, *skos:Concept* is defined as an OWL class; it follows that in OWL instances of *skos:Concept* are individuals. Treating *skos:Concept* individuals as OWL classes can be seen as a metaclass modelling approach, where SKOS users specify class-level characteristics of SKOS concepts. That is possible in OWL-Full, where the sets of classes and individuals do not require to be disjointed. In OWL-DL, where disjointedness condition between classes and individuals must be hold, users cannot use this metamodeling mechanism. They can nevertheless use dedicated OWL properties to bridge them like in the following example:

```
ex:Database rdf:type owl:Class.
ex:Protein rdf:type skos:Concept.
ex:SWISS-Prot rdf:type owl:DataBase
ex:SWISS-Prot ex:concerns ex:Protein.
```

We use this kind of approach as illustrated in the next chapter for defining the $\mathcal{R}$*esourceome* Multi-Level Ontology Model.

Chapter 3

Distributed Computation

A distributed system is a collective of independent computers, interconnected via a network, able to collaborate on a task. Distributed computing is a type of segmented or parallel computing, but the latter term is most commonly used to refer to processing in which different parts of a program run simultaneously on two or more processors that are part of the same computer.

While both types of processing require that a program be segmented and divided into sections that can run simultaneously, distributed computing also requires that the division of the program takes into account the different environments on which the different sections of the program will be running. The paradigms of distributed computation can be presented in order to their level of abstraction (see Fig. 3.1).

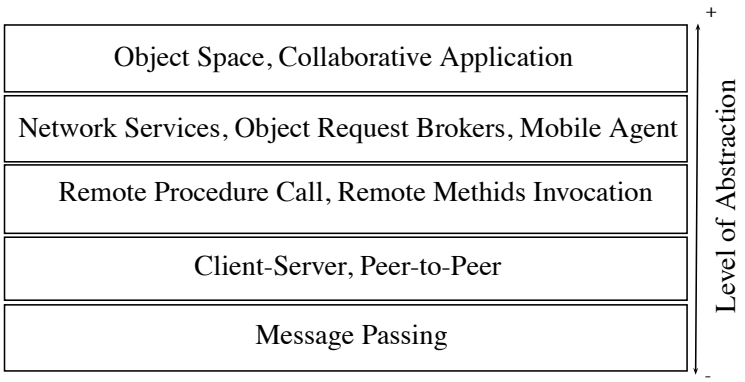

Figure 3.1: Distributed computing paradigms and their level of abstraction.

In the following, we will describe two distributed computation paradigms: the *Service-Oriented* architecture (*SOA*) and the *agent*-based paradigm. In

particular, we will focus on the latter one, introducing $\mathcal{H}ermes$, an agent-oriented FIPA Compliant middleware, used in $\mathcal{R}esourceome$ to properly support mobile and distributed computations.

3.1 The SOA Paradigm

Service-Oriented Architecture (*SOA*) is an architectural style for building software applications that use services available in a network such as the Web. It promotes loose coupling between software components so that they can be reused. Applications in *SOA* are built based on services.

A service is an implementation of a well-defined business functionality, and such services can then be consumed by clients in different applications or business processes. SOA allows for the reuse of existing assets where new services can be created from an existing IT infrastructure of systems. In other words, it enables businesses to leverage existing investments by allowing them to reuse existing applications, and promises interoperability between heterogeneous applications and technologies.

SOA provides a level of flexibility that was not possible before in the sense that:

- Services are software components with well-defined interfaces that are implementation-independent. An important aspect of *SOA* is the separation of the service interface (the what) from its implementation (the how). Such services are consumed by clients that are not concerned with how these services will execute their requests.
- Services are self-contained (perform predetermined tasks) and loosely coupled (for independence).
- Services can be dynamically discovered.
- Composite services can be built from aggregates of other services *SOA* uses the find-bind-execute paradigm as shown in Fig. 3.2. In this paradigm, service providers register their service in a public registry. This registry is used by consumers to find services that match certain criteria. If the registry has such a service, it provides the consumer with a contract and an endpoint address for that service.

More in detail, *SOA* relies on services exposing their functionality via interfaces that other applications and services can read to understand how to utilize those services, as well as it enables the development of applications that are built by combining loosely coupled and interoperable services.

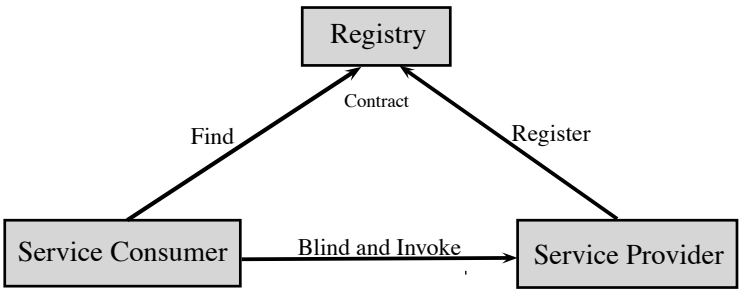

Figure 3.2: *SOA*'s Find-Bind-Execute Paradigm .

These services inter-operate based on a formal definition (or contract, e. g., WSDL[96]) that is independent of the underlying platform and programming language. The interface definition hides the implementation of the language-specific service.

SOA-based systems can therefore function independently of development technologies and platforms (such as Java, .NET, etc). *SOA* can support integration and consolidation activities within complex enterprise systems, but *SOA* does not specify or provide a methodology or framework for documenting capabilities or services.

High-level languages such as BPEL[5] and specifications such as WS-CDL[121] extend the service concept by providing a method of defining and supporting orchestration of fine-grained services into more coarse-grained business services, which architects can in turn incorporate into workflows and business processes implemented in composite applications or portals.

3.2 The Mobile Agent Paradigm

Agents can be defined to be autonomous, problem-solving computational entities capable of effective operation in dynamic and open environments.

They are often deployed in environments in which they move, interact, and sometimes cooperate with other agents (including both people and software) that have possibly conflicting aims. These environments are known as *multi-agent* systems (MASs).

Since agents are autonomous entities capable of exercising choice over their actions and interactions, act to achieve individual objectives, they cannot, therefore, be directly invoked but can be assigned tasks by their owners. These notions find application in relation to three distinct views, considered below.

- *Agents as design metaphor:* Agents provide designers and developers with a way of structuring an application around autonomous, communicative components, and lead to the construction of software tools and infrastructure to support the design metaphor. In this sense, they offer a new and often more appropriate route to the development of complex systems, especially in open and dynamic environments. In order to support this view of systems development, particular tools and techniques need to be introduced. For example, methodologies to guide analysis and design are required, agent architectures are needed for the design of individual components, tools and abstractions are required to enable developers to deal with the complexity of implemented systems, and supporting infrastructure (including more general, current technologies, such as Web Services) must be integrated.

- *Agents as a source of technologies:* Agent technologies span a range of specific techniques and algorithms for dealing with interactions in dynamic, open environments. These address issues such as balancing reaction and deliberation in individual agent architectures, learning from and about other agents in the environment, eliciting and acting upon user preferences, finding ways to negotiate and cooperate with other agents, and developing appropriate means of forming and managing coalitions. Moreover, the adoption of agent-based approaches is increasingly influential in other domains. For example, MASs have already provided faster and more effective methods of resource allocation in complex environments, such as the management of utility networks, than previous approaches.

- *Agents as simulation:* MASs offer strong models for representing real-world environments with an appropriate degree of complexity and dynamism. For example, simulation of economies, societies and biological environments are typical application areas. The use of agent systems to simulate real-world domains may provide answers to complex physical or social problems that would be otherwise unobtainable, as in the modelling of the impact of climate change on biological populations, or modelling the impact of public policy options on social or economic behaviour. Agent-based simulation spans: social structures and institutions to develop plausible explanations of observed phenomena, to help in the design of organizational structures, and to inform policy or managerial decisions; physical systems, including intelligent buildings, traffic systems and biological populations; and software systems of all types, currently including e-Commerce and information agency. In ad-

dition, multi-agent models can be used to simulate the behaviour of complex computer systems, including multi-agent computer systems. Such simulation models can assist designers and developers of complex computational systems and provide guidance to software engineers responsible for the operational control of these systems. Multi-agent simulation models thus effectively provide a new set of tools for the management of complex adaptive systems, such as large-scale online resource allocation environments.

3.3 *Hermes*, an agent-oriented FIPA Compliant system

In the following, we introduce *Hermes*, a component-based agent-oriented FIPA Compliant system.

3.3.1 *Hermes* Middleware

Hermes is structured as a component-based, agent-oriented system with a 3-layer software architecture shown in Figure 7.2. In the following, we will give a detailed description of each layer.

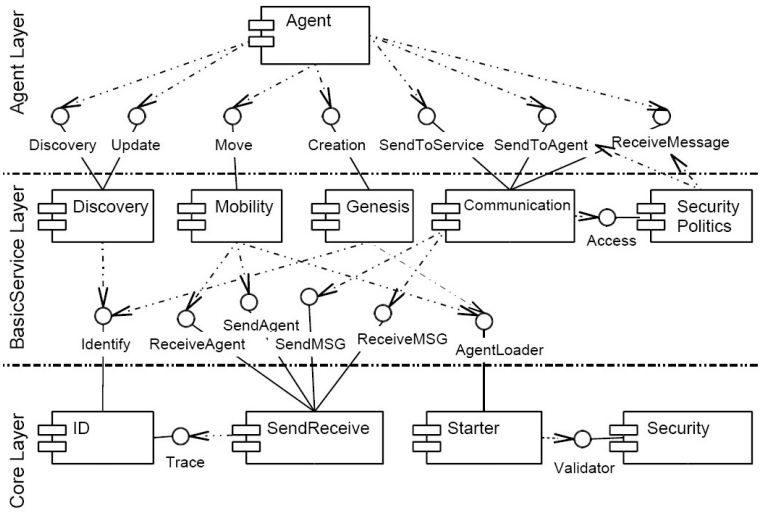

Figure 3.3: 3-Layered Architecture of *Hermes* Mobile Computing Platform.

Core Layer

It is the lowest layer of the architecture and contains base functions of the system, such as the implementation of the inter-platform communication protocols and agent management functions. This layer is composed of four components: *ID*, *SendReceive*, *Starter* and *Security*.

The *ID* component, implements general identity management functions by managing a repository containing information about locally generated agents. This repository is accessed whenever we want to know the current position of an agent. The *ID* component is also responsible for the creation of the identifiers to be associated to new agents. These identifiers contain information about birthplace, date and time of the agent's creation. Agent localization is simplified by information contained directly in the *ID*, such as birth place. In fact, the birth place of an agent hosts information about agent's current location.

A second important feature of the Core is the *SendReceive* component. This component implements low level inter-platform communication by sending and receiving messages and agents. By using traceability services offered by the *ID* component, *SendReceive* can easily update or retrieve the exact position of a specific user agent.

The *Starter* component processes any request for agent creation. This particular component, in fact, take an inactive agent (just created or migrated), and checks it for the absence of malicious or manipulated code. These agents, before activation, are dynamically linked to all basic services of the platform.

During execution the agent is isolated from the *Core* Layer by the *BasicService* layer. The *Security* component, as mentioned above, checks for the presence of malicious code or manipulations within agent code.

BasicService Layer

This layer has five main components: *Discovery*, *Mobility*, *Genesis*, *Communication* and *Security Politics*.

The *Discovery* component searches and detects service agents. When a user agents wants to communicate with a service, it will ask the *Discovery* for the right identifier to use as the messages's receiver. The service detection strategy can be implemented in several ways; for example by a fixed taxonomy or by an UDDI [31], commonly used in Web Service application domain.

The mobility component enables the movement of code across platforms [37], it implements the interface used by the *Agent* component and it accesses to

components of the *Core* layer to send, receive and load agents. It is important to note that real communication between different locations can be achieved only through Core's *SendReceive* component, and then migration is independent of the type of used transport.

Mobility consists on copy the agent i.e. its code and its current state and send it to the destination platform where it will re-started in a specific point (weak mobility). The local agent is destroyed.

The *Communication* component makes possible to send and receive agent-directed messages both in an intra- and inter-platform context. Intra-platform messages are messages sent between agents and services residing in the same platform. Inter-platform messages are messages sent to agents residing in different platforms (our system does not allow for remote communication between user agents and service agents). The agent requesting the dispatch of a message does not need to know, effectively, where the target agent is; in fact, the *ID* is sufficient to post correctly a message.

The *Communication* component uses one of the *Security Policy*'s interfaces to ascertain whether the specific *UserAgent* or *ServiceAgent* has the right privileges for communication. If an Agent is not authorized to use a service, the message is destroyed. Before accessing resources and services, an agent must authenticate itself. The identification is performed by sending a login message to a specific ServiceAgent, as consequence the *SecurityPolitics* component jointly with the *Communication* component intercept the message and unlock the communication.

The *SecurityPolitics* component centralizes control of permissions, protects services and resources from the user agents, and provides the administrator with an easy way to manage all permissions.

The last component of this layer is the *Genesis* component that enables agent creation. A special case of agent creation is cloning, that is performed when it is necessary to create a copy of an existing agent. The two copies differ only for the agent identifier.

Agent Layer

The *Agent* Layer is the upper layer of the mobile platform and contains all service and user agents. This component has not any interface, but it has only several dependencies upon the *BasicService* Layer. The *Agent* component provides a general abstract *Agent* class. *ServiceAgent* and *UserAgent* classes extend this abstract class. *ServiceAgent* consists of agents enabling access to local resources such data and tools. Agents in *UserAgent* execute complex tasks and implement part of the logic of the application. Java programmers can also develop *UserAgent* by using the API provided by $\mathcal{H}ermes$ Mobile

Computing Library. Listing 3.1 shows a simple demo. A MkDuckAgent called *"Della Duck"* creates three sons *Qui*, *Quo* and *Qua* -lines 24 to 40- by cloning itself. After clonation, each new agent starts its behaviour calling *"afterCloning"* as initial method.

```
1  package samples;
2  import hermesV2.*;
3  import hermesV2.agent.*;
4
5  public class McDuckAgent extends UserAgent {
6
7    public McDuckAgent(String agentName) {
8      super("Della Duck");
9    }
10
11   public void init() {
12     reception(); //I enable the reception
13                  //of messages for the father
14
15     System.out.println("Hello World !!");
16     System.out.println("I'm Della Duck !!!");
17
18     Identificator temp=null,son1=null,
19                   son2=null,son3=null;
20
21     /*afterCloning is the first method
22       called after the clonation*/
23
24     try {
25       son1 = clone("afterCloning", "Qui");
26       System.out.println(new Date(
27                System.currentTimeMillis()) +
28                ": Qui was born !!");
29       son2 = clone("afterCloning", "Quo");
30       System.out.println(new Date(
31                System.currentTimeMillis()) +
32                ": Quo was born !!");
33       son3 = clone("afterCloning", "Qua");
34       System.out.println(new Date(
35                System.currentTimeMillis()) +
36                ": Qua was born !!");
37     } catch (CloneException ce) {
38       System.out.println(ce);
39     }
40     Message m0=null,m1=null,m2=null,m3=null;
41     while (!(m1!=null && m2!=null &&
42            m3!=null)){
43       m0 = getMessageSynch();
44          temp = m0.getSenderAgentId();
45          if (son1.equals(temp)) m1 = m0;
46          if (son2.equals(temp)) m2 = m0;
47          if (son3.equals(temp)) m3 = m0;
48          System.out.println((String)m0.getObject());
49     }
50     /*The mother replies to sons*/
51     Identificator myId = getIdentificator();
52     m1 = new Message(myId, son1,
53                 "Mom: Ok Qui !! \n      "+
54                 "I've receive your message.");
55     m2 = new Message(myId, son2,
```

```
56                  "Mom: Ok Quo !! \n      "+
57                  "I've receive your message.");
58      m3 = new Message(myId, son3,
59                  "Mom: Ok Qua !! \n      "+
60                  "I've receive your message.");
61      try {
62        sendMessageToUserAgent(m1);
63        sendMessageToUserAgent(m2);
64        sendMessageToUserAgent(m3);
65      } catch (CommunicationException ce) {
66        System.out.println(ce.getMessage());
67      }
68    }
69
70    public void afterCloning() {
71      Identificator myId = getIdentificator();
72      PlaceAddress  myBPA = myId.getBornPlaceAddress();
73      int           myBPAPort = myBPA.getPort();
74      try {
75        int port = (myBPAPort == 9100) ? 9000 : 9100;
76
77        PlaceAddress  myMPA =
78                new PlaceAddress(myBPA.getIp(), port);
79        this.move(myMPA, "afterMoving");
80      } catch (MigrationException me) {
81        System.out.println("MigrationException" + me);
82      }
83    }
84
85    public void afterMoving() {
86      reception(); //I enable the reception
87                   //of messages for the son
88
89      Identificator myId = getIdentificator();
90      Identificator mother = getFatherIdentificator();
91      Message m = null;
92
93      try {
94          m = new Message(myId, mother,
95                 getAgentName() +
96                 ": I have moved to another Place");
97          sendMessageToUserAgent(m);
98      } catch (CommunicationException ce) {
99        System.out.println(ce.getMessage());
100     }
101     m = getMessageSynch(mother);
102     System.out.println((String)m.getObject());
103   }
104 }
```

Listing 3.1: McDuckAgent.java

By using *"move"* method -line 79- Qui, Quo and Qua migrate to a *Place* different from where they were born. When they arrive in the new *Place* each one calls the *"afterMoving"* -line 85- method. Then they notify to their mom their moving by using *"sendMessageToUserAgent"* -line 97- and *"getMessageSynch"* -line 101- methods. Figure 3.4 shows the final results.

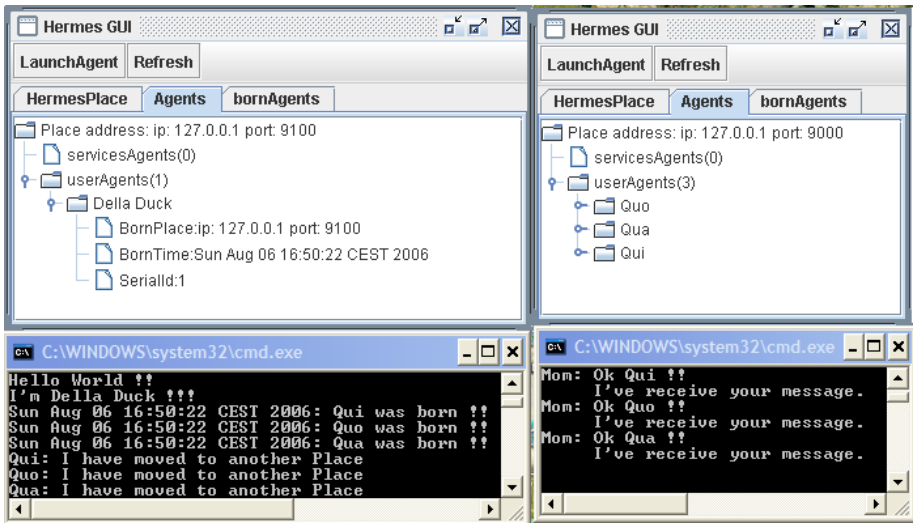

Figure 3.4: Final result produced by McDuckAgent.java

3.3.2 Distributed Process Specification

Processes can be described by their starting and stopping points and by the kind of changes that take place in between. In *Physical Process*, that can be seen as continuous process, change take place continuously, and can be associated to an analog computer. *Discrete Process* changes occur in discrete steps called events and states. Typically events are interleaved with states that represent periods of inactivity. To program the original von Neumann machine, Neumman and Goldstine designed *Flow-Charts*. In Flow-Charts boxes represent computational events while diamonds represent decisions, where after a decision, the flow can take an alternative path. Flow-Charts can compute any function that is computable by a Turing machine. Although functional language can compute any computable function, they cannot be used to enforce a particular order of execution. Procedural languages are a good approach for such specification. Flow-Charts, State-transition Diagrams (Finite-State Machine) and Petri Nets are good notation tools for representing procedures.

Petri Nets merge the ideas of both Flow-Charts and Finite-State Machine formalism, where events of Flow-Charts are mapped into Petri Net transactions, while the states of Finite-State diagrams are mapped into Petri Net places (see Figure 3.5).

Although Flow-Charts and Finite-State Machines can represent branches and loops, Petri Nets are a better way to specify and represent parallel and concurrent processes as well as processes synchronization. In particular, we

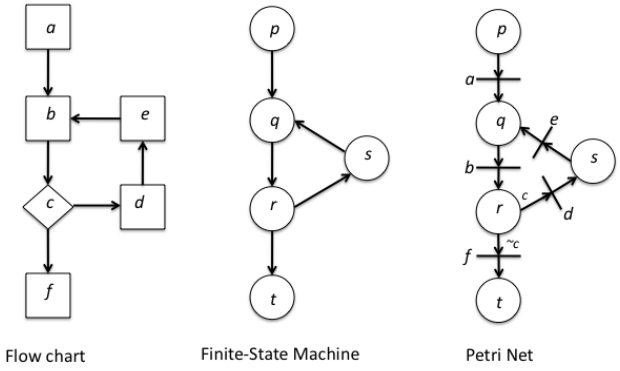

Figure 3.5: Graphic notations for procedure specification

have used a Petri Net-based notation to describe semantic workflow specifications, activities and related roles. Automating such processes requires a model that describes the coordination of the activities to be executed, the roles involved in the organization and the needed resources.

Part III

Resourceome Ontology Framework

Chapter 4

Resourceome

Formal Knowledge Representation (KR) consists of building formal models of a particular domain or problem in order to allow automatic reasoning and interpretation. Such formal models are called *ontologies*. Ontology languages can be used to provide formal semantics to any sort of information.

In this chapter we propose a multi-level ontology model, called *Resourceome*, which can be considered as a conceptual schema to represent knowledge in the most formal and reusable way.

4.1 Introduction

In the field of Computer Science, ontology has became popular as a paradigm for Knowledge Representation in Artificial Intelligence, by providing a methodology for easier development of interoperable and reusable knowledge base. The Handbook of Knowledge Representation [18] defines the ontology as follows: "An ontology is an explicit specification of a shared conceptualization that holds in a particular context". The gradual changes of the definition of what an ontology is are interesting in their own right. Gruber's original definition takes ontology to be an explicit specification of a conceptualization [6]. The introduction of the adjective shared occurred some years later, especially in the work of Borst et al. [19]. The reference to validity in a particular context is again a later addition, and attempts to encapsulate lessons learned in building and using ontologies in practical applications, especially in knowledge management[21]. The original idea of ontologies is that of a consciously engineered artifact to express intended meaning through specifying the formal semantics of concepts. In the latter conception, ontologies figure as social network-created, inseparable from the context of the commu-

nity in which they are created. These two different approaches to ontologies do not necessarily exclude each other. In the engineering approach, an important critical issue is the distance between the engineered ontology and its intended user or application community. In the social community approach, on the other hand, a critical issue is that in ontology extracted as a matter-of-fact empirical (i.e. non-engineered) social phenomenon ultimately "anything goes": shared meaning simply becomes what any community happens to believe, without any further rational foundation or justification. As quite an elementary point of context, ontologies are formal conceptualizations helping an actor to achieve a goal or task. Often that task involves knowledge-intensive reasoning. The conceptual distinctions that we make in our attempts to understand the world are not just static and descriptive domain characterizations, but they are made to serve practical purposes of action by someone or something in that world. We cannot detach knowledge from action. This is clearly a pragmatic, use-oriented view of ontologies. Part of the context surrounding the normal static ontologies is computationally defined by explicating the reasoning task and methods in which ontologies find their use.

4.2 The main features of the $\mathcal{R}esourceome$ knowledge model

The $\mathcal{R}esourceome$ knowledge organization is fundamentally a multilevel ontology-based model for semantic annotation of *resources* and *activities*, in according to Guarino's approach [11]. It is composed of three levels - Top knowledge, Base knowledge and Application knowledge level (Fig. 4.1).

The $\mathcal{R}esourceome$ knowledge model implements the following features:

1. *Separation between domain concepts (abstract and concrete concepts) and resource concepts (only physical concepts).*

   In $\mathcal{R}esourceome$ it is possible to derive specific resources from resource concepts w.r.t. the domain where they are conceptualized. This is obtained by splitting the Domain Ontology (in the sense of Guarino's approach) into two "orthogonal" ontologies - Resource Ontology and Domain Ontology (in our sense) - and by connecting them by a "*concerns*" relation. This mechanism permits to parameterize the knowledge representation w.r.t. the domain. As a consequence, $\mathcal{R}esourceome$ is able to formalize a resource conceptualization, contextualized in and concerned to *any* fixed conceptualized domain. Through the "*concerns*"

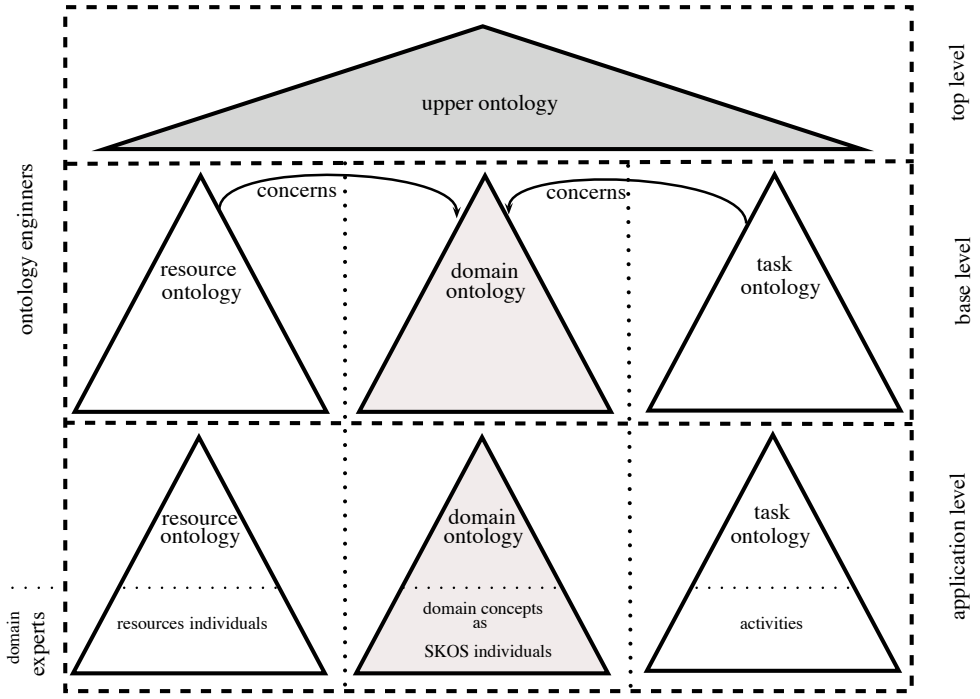

Figure 4.1: The $\mathcal{R}$*esourceome* model.

relation, a more specific resource can be simply connected to the domain topics which it refers to, rather than introduced in the Resource Ontology as a more specialized concept. Figure 4.2 and Figure 4.3 shows how a resource can be specialized w.r.t. a context either in the classical approach (see Figure 4.2) or in $\mathcal{R}$*esourceome* (see Figure 4.3).

2. *Independence from different workflow views and concrete formalisms.*

   The *behavioral aspect* is the most explicit view of a workflow: it basically describes the order in which the different activities are executed. However, a workflow is more than the mere connection of activities: other basic aspects to consider are the *organizational* one - describing the organization structure, the involved *objects* and *roles* and in which way they are involved in the workflow - and *the informational* one - describing the concrete *informations* (or *documents*) associated to objects and roles that are involved in the workflow, how and where they are represented, and how they are propagated among different activities.

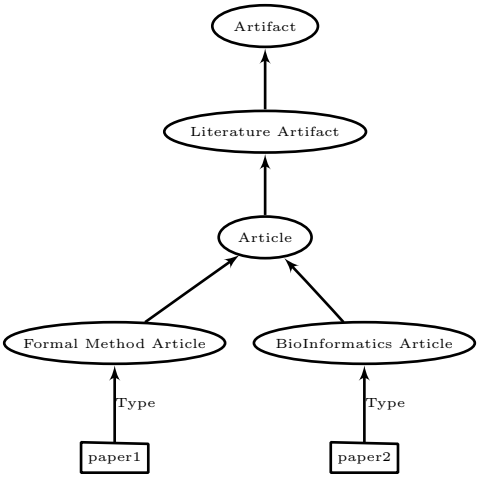

Figure 4.2: Classic Conceptualisation.

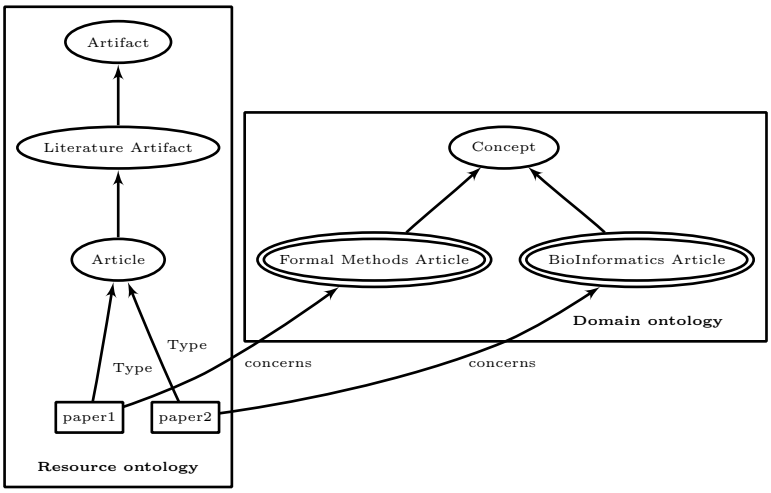

Figure 4.3: $\mathcal{R}esourceome$ Conceptualization.

Usually, each view is modeled by a specific and suitable formalism: for instance, Petri Nets, organigrams and ER diagrams respectively for behavioral, organizational and informational aspects. For this reason, the Task Ontology in $\mathcal{R}esourceome$ has been defined in order to be independent from concrete formalisms and to catch the essence of each view.

The Task Ontology is connected to Domain and Resource Ontologies by abstract relations, in such a way to describe the execution context

of an activity in a semantically rich fashion. These relations link (the most general) activity concept in the Task Ontology with (the most general) domain and resource concepts. Specializations of these relations permit to attach the involved roles, documents and objects to a specific activity, as well as to a domain, i.e. the context in which the activity works. These relations provide the needed support for realizing the semantic-driven workflow compilation process in $\mathcal{R}esourceome$.

3. *Representation of social roles.*

   The notion of *role* is crucial in a workflow scenario. A role is a logical abstraction of one or more physical actors which can perform an activity. The conceptual modeling and object modeling literature considers the roles as anti-rigid and relationally dependent unary predicates. For example, take the role *Student* that is subsumed by the kind *Person*: *Student* is anti-rigid because persons are only contingently students, for example a person can be a student only during a short period of his life. Additionally, *Student* is relationally dependent because, for a person to be *Student* it requires the existence of another entity, namely a certain *University* in which this person *is Enrolled*. $\mathcal{R}esourceome$ implements the approach proposed in [14], which it has been proved to solve the so-called "Counting-Problem". Figure 4.4 shows how a social role can be described as classical approach[1] or in $\mathcal{R}esourceome$ model Figure 4.5, where:
   $Student \equiv Person \sqcap \exists\, Enrroled.Univerity \sqcap \exists\, playRole.\{\mathsf{StudentRole}\}$
   and $Student(\mathsf{Rossi})$ is inferred.

In the following, we will describe in detail the multilevel architecture of the $\mathcal{R}esourceome$ knowledge model.

4.2.1 Top knowledge level

The Top knowledge level is formed by an Upper Ontology describing very general and domain-independent concepts shared across a large number of ontologies. Several standard upper ontologies are available, for instance DOLCE (Descriptive Ontology for Linguistic and Cognitive Engineering) [13] and SUMO (Suggested Upper Merged Ontology) [23]. The choice of the Upper Ontology concepts depends on what and how the knowledge is going to be described.

[1] This representation suffers of the so-called "Counting Problem", as stated in [14].

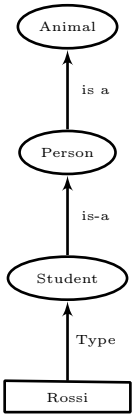

Figure 4.4: Social roles classic representation.

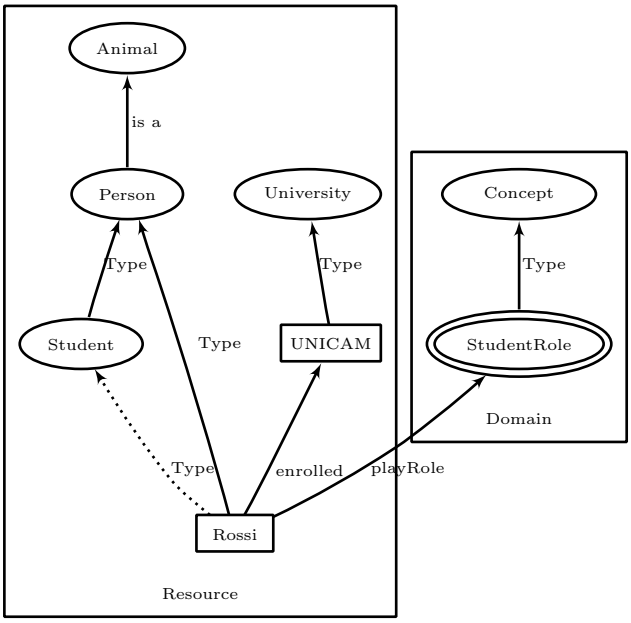

Figure 4.5: Social role representation in $\mathcal{R}esourceome$.

4.2.2 Base knowledge level

The Base knowledge level describes a specific vocabulary by specializing the terms introduced in the Upper Ontology w.r.t. a particular domain of interest.

Domain and Resource Ontologies. The Domain Ontology (in the sense of Guarino's approach) is split into two "orthogonal"[2] ontologies: Resource Ontology and Domain Ontology (in our sense).

The Resource Ontology represents the kind of resources existing in the universe of a domain. It is a representation of a physical world, since it models the types of resources existing in the described domain. It contains the declaration of terminology and relations in the form of OWL-DL entities.

The Domain Ontology represents the semantic relationships between the concepts of the domain. It is implemented with a hybrid OWL-DL/SKOS semi-formal language, in order to provide more flexible and less formal description of concepts and metadata[3]. A domain concept may be both a sub-class of a domain OWL-DL class and a sub-class of SKOS-concept class. On the one hand, SKOS permits to represent concepts as individuals of SKOS-concept class and to join them together with more flexible relations instead of the strict "*is-a*" one. On the other hand, SKOS does not provide strict computational semantics and thus cannot be used for performing automated tasks associated to the knowledge represented in the scheme. In the whole, the hybrid OWL-DL/SKOS semi-formal specification of the domain offers a great flexibility and the possibility of creating relationships between individuals and domain/operational concepts: it combines the OWL-Full flexibility - since it can describe relations between resource individuals and domain concepts - and the OWL-DL computational power - since the adopted interbreeding technique does not add non-computable elements.

Task Ontology. The Task Ontology contains the declaration of terminology and relations in the form of OWL-DL entities. The Base level is substantially a meta-model for catching different workflow views (behavioral, organizational and informational aspects), in such a way to be independent from concrete formalisms usually used for modeling them. The pivot of this level is represented by the generic concept of *activity*. Every activity executes in a defined context (domain concept) and includes the following items:

- *Role* (domain concept): A logical abstraction of one or more physical actors, usually in terms of common responsibility or position. An actor may be a member of one or more roles. Roles specify the actor types (e.g. researcher, teacher, etc.) that participate in the workflow and perform activities. For maximum flexibility and re-use, activities are

[2]As already said in Section 4.2, the "orthogonality" is realized by a "*concerns*" relation, which allows us to connect the two ontologies and to parameterize the knowledge representation w.r.t. a domain.

[3]SKOS is represented in OWL-DL. In detail, in OWL 2 it is possible to have an ontology where the names of individuals can be the same as the names of classes.

assigned to roles rather than to named users. Using roles instead of assigning a real user's name makes changes easy to manage and it enables a user to easily delegate and reassign activities. Every role has a set of *competencies*, which describe what the people associated with that role are allowed to do.

- A set of *objects* (resource concepts), which the activity eventually inputs, outputs and uses and/or consumes.

- *Document*: The set of informations (resource concepts) that are interchanged as part of an activity.

- A *code* implementing the activity.

- It possibly has *preconditions* and *effects* (rules)[4].

- It is decomposed into more detailed *sub-activities* (task concepts).

As already said, the Task Ontology is connected to Domain and Resource Ontologies by abstract relations - in detail, "*hasRole*", "*hasDocument*", "*hasComplexInput*", "*hasComplexOutput*" and "*concerns*". Their specializations permit to attach the involved roles ("*hasRole*"), documents ("*hasDocument*") and objects ("*hasComplexInput*", "*hasComplexOutput*") to a specific activity, as well as to a domain ("*concerns*") or context in which the activity works.

4.2.3 Application knowledge level

The Application knowledge level introduces very specific concepts depending on the particular domain, resources and activities.

Domain and Resource Ontologies. The Resource Ontology depth hardly grows: as already explained in Section 4.2, in the most cases inserting more specific resources implies connecting their more generic forms, already in the Resource Ontology, to related domain concepts which they refer to. Eventually, new domain concepts and new general resource concepts are inserted respectively in Domain and Resource Ontologies.

Task Ontology. This level models the semantic (specialization/generalization) relationships among concrete activities, the conditions to be met for an activity specialization to take place, and the context each activity works on. Basically activities can be retrieved by their name, and a minimal set of property fields can store unlimited strings, so that any implementation is

[4]Actually, $\mathcal{R}esourceome$ does not support rules. This item is introduced for further extensions.

possible. This layer is actually a forest, i.e. a tree for each activity: the child (hypoactivity) of a node activity (hyperactivity) is a more specific version of it. The activity also specifies additional constraints about the value of some of the parameters defined by its hyperactivity. These constraints, expressed in any form supported by Domain and Resource Ontologies, are used to determine the applicability of the activity with respect to the execution of its hyperactivity.

4.3 Hybrid Conceptualization in $\mathcal{R}esourceome$

As previously said, in the $\mathcal{R}esourceome$ knowledge model Resource and Task Ontologies are defined in OWL-DL, while the Domain Ontology is "hybrid", where "hybrid" stands for a conceptualization which is partly formal and partly semi-formal.

SKOS Core provides a model to express the basic structure and content of concept schemes such as thesauri, classification schemes, subject heading lists, taxonomies, many other types of controlled vocabulary and concept schemes embedded in glossaries and terminologies. In this scenario, our intention is to represent the Domain Ontology as a mix of ontology and thesaurus, where thesaurus stands for a taxonomy with broader/narrower terms, synonymous terms, top terms and related terms. Concepts in SKOS are instances of the *skos:Concept* class[5]. This allows the representation of both "is-a"-like and "part-of" relations using the same relation, as is common in many thesauri.

In this situation, OWL and SKOS are used side-by-side to model different parts of the same conceptualization: to better understand the mechanism, in the following the first example shows how we use OWL to define classes of Resource and their relevant properties, while the latter shows how we use SKOS to define the Domain concept.

```
ex:DataBase rdf:type owl:Class
ex:concerns rdf:type owl:DataRetrival
rdfs:domain ex:Resource
rdfs:range ex:Protein
ex:Protein rdf:type owl:Class
rdfs:subClassOf skos:Concept

ex:ProteinID rdf:type ex:Protein
```

[5]Note to the reader: *skos:Concept* is an *owl:Class*, but an instance of *skos:Concept* may also be an instance of *owl:ObjectProperty*, or of *owl:Class* itself

```
ex:SwissProtProteinIdentificator rdf:type ex:Protein
ex:SwissProtProteinIdentificator skos:broader ex:ProteinID
```

Finally, a relation (*concerns*) between Domain and Resource concepts links the two conceptualizations.

This kind of specification is perfectly compatible with OWL-DL, and does not place any constraints on our choices for the semantics of *skos:Concept*. Although SKOS and OWL vocabularies are used in the same graph, the "SKOS and OWL streams" are effectively kept separate. Relationships between concepts are captured using e.g. *skos:narrower*, *skos:broader* and *skos:related*, rather than *rdfs:subClassOf*. In the $\mathcal{R}esourceome$ knowledge model, every individual of *skos:Concept* or subclass of *skos:Concept* represents a concept. From a philosophical point of view, this kind of concepts represents the universal concept: due to this approach, we can define relationships between individuals and concepts.

Furthermore, while in SKOS*skos:broader* and *skos:narrower* properties are defined neither transitive nor intransitive, in the $\mathcal{R}esourceome$ knowledge model *skos:broader* and *skos:narrower* are assumed to be intransitive relations and two new relationships, namely *skos:narrowerTransitive* and *skos:broaderTransitive*, are defined to explicitly denote their transitive versions.

4.3.1 An example of Conceptualization in $\mathcal{R}esourceome$

In this example, the Resource Ontology formalizes the basic entities which Computer Science researchers interact every day with. Our intention is to annotate the literature resources as Articles and Books in the Formal Methods context. In this case we identify the Formal Method research topic, in particular the Process Algebra topics (Process Calculi). Two kinds of process algebra are CCS and ASP. The Resource Ontology describes the Article and Book artifacts, while the Domain defines the concepts that arise into Formal Methods topics. Using SKOS, we build a thesaurus of Research Topics, where Formal Methods is a concept schema and Process Algebra is a top concept of this schema as much as his synonymous Process Calculi. ACP and CCS are narrower concepts of both Process Algebra and Process Calculi. Using concerns relationship, we can classify the paper1 and paper2 in CCS and ASP respectively, as showed in Fig. 4.6

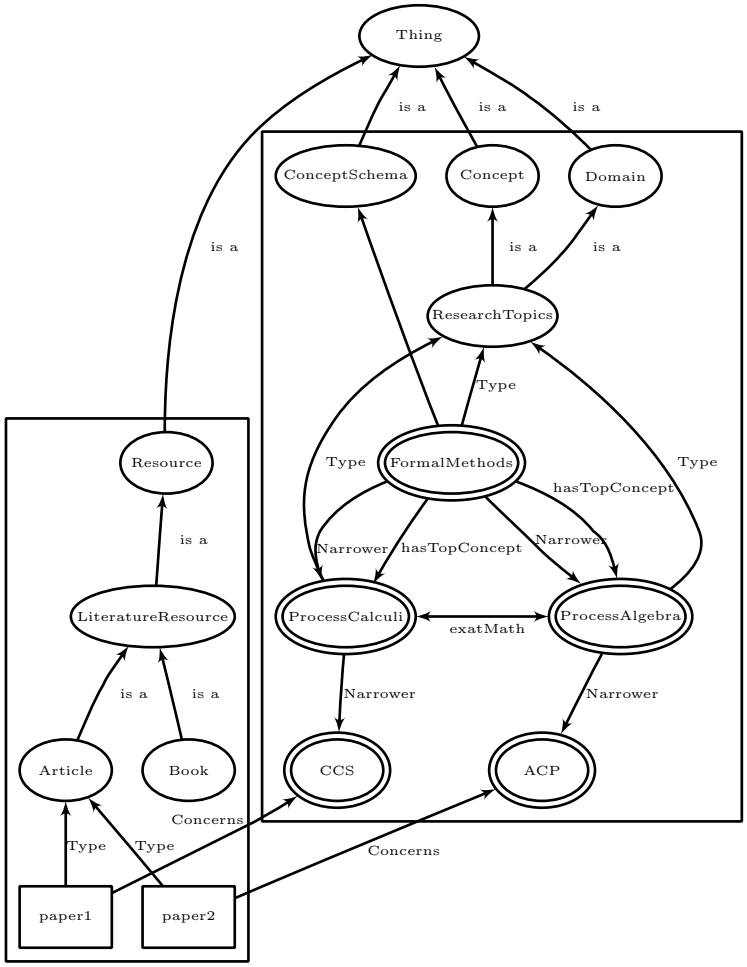

Figure 4.6: $\mathcal{R}esourceome$ ontology example.

4.4 Anti-Rigid Properties and Social Roles in $\mathcal{R}esourceome$

Some entities appear just because are social conventions in various ways on communities of agents; these kinds of entities can be social concepts like *Student*, *President*, *Engineer* or individuals like *Rossi is a Student at University of Camerino* or *Bianchi is a Ph.D Student at the same University*.

Intuitively, it is possible to distinguish two senses of sociality.

- In the first sense, here investigated, an entity is social if it is an immaterial product of a community: in this sense, "social" is roughly synonymous of "conventional". Guarino suggests that such roles must

be anti-rigid, i.e. they are properties that are contingent (non-essential) for all their instances. The ontological nature of the contexts that determine roles, and he individuates and analyzes in detail three kinds of role: *Relational Roles* (ways of participation in a relation), *Processual Roles* (ways of participation in an event), and *Social Roles*. Social Roles are concepts that can be played by certain entities, when they enter in relationships with other entities.

- Social roles (in the broader sense) and statuses have similar features: entities have statuses; statuses are created, accepted, and destroyed by a community of agents needing a notion of collective acceptance. Statuses are strongly connected to status functions which have the form: "X counts as Y in context C", where X is a (physical or non physical) entity, Y is a status, and C a context.

As introduced above, the $\mathcal{R}esourceome$ knowledge model allows to represent anti-rigid properties like social role in a flexible and intuitively way. Thinking about the relational role "enrolled", a Student is a Person enrolled in a University. But also a working student is a Person enrolled in some University. It is obvious that Working Student is Person enrolled in some University and working in some Enterprise. University system must also manage Enterprise for distinguishing this kind of Student.

In the $\mathcal{R}esourceome$ model, this scenario can be expressed as follows:

- $Student \equiv Person \sqcap \exists\, Enrolled.University \sqcap \exists\, playRole.\{\mathsf{StudentRole}\}$
- $WorkingStudent \equiv Person \sqcap \exists\, Enrolled.University \sqcap \exists\, playRole.\{\mathsf{WorkingStudentRule}\}$

This kind of restrictions describes *anonymous* classes (i.e. unnamed class): the *anonymous* classes contain all of the individuals that satisfy the restrictions, i.e. all of the individuals that have the relationships required to be a member of the class.

4.5 Related Works

As related work we would like to mention PROTON. The PROTON (PROTo ONtology) ontology has been developed in the SEKT project as a lightweight upper-level ontology, serving as the modeling basis for a number of tasks in different domains. In PROTON ontology, schema-ontology and topic-ontology describe respectively extensional semantics and intensional semantics. Anyway, PROTON as $\mathcal{R}esourceome$ suggests to separate this two kinds of ontology adopting the following principles:

- The class hierarchy of the schema-ontology should not be mixed with topic hierarchies.
- Representing topics as instances of the Topic class avoids the computational intractability inherent in allowing classes as property values.

The Topic class is a bridge between topic- and schema-ontologies. The specific topics should be defined as instances of the Topic class (or of a sub-class of it) (see Fig. 4.7, where InformationResource belongs to same schema-ontology and Topic is the top-class of topic-ontology). The Topic

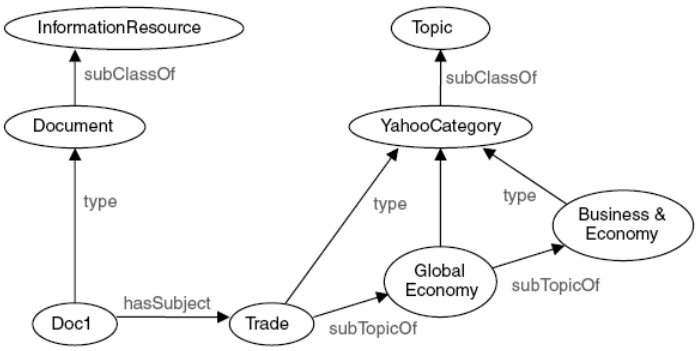

Figure 4.7: Proton topic modeling example.

class represents a top-class for linkage of logically informal taxonomies, while the instances of Topic are the only concepts in PROTON which are defined to serve as topics. We believe that this kind of approach is confuse and confusing. A "hybrid" conceptualization, as proposed in $\mathcal{R}esourceome$, provides a better way to represent this kinds of conceptualizations.

4.6 $\mathcal{R}esourceome$ and Ontology Evolution

Ontologies, similarly to the parts of the world that they describe, need to change continuously to be effective. In Stajanovic *et al.*(2002) the authors proposed a six-phase Ontology evolution process, as depicted in Figure 4.8.

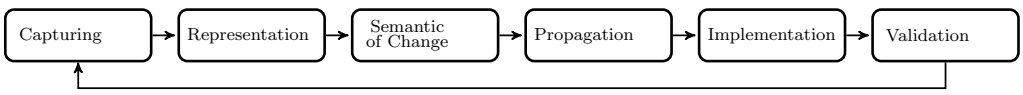

Figure 4.8: Ontology Evolution Process

The ontology evolution process starts with the *Capturing Change* phase, either from *explicit requirements* or from the *change discovery* results. In this phase, Ontology engineers adapt the ontology to some new requirements by explicit feedback about the ontology's usability that end-users provide: this type of changes are called explicit requirements or *top-down changes*.

In the following, we will sketch how $\mathcal{R}esourceome$ can support Ontology engineers to discover any anomaly during the *Capturing Change* phase and, in particular, how $\mathcal{R}esourceome$ results to be a suitable knowledge model for *top-down changes*.

4.6.1 Usage-Driven Ontology Changes by $\mathcal{R}esourceome$

One of the main issues in developing ontologies resides in the hierarchy of concepts. Stajanovic (2004) defines *data-driven change discovery* as the issue of deriving ontological changes from the ontology instances by applying techniques such as data-mining, Formal Concept Analysis (FCA) or various heuristics. For example, one possible heuristic might be:

If no instance of a concept C uses any of the properties defined for C, but only properties inherited from the parent concept, C is not necessary.

Another interesting example may be taken always form Stajanovic (2004):

Let us assume that in the initial hierarchy, the concept X has ten sub-concepts $(c_1, c_2, \ldots c_{10})$, that is an ontology engineer has found that these ten concepts correspond to the users' needs in the best way. However, the usage of this hierarchy in a longer period of time showed that about 95% of the users are interested in just three sub-concepts of these ten. This means that 95% of the users, as they browse the hierarchy, find irrelevant 70% of the sub-concepts.

In order to make this hierarchy more suitable to users' needs, there would be two useful ways for re-adapting the initial hierarchy:

Expansion: to move all seven irrelevant subconcepts down in the hierarchy by grouping them under a new sub-concept g (see Figure 4.9(c) taken from).

Reduction: to remove all seven irrelevant concepts, while redistributing their instances into the remaining sub-concepts or the parent concept 4.9(d) taken from).

Thanks to the resources sharing by its domain, the $\mathcal{R}esourceome$ model can play a role in the ontology evolution. Consider the next scenario:

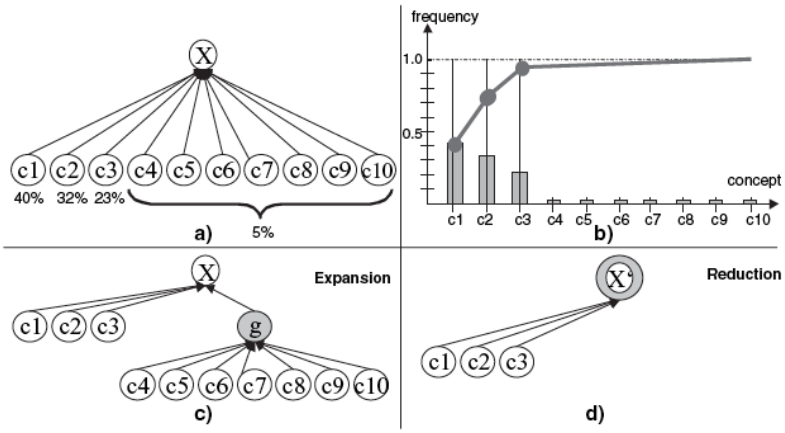

Figure 4.9: An example of the non uniformity in the usage of concepts.

Let us assume that in the initial hierarchy, the concept X represents some concept in the Resource Ontology hierarchy - proposed by an Ontology engineer - while the Domain Ontology contains the following concept definitions:

- $Concept(dc_1)$, $Concept(dc_2)$ and,
- $exatMatch(dc_1, dc_2)$,
- where $Concept(dc_1)$, $Concept(dc_2)$ stand for *skos:Concept* types .

Let us suppose that users add ten individuals $x_1, x_2, \ldots, x_{10} \mid X(x_i) \ \forall\, 1 \leq i \leq 10$. After a period of time, the users' annotations produces the following configurations:

Case 1(Figura 4.10):

- $concerns(x_i, dc_1), \ \ \forall\, 1 \leq i \leq 9$, and
- $concerns(x_{10}, dc_2)$.

This situation suggests to the Ontology engineers to expand the hierarchy by creating a new concept (class) $X_1 \mid X_1 \sqsubseteq X$ (see Figure 4.11) with an associated label that specifies the concept dc_1, being much relevant of dc_2 , and moves all instances $x_1, x_2, \ldots, x_{10}$ belong X_1.

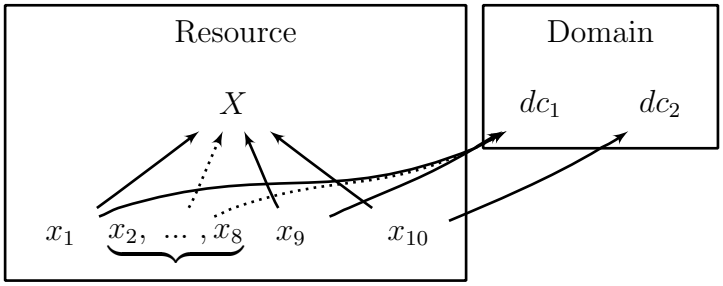

Figure 4.10: Case1

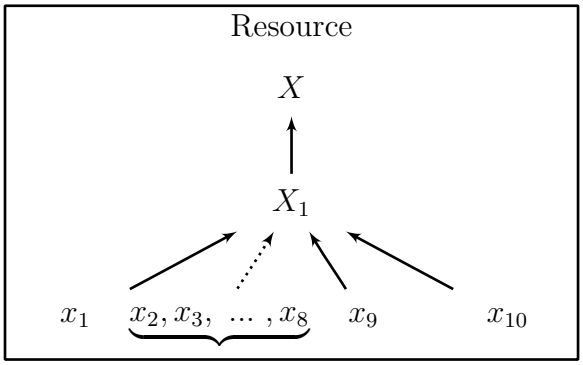

Figure 4.11: Case 1 Ontology Evolution.

Case 2:

- $concerns(x_i, dc_1)$, $\forall\ 1 \leq i \leq 5$, and
- $concerns(x_i, dc_2)$ $\forall\ 6 \leq i \leq 10$.

In this case, Ontology engineers can create two new concepts (classes) $X_1, X_2 \mid X_1 \sqsubseteq X$, $X_2 \sqsubseteq X$ and $X_1 \equiv X_2$ with associated labels that identify concepts dc_1 and dc_1, respectively. After that, all instances must be moved t.c. $X_1(x_i)$ $\forall\ 1 \leq i \leq 5$ and $X_2(x_i)$ $\forall\ 6 \leq i \leq 10$.

Part IV

Distributed Computation

Chapter 5

Distributed Workflow

This chapter is devoted to define a Petri net-based methodology showing how a user workflow-based application specification can be translated into Interactive Components. As a case study, this methodology will be applied in $\mathcal{H}ermes$ [48], an agent-based middleware, for the design and the execution of activity-based applications in distributed environment introduced above (see Section 3.3.1 in the background chapter). All these notions are preparatory to understand Chapter 7, where finally we will present ad integrated framework consisting of $\mathcal{R}esourceome$ and $\mathcal{H}ermes$ for running semantic workflows in distributed environments.

5.1 Workflow Management System

Workflow Management Systems (WMSs) provide an automated framework for managing intra- and interprise business processes. A WMS is defined by WfMC as: *"A system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications."* [61]. The most part of implemented WMS are based on a client/server architectural style. In these systems, the workflow enactment is entrusted to a central component, that acts as a server and is responsible for the correct execution. These systems lack the flexibility, scalability and fault tolerance required for a distributed cross-organizational workflow; in fact a monolithic architecture does not allow the execution of workflow or parts of it over distributed and heterogeneous systems. To overcome these limitations, a Migrating Workflow Model (MWM) has been proposed in [47]. In this model, instances of a

workflow or parts of it can migrate; i.e., it is possible to transfer the code and the whole execution state, including all data gathered during the execution, between sites participating in workflow's execution. This model provides two main benefits. First, migrating workflow decreases efficiently traffic network; usually code implementing workflow specification is less heavy to transfer than the amount of data needed during its execution. The second asset concerns the possibility for the workflow to be executed even in mobile and weekly network connected devices. This model requires a suitable middleware to guarantee code mobility support.

5.2 Petri nets

A Petri net [57] is a directed bipartite graph with two node types called places and transitions. The nodes are connected via directed arcs. Connections between two nodes of the same type are not allowed. Places are represented by circles and transitions by boxes or bars. According to [56], an *ordinary* Petri net can be defined as a 4-tuple, $PN = (P, T, F, M_0)$ where:

1. $P = \{p_1, p_2, \cdots, p_m\}$ is a finite set of places,
2. $T = \{t_1, t_2, \cdots, t_n\}$ is a finite set of transitions,
3. $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs (flow relation),
4. $M_0 : P \to \mathbb{N}$ is the initial marking function,
5. $P \cap T = \oslash$ and $P \cup T \neq \oslash$.

Ordinary means that all arcs have weight 1.

A place p is called an *input place* of a transition t if and only if there exists a directed arc from t to p. Place p is called an *output place* of transition t if and only if there exists a directed arc from p to t. We use $\bullet t$, $t\bullet$ to denote respectively the set of input places and the set of output places a transition t. The notation $\bullet p$ and $p\bullet$ identifies instead the set of transitions sharing p as input place and as output place respectively.

At any time a place contains zero or more tokens, drawn as black dots. A marking function $M \in P \to \mathbb{N}$ is the distribution of tokens over places and represents the state of PN. In this definition we do not consider any *capacity restrictions* for places. The number of tokens may change during the execution of the net.

Transitions are the active components in a Petri net: they change the state of the net according to the following *firing rule*:

1. A transition t is said to be *enabled* if and only if each input place p is marked with at least one token.

2. An enabled transition may *fire*. If transition t fires, then t *consumes* one token from each input place p of t and produces one token in each output place p of t.

5.3 High level Petri nets

A High-level Petri Net (HLPN) [41] is a PN with three main extensions:

- *Extension with color* - in Coloured Petri Net (CPN) [55] tokens are typed and each token has a value often referred as *color*. Transitions determine the values of the produced tokens on the basis of the values of the consumed tokens. Moreover preconditions can be specified taking into account the color of tokens.

- *Extension with time* - using time extension, tokens receive a *timestamp* value that indicates the time from which the token is available. A token with timestamp 10 is available for the consumption by a transition only from moment 10. A transition is enabled only at the moment when each of the tokens to be consumed has a timestamp equal or subsequent to the current time.

- *Extension with hierarchy* - hierarchical extension allows to model complex processes more easily by dividing the main process into ever-smaller subprocesses to overcome the complexity. Here we use the notation proposed by Wil van der Aalst [62], where a subprocess is a transition represented by double-border square as Figure 5.1 shows.

5.3.1 Workflow Nets

Workflow Nets (WF-nets) [62] are a subclass of HLPN where tasks are represented by transitions and conditions by places. A WF-net satisfies two requirements. First of all, it must contain at least two special places: i and o. Place i is a source place with $\bullet i = \oslash$. Place o is a sink place with

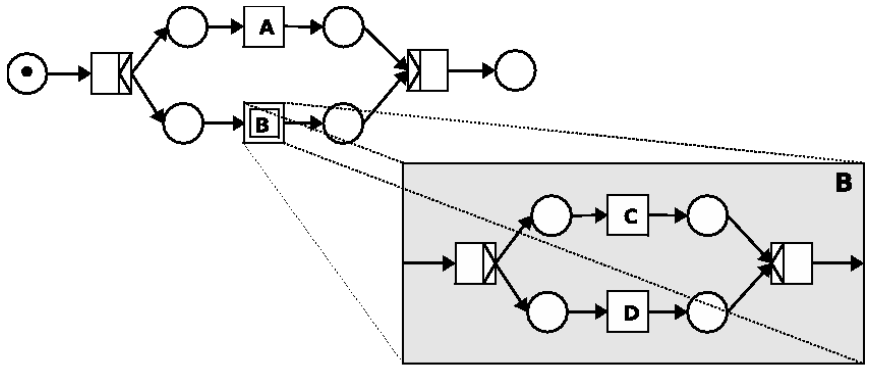

Figure 5.1: A subprocess

$o\bullet = \oslash$. Secondly, it must hold that if we add a transition t^* which connect place o with i - i.e. $\bullet t^* = \{o\}$ and $t^*\bullet = \{i\}$ - then the resulting Petri net is strongly connected -from each node there exists a directed path to every other node-. This requirement avoids dangling tasks and/or conditions. In order to make the WF-net suitable for workflow process modelling a set of notational extensions was applied to the standard Petri net definition. In particular, as referred [62], the author of WF-net added to the classical Petri net transition a set of special transitions (AND split, AND join, XOR split, XOR join, AND/OR split), shown in Figure 5.2 with their translations, to express branching decisions in a more compact and user friendly way.

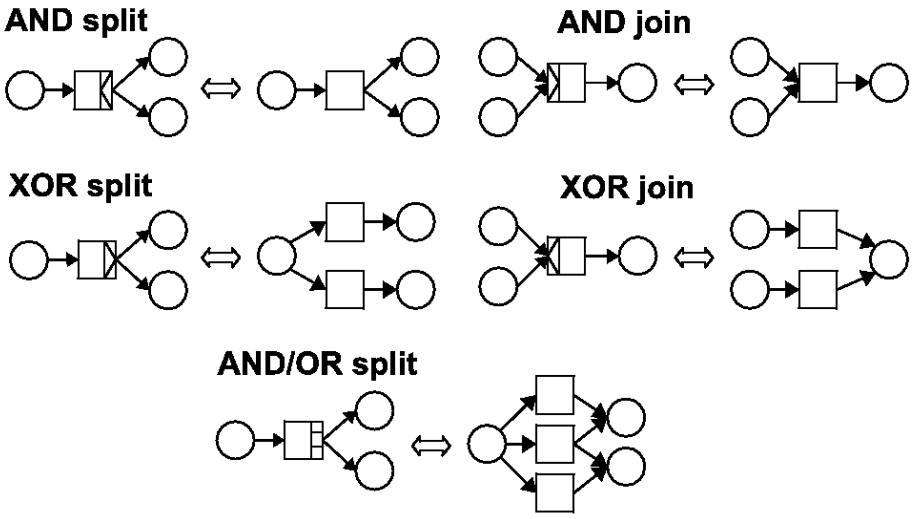

Figure 5.2: Special transitions and their translation

In the workflow theory [60], routing primitives are defined as a set possible basic patterns that determine which tasks need to be performed and in which order.

Using the prevoius defined special transitions as control flow, a set of four basic routing primitives can be obtained as Figure 5.3 shows:

1. *Sequential routing* - task A is executed before task B,
2. *Alternative routing* - either task A or task B are executed non deterministically,
3. *Concurrent routing* - task A and task B are executed concurrently,
4. *Iterative routing* - task B is repeated.

In order to model dependencies between the workflow process and its operative environment three different constructs named "triggers" were added to the standard Petri net - resource, message and time trigger. In this paper we will consider only the resource trigger. In this particular case, a trigger is associated to a specific resource needed to execute a task. As Figure 5.4 we can consider a trigger as special place linked with the transition representing a task. When the needed resource is not available this place is empty and

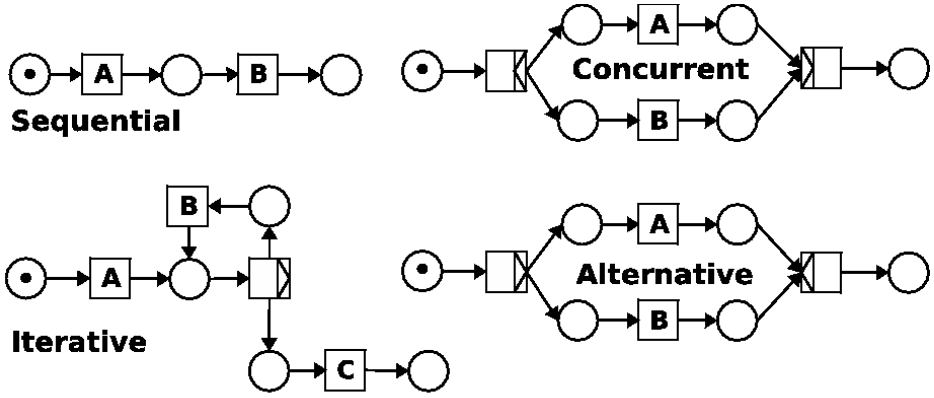

Figure 5.3: Routing primitives

the transition is not enabled, while if it contains a token it means that the resource is available and the task related to the linked transition could be executed. In the following sections we will consider interactive components as computational resources able to execute tasks under particular cases. The resource trigger can be assigned to every transition and is represented by a small, self-explaining icon (⇓) near the associated transition symbol as Figure 5.4 shows.

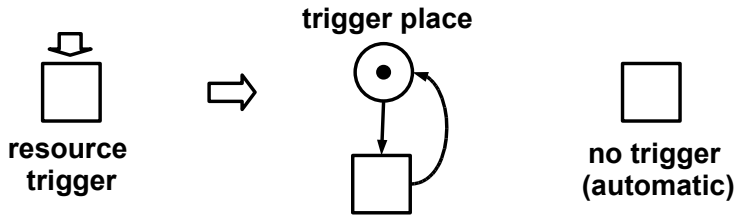

Figure 5.4: Resource trigger

5.4 Adding roles to Workflow Specification

A workflow process specification defines which tasks need to be executed and in what order. A set of cases, identified by pre- and postcondition, are handled by executing tasks in a specific order. A task which needs to be executed for a specific case is called *work item* [59]. A workflow specification is the composition of both primitive and complex work items. A primitive work item can be directly executed. A complex work item - called subprocess in [59] - must be specified before it can be used; the specification of a subprocess is a workflow of complex and primitive work items. By using subprocesses the specification of workflows is simplified because they enhance both hierarchical specification and reuse: we can use an already existing subprocess without having care of its specification. Work items are generally executed by a *resource* that can be either a machine - i.e. printer or a fax -, a computational entity - i.e. an agent - or a person. Resources are allowed to deal with specific work items. Grouping resources into classes facilitates the allocation of work items to resources. A resource class based on the capabilities of its members is called *role*. A work item which is being executed by a specific resource is called an *activity*.

A workflow designer, whose primary expertise is generally in the application domain, should be free to focus on coordinating domain specific activities rather than being concerned with the complexity of a domain specific activity or resources involved to execute it. Users in fact may ignore the topological organization of the distributed environment and resource classes available.

The first step of the proposed methodology translates a user workflow specification to a role-based workflow specification. During this step each work item is assigned to a role able to perform it. This operation could be made manually or automatically. In the first case, an expert user can assign role by itself, while in the second case an activity repository store

all informations about complex activities and the user knows only there is an automatic mapping from domain specific work items and activities. This resource allocation is applied recursively in all work items of each subprocess.

Figure 5.5 shows an example in Bioinformatics. In this case, a bioscientist has designed an in-silico experiment - shown on the top of the Figure 5.5 - to globally align some omologous sequences to a given one. This workflow involves five main *work items*:

1. *get_gene_seq* - given a gene id, retrieves the gene DNA sequence,
2. *search_genbank_omologous* - given a DNA sequence, retrieves a set of DNA sequence omologous from NCBI Genbank [43],
3. *search_PDB_omologous* - given a DNA sequence, retrieves a set of DNA sequence omologous from the Protein Data Bank (PDB)[44],
4. *merge_seqs* - merges two or more set of sequences in a set of sequences,
5. *global_alignment* - given a set of DNA sequences, calculates the global alignment.

In the Role-based Workflow Specification - shown on the bottom of Figure 5.5 - subprocesses *search_genbank_omologous*, *search_PDB_omologous* and *merge_seqs* are substituted with the corresponding set of primitive work items. Each primitive work item is assigned to a specific role. In this case we have three roles A, B and C. Roles are translated into Interactive Components in the next step.

5.5 Interactive Components Specification

In the second step, the Role-based Specification is translated into Interactive Components. In order to specify the behaviour of each component independently from the corresponding generated code, we use BRICs [49], another Petri net-based notation.

In the following, we provide transformation rules to translate WF-net to BRICs notation.

5.5.1 BRICs notation

Block Representation of Interactive Components (BRICs) [49] is an high-level language for the design of MultiAgent systems based on a modular approach. A BRIC component - see Figure 5.6 (a) - is a software structure characterized

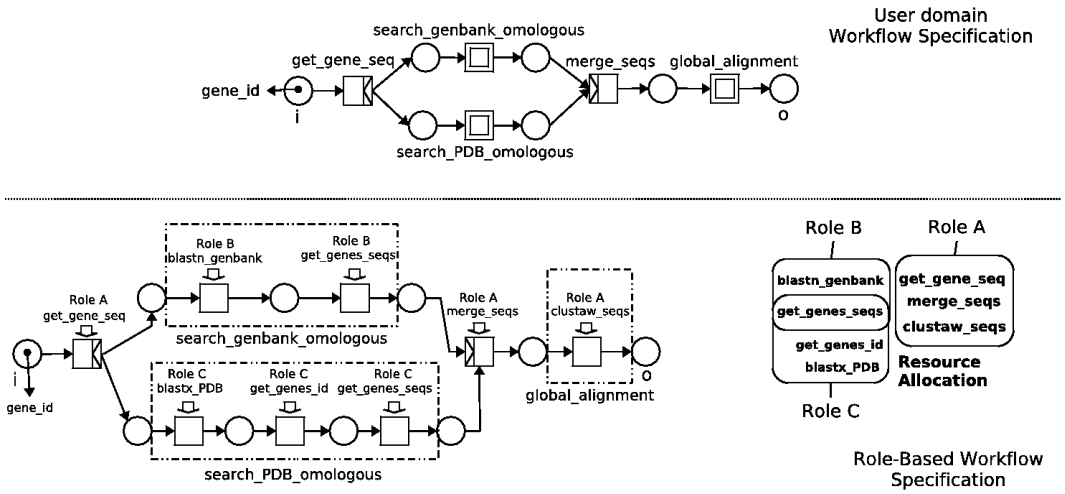

Figure 5.5: From User to Role-Based Workflow Specification

externally by a certain number of input and output terminals and internally by a set of components.

Every component is an instance of a class, which describes its internal structure. A structured component is defined by the assembly of the its subcomponents. The input terminals of the structured components are linked to the input terminals of the the sub-components and is also possible to combine terminals of the composite components with sub-components as showed in Figure 5.6 (b).

The behaviour of elementary components is described in terms of a Petri net-based formalism. The default net formalism normally used in BRIC is colored Petri nets with inhibitor arcs.

Figure 5.6 (c) represents the general form of a transition. A transition is defined by *entry arcs*, *exit arcs* and *pre-condition* of activation. Entry arcs are carriers of a condition, in the form of the description of a token including variables. When the place contains a token corresponding to this description, the arc is validated. There are three categories of entry arcs:

1. *Standard arcs*, denoted $a_1, \cdots, a_n$, trigger the transition only if they are all validated consuming tokens which act as triggers and deleting them from input places.

2. *Inhibitor arcs*, denoted $i_1, \cdots, i_m$, inhibit the triggering of the transition if they are enabled without deleting tokens from the input place.

3. *Non-consumer arcs*, denoted $b_1, \cdots, b_k$, work as standard arcs, but they do not delete the input tokens.

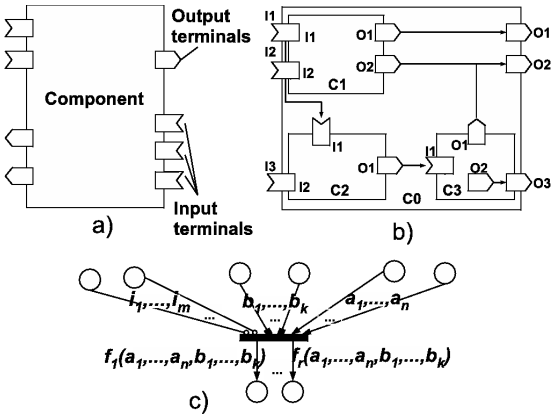

Figure 5.6: BRICs notation

An exit arc associates a transition with an output place producing in this position new tokens that depend on the tokens used for triggering the transition. The pre-condition associated with a transition relates to the external conditions. The components communicate by exchanging information along communication links which connect output terminals to the input terminals. Information is transported through the net in the form of tokens.

A token is either an elementary piece of information whose value is a mere presence or absence, or a predicate in the form $p(l_1, \cdots, l_n)$, where each l_i represents a number or a symbol in a finite alphabet. Other important assumptions concerning this notation are:

1. Input terminals are considered as places, thus names of input terminals are taken to be place identifiers.

2. Any direct link between an input terminal of an incorporating component and an input terminal of incorporated component is assumed to comprise a transition, in accordance with Petri net design rules.

5.5.2 Mapping roles with structured components

The translation from a role-based workflow to interactive components specification requires the definition of a structured component skeleton that represents a role-specific implementation.

As Figure 5.7 shows, the basic skeleton has two essential capabilities. First, since it must be able to receive messages from the other external components asynchronously, we specify a subcomponent called *MessagesQueue*

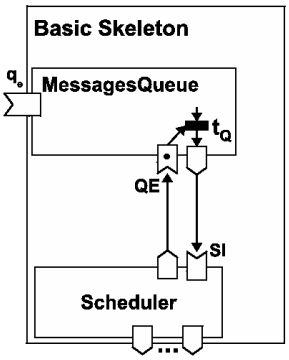

Figure 5.7: Basic skeleton component

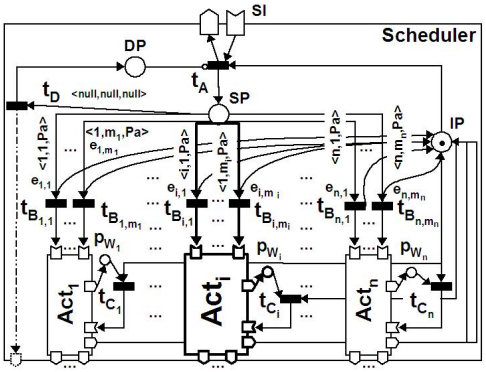

Figure 5.8: Scheduler component

that stores messages as colored tokens following a First In First Out (FIFO) approach. Each message is defined in the form:

```
<sender>: <address> << <Act, Pre, Pa>
```

where *sender* is the identifier of the component sending the message, *address* is the identifier of the component to which the message is addressed. `Act` and `Pre` are respectively the activity to be chosen and the pre-condition to be set, *Pa* is a possible input parameter for the activity -*null* value means no parameters.

In the basic skeleton we specify a second subcomponent, called *Scheduler*, providing, as Figure 5.8 shows, a set of places and transitions to receive tokens from *MessagesQueue* and to schedule the execution of a set of tasks following the order and cases defined by the role-based workflow specification. Scheduler component has four main places:

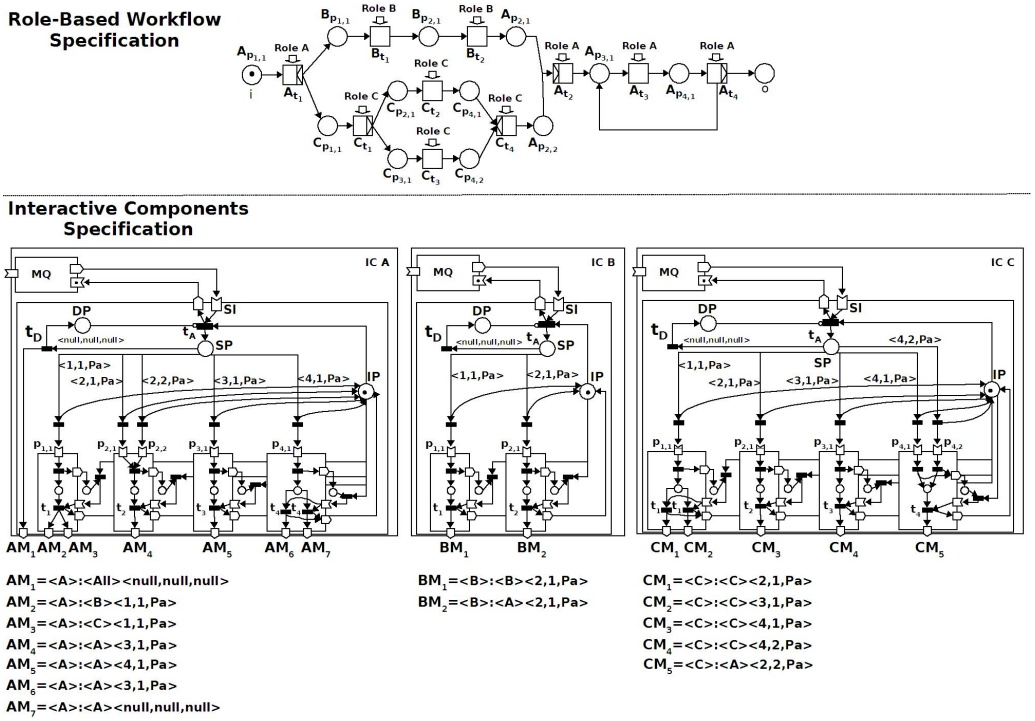

Figure 5.9: From Role-Based Workflow to Interactive Components Specification

1. *Scheduler Input* (SI) - a token in this place means a new message for the scheduler.

2. *Schedule Place* (SP) - after t_A firing produces a colored token in SP in the form:

   ```
   <Act, Pre, Pa>
   ```

   Each *Scheduler* component contains a set of n *Act* components and $\forall t_{B_{i,k_i}}$ we define an entry arc e_{i,k_i} with the description:

   ```
   <i, k, Pa>
   ```

   where $1 < i < n$, $1 < k_i < m_i$ and m_i is number of pre-conditions for Act_i. A token in SP matching with a description of an entry arc e_{i,k_i} enables the corresponding transition $t_{B_{i,k_i}}$. The entry arc description for the transition t_D is defined as:

   ```
   <null, null, null>
   ```

3. *Idle Place* (IP) - when this place contains a token the *Scheduler* is waiting for a new message.

4. *Dead Place* (SP) - the transition t_D when is enabled produce a token in SP inhibiting the transition t_A. Consequently the *Scheduler* cannot receive any token in SP place. This place is called dead, because a token here stops the behaviour of this component. When this happens t_D can produce also a token for the external components to stop their behaviour too:

```
<me>: <All> << <null, null, null>
```

5.5.3 Mapping activities

An Interactive Component (IC) is an executor of a piece of workflow specification. The final behaviour of an IC is obtained by plugging the activities of the corresponding role into the basic skeleton previously defined. Each primitive activity defined in the Role-based specification is associated with an *Act* component in ICs specification. Figure 5.10 shows how the routing constructs in Figure 5.3 are mapped into *Act* components.

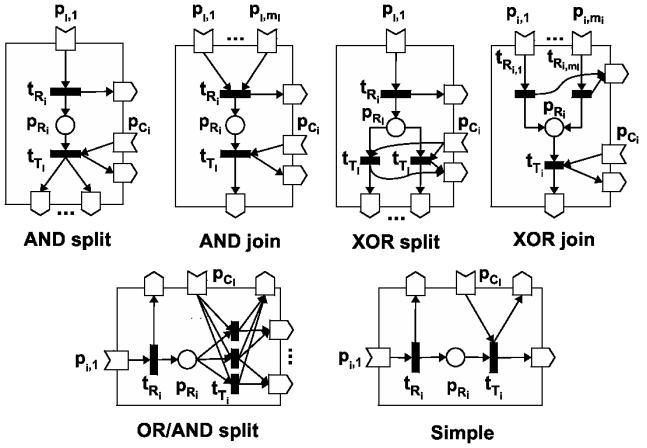

Figure 5.10: Mapping activities

A component Act_i contains an input terminal for each pre-condition of the mapped activities, which are labelled $p_{i,1}, \cdots, p_{i,m_i}$ where m_i is the number of the activity pre-conditions. When the routing transition t_{R_i} fires the token produced in p_{R_i} enable the task transition t_{T_i} -representing a task to be execute by an IC- is enabled iff IP is not empty. The colored token produced by t_{T_i} is a message - as previously defined - for its and/or other ICs

MessageQueue.

5.5.4 An example

Figure 5.9 shows, on the top a Role-based specification using all possible routing primitives and on the bottom the translation in ICs specification. For each role in the first corresponds to an IC in the latter. All pre-conditions and activities are mapped into Act components adding the right routing transitions and are plugged in the Scheduler of the IC basic skeleton.

An Act component produce at least a message describing which are the next IC, Act component and input terminal to be reach and an optional parameter for the task transition. The field *address* in the message specifies which are the receiver IC and an entry arc is assumed from this output terminal and the external input terminal of the IC specified.

Chapter 6

Web Service Integration

Web Services are *SOA*-based software components that expose specific functionality on the Internet (see Chapter 2 to recall the *SOA* paradigm). For this reason, they can be considered *resources* - and, as a consequence, can be conceptualized as resources in the $\mathcal{R}esourceome$ model - and can be involved in the definition of workflows - i.e. can be invoked by specific tasks or activities in the $\mathcal{R}esourceome$ model by Web protocols and data formats such as HTTP and XML.

However, it is well-known that it is very hard to guarantee automatic interoperability among different applications. This is particularly evident in the Web scenario, where clients perform run time queries in search of services, services provide some given capabilities, and both systems try to automatically communicate by exchanging messages.

Correct service invocations are very difficult to achieve for several reasons as, for instance, lack of semantic information exposed by the service interaction interfaces, their heterogeneity and the fact that the set of exchanged messages between client and service - called *conversation* - is not based on a fairly rigid request-response interaction style. Several standards and models have been defined in order to fill this gap:

- On the lowest layer, the Web Service Description Language (WSDL) [96] is the standard used for publishing abstract and concrete descriptions of Web Services - including the schema of exchanged messages, the name and type of operations that the service exposes and some simple interaction patterns. A WSDL document is analogous to a collection of methods and their signatures, together with information about the syntax of the inputs and the outputs. In particular, the parameters types are defined by XML Schema [92], which provides a simple syntax to constrain the operation parameters but does not allow complex con-

straints to be expressed (for instance, in XML Schema it is impossible to define inter-dependencies and dynamic constraints).

- On the middle layer, a multitude of specifications for describing conversations - some examples are [120], [122], [121] and [123] - defines a structured language expressing (temporal, priority, etc.) relationships between the exchanged messages.

- On the uppermost layer, concrete platforms provide the client-side generation of stubs from XML Schema-typed WSDL descriptions for single and multi-step invocations of Web Services. Stubs allow Web Services to be invoked by programming languages (e.g. Java). Their basic functionalities are marshalling and unmarshalling of service parameters: the former functionality is used to serialize parameters to XML format and to produce a SOAP [97] message, while the latter acts as the vice versa.

Since on the one hand XML Schema cannot express complex constraints and, on the other hand, the available models and specifications for client-server conversations do not express constraints both on message parameters and transitions among exchanged messages, a classic stub does not allow a service to be properly invoked and, as a consequence, does not allow a *valid* client-server conversation based on *valid* exchanged messages to be realized - where *valid* is intended w.r.t. message parameters and transitions among exchanged messages.

In this scenario, we propose a theoretical and practical framework for realizing a dynamic monitoring/validation of *valid* sequences of *valid* invocation messages. Our contribution is structured as follows:

- first, we define a framework that preserves static and dynamic integrity constraints of invocation parameters. The main ingredients of the framework are: WSDL, CLiX [73], a language for constraint specification in XML that allows the specification of static and dynamic integrity constraints, and reflection mechanisms for managing complex user-defined types. The proposed framework is entirely based on XML-based technologies and allows only provably correct Web Services invocations be forwarded by client-side checking CLiX formulas.

- then, we define two conversation specification language-independent models - the first based on first-order guarded automata and the latter based on attribute grammars - for describing *valid* client-server conversations.

- finally, we build a parser for realizing a dynamic monitoring/validation of *valid* sequences of *valid* invocation messages.

6.1 A CLiX-constrained WSDL interface

The first step consists of defining an XML-based framework allowing the client-side automatic generation of stubs, for properly invoking Web Services. The proposed framework is presented schematically in Fig. 6.1 and is described in detail in the following.

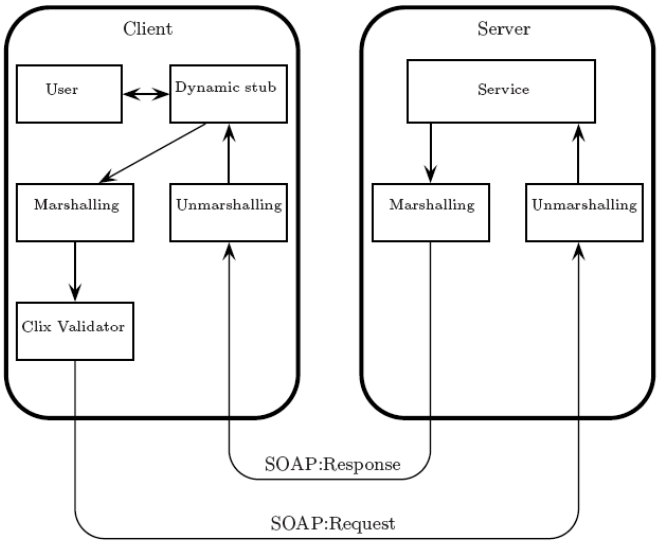

Figure 6.1: Client-side automatic generation of valid stubs.

The core of this framework is CLiX, an XML-based logic language allowing constraints on XML documents to be expressed by a mixture of First-order logic and XPath expressions. On the server side:

- the service WSDL description is *enriched* with CLiX logic formulas, expressing static and dynamic integrity constraints of the invocation parameters, and CLiX formulas are linked to the WSDL document by a namespace[1].

[1]It follows that CLiX is independent from the service description, i.e. every (XML) formalism can be freely chosen to describe the service.

- the Web Service makes available the WSDL description of its signature, the XML Schema modeling inputs and outputs and the set of its specific CLiX formulas.

On the other side, the client activity consists on two steps:

- producing a stub by means of a concrete platform, starting from the WSDL description and the XML Schema.

  For our purpose, WSDL2Java [80], a concrete platform translating WSDL descriptions into stubs for invoking services by Java language, has been modified in order to allow a complex types handling *on the fly*.

- checking the (XML) serialized translation of the generated stub w.r.t. the set of CLiX formulas, before forwarding the service invocation.

  The marshalling operation is enriched with OpenCLiXML[74], an open source Java implementation of the freely available CLiX specification from Systemwire [119], in order to support the validation of (XML) serialized stubs w.r.t. both XML Schema and CLiX formulas. In particular, the validator provides optimized rules processing against data represented in XML, including inter-document checks.

6.1.1 An Example: Electric Circuits and Kirchhoff's First Law

A lot of computational problems (flow control, GIS, optimization, etc.) make use of graphs; constraints can be very complex and can involve several parameters. Consider, for instance, a Web service able to elaborate descriptions of electric circuits verifying Kirchhoff's First law. The service models a circuit as an oriented and weighted graph. Vertexes and edges symbolize crossing points and conductors, respectively. The edges are oriented and weighted, since direction and intensity are parameters characterizing the electricity.

In this case, a circuit verifies Kirchhoff's First law if and only if

> *At any point in it, where charge density is not changing in time, the sum of currents flowing towards that point is equal to the sum of currents flowing away from that point.*

A *sound* electric circuit, i.e. an electric circuit verifying Kirchhoff's First law, can be modeled by a graph, equipped by the following constraints:

1. There is no loop;

2. There is no *source vertex*, i.e. vertex having no incoming edge;

3. There is no *shaft vertex*, i.e. vertex having no outgoing edge;

4. For every vertex, the sum of incoming edges values equals to the sum of outgoing edges values.

Notice that some integrity constraints can not be validated at run time, i.e. on a partial graph, but only when the graph is completed. It suffices to consider item 4: whenever a vertex is created, the constraint expressed by item 4 does not hold as long as appropriate edges are added. Even preconditions and postconditions are not useful to validate this constraint before completing the graph. This fact justifies the use of a validator *after* marshalling input and output parameters.

Figure 6.2 shows graphically the XML Schema, embedded in the service WSDL document, modeling the input type (e.g. the graph).

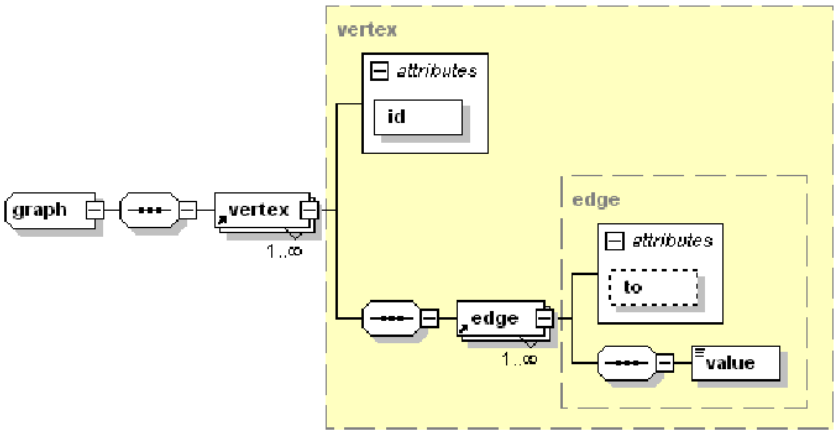

Figure 6.2: XML Schema

The following rules model the constraints expressed in items 1 and 2, respectively:

```
<clix:rule id="rule-item1">
    <clix:forall var="vertex" in="/graph/vertex">
      <clix:forall var="edge" in="$vertex/edge" >
          <clix:notEqual op1="$vertex/@id"  op2="$edge/@to" />
      </clix:forall>
    </clix:forall>
</clix:rule>
```

```
<clix:rule id="rule-item2">
    <clix:forall var="vertex" in="/graph/vertex">
      <clix:exists var="edge" in="/graph/vertex/edge" >
          <clix:equal op1="$vertex/@id"  op2="$edge/@to" />
      </clix:exists>
    </clix:forall>
</clix:rule>
```

Regarding item 3, both the constraint expressed by XML Schema (e.g. every vertex has at least one edge) and CLiX rule-item1 suffice to ensure that there is no shaft node in the input graph.

Finally, item 4 can be expressed in CLiX as follows:

```
<clix:rule id="rule-item4">
   <clix:forall var="vertex" in="/graph/vertex">
       <clix:forall var="edge" in="/graph/vertex/edge">
           <clix:equal op1="sum($edge/value)"
                       op2="sum($edge/@to=$vertex@id)"/>
       </clix:forall>
    </clix:forall>
</clix:rule>
```

6.2 Modeling valid constrained client-server conversations

The next step consists of defining a language-independent model for describing *valid* client-server conversations based on *valid* exchanged messages - where *valid* is intended w.r.t. message templates and transitions. Without loss of generality, we only assume to handle a generic XML-based document describing conversations, a WSDL document describing message schema[2] and CLiX.

The first model we propose is obtained by extending the boolean guarded automata model [113] as follows: (i) by imposing *first-order logic* guards, (ii) by expressing them as CLiX rules, (iii) by describing message types as WSDL operation parameters, as well as (iv) by optimizing the automaton memory representation as a finite vector of XML variables and (v) by modeling the automaton history - made of sent/received message instances - as a simple concatenation of XML documents.

The second model we propose is based on *attribute grammars* [115] - i.e. *context-free grammars* equipped with a finite set of *attributes*, a set of

[2]The proposed framework also fits on a scenario where message templates are described by XML schema [92].

evaluation rules and a set of *logical conditions* for attribute values - which is more suitable than the first model to build a parser for realizing a dynamic monitoring/validation of *valid* sequences of *valid* invocation messages.

6.3 The FOG automata model

In this section, we define the first model for *valid* first-order constrained client-server conversations, where *valid* is intended w.r.t. a set of CLiX rules.

Notation 6.3.1 We denote by W_c a generic (XML-based) document describing a client-server conversation; by W_m a generic XML-based document containing the templates of any W_c conversation message, and by $\mathcal{G}$ a set of CLiX rules constraining W_c and W_m (message) XML elements; by $M = \{m_k | k \in [1...n], n \geq 1\}$ the finite set of message types involved in W_c and described in W_m; by M_i and M_o the finite sets of respectively inbound and outbound message types in $M = M_i \cup M_o$; by $x(d)$ the d's XML scheme.

First, we abstract from the tuple $\langle W_c, W_m, \mathcal{G} \rangle$, replacing it with its guarded automaton-based representation.

Definition 6.3.1 A *First-Order guarded (FOG) automaton* associated to $\langle W_c, W_m, \mathcal{G} \rangle$ is $\mathcal{A} = \langle S, M, H, q_0, q_n, \delta, \mathcal{G} \rangle$, where:
i. $S = \{q_t \mid t \in [0...n], n \in \mathbf{N}\}$ is a finite set of states;
ii. $M = M_i \cup M_o$ is as above described;
iii. $H = \langle h_1, ..., h_{|M|} \rangle$ is a vector of XML local variables, where $\forall j \in [1...|M|], v_j$ is associated to $m_j \in M$;
iv. $q_o \in S$ is the initial state and $q_n \in S$ is the final state[3];
v. $\mathcal{G} = \{g_{(i,k)} = g(q_i, m_k, \langle d_1, ..., d_{|M|} \rangle)$ CLiX rule$\}$, such that $q_i \in S$, $m_k \in M$ and $\forall j \in [1...|M|]$, $d_j \in \{h_j(i), \lambda\}$.
vi. $\delta = \{(q_i, (l, g_{(i,k)}), Q_i)\}$ is a *state transition* relation, where $q_i \in S$, $Q_i \subseteq S$, $l \in \{m_k | \, m_k \in M_i\} \cup \{\overline{m}_k | \, m_k \in M_o\}$ and $g_{(i,k)} \in \mathcal{G}$.

Message types and local variables are XML documents. Each local variable h_j in H corresponds to a message types m_j in M. $\forall q_i \in S$ and $\forall j \in [1...|M|]$, $h_j(i)$ denotes an XML document - the *history* of m_j until the state q_i - obtained by enqueuing all the sent/received message instances, until the state q_i, that correspond to the type m_j. Each transition $\tau \in \delta$ is in one of the following two forms:
(receive-transition) $\tau = (q_i, (m_k, g_{(i,k)}), Q_i)$, where $m_k \in M_i$: the transition nondeterministically changes the state of the automaton from q_i to $q_p \in Q_i$,

[3] q_0 and q_n can coincide.

it removes the received message instance (of type m_k) from the input queue and it updates h_k in H, corresponding to m_k, by the concatenation of the received instance, in the case $g_{(i,k)}$ holds;
(*send-transition*) $\tau = (q_i, (\overline{m}_k, g_{(i,k)}), Q_i)$, where $m_k \in M_o$: the transition nondeterministically changes the state of the automaton from q_i to $q_p \in Q_i$, it appends the sent message instance (of type m_k) to the input queue of the client and it updates $h_k \in H$, corresponding to m_k, by the concatenation of the sent instance, in the case $g_{(i,k)}$ holds.

Definition 6.3.2 Let $\mathcal{A} = \langle S, M, H, q_0, q_n, \delta, \mathcal{G} \rangle$ be a FOG automaton associated to $\langle W_c, W_m, \mathcal{G} \rangle$. Given a guard $g_{(i,k)} = g(q_i, m_k, \langle d_1, ..., d_{|M|} \rangle) \in \mathcal{G}$, $\langle d_1, ..., d_{|M|} \rangle$ denotes the *actual context* in q_i of $g_{(i,k)}$, obtained filtering out all the local variables such that no XML attribute of theirs is involved in $g_{(i,k)}$.

Notation 6.3.2 Let W_m be a WSDL document. We denote by O_m the set of operation in W_m; for every $o \in O_m$, by $p_{in}(o)$ and $p_{out}(o)$ the sets of respectively input and output/fault parameters of o. We also assume that W_c and W_m are related as follows: for each operation $o \in O_m$, for every $p_k \in p_{in}(o)$ (resp. $p_{out}(o)$), for every $m_k \in M_i$ (resp. M_o), $x(m_k) = x(p_k)$ holds. We formally define this kind of relationship between W_c and W_m as follows.

Definition 6.3.3 Let $\mathcal{A} = \langle S, M, H, q_0, q_n, \delta, \mathcal{G} \rangle$ be a FOG automaton associated to $\langle W_c, W_m, \mathcal{G} \rangle$, and let W_m be a WSDL document. $\mathbf{W} = \langle \mathcal{A}, W_m \rangle$ is *stable* if and only if $\forall q_{i_1} \in S$ such that $(q_{i_1}, (m_{k_1}, g_{(i_1,k_1)}), Q_{i_1}) \in \delta$:

i. $\exists o \in \mathcal{O}_m$ s.t. $p_{in}(o) = \{p_{k_1}\}$ and $x(p_{k_1}) = x(m_{k_1})$;

ii.$\exists q_{i_h} \in Q_{i_1} (2 \leq h \leq 3)$ s.t. $(q_{i_h}, (\overline{m}_{k_h}, g_{(i_h,k_h)}), Q_{i_h}) \in \delta$ iff $p_{out}(o) = \{p_{k_h} \mid 2 \leq h \leq 3\}$ and $x(p_{k_h}) = x(m_{k_h})$.

The *stability* assumption (Definition 6.3.3) implies that it is possible to use everywhere the WSDL operation parameter p_k in place of the message m_k, and that the actual context of any guard in $\mathcal{G}$ only involves W_m operation parameter XML schema.

6.4 The FOA grammar model

In the following, we introduce some background knowledge on the attribute grammar model. Expert readers can skip the topic and jump to Section 6.5.

Definition 6.4.1 A *context-free grammar* is a tuple $G = \langle N, T, s, \mathcal{P} \rangle$, where N is a finite set of nonterminal symbols, T is a finite set of terminal symbols, $s \in N$ is the start symbol and $\mathcal{P}$ is a set of productions of the form $X_0 ::= X_1 X_2 \dots X_{m-1} X_m$, where $X_0 \in N$, $X_1 X_2 \dots X_{m-1} X_m \in (N \cup T)^+$ and $m \geq 1$.

The language associated to a context-free grammar G is denoted by $\mathcal{L}(G)$ and it is the set of strings $w \in T^*$ which can be derived by the start symbol s applying a finite number of productions in $\mathcal{P}$:

$$\mathcal{L}(G) = \{w \in T^* | \, s \stackrel{*}{\Longrightarrow} w\}$$

where $S \stackrel{*}{\Longrightarrow} w$ means that $s = \beta_0 \Longrightarrow \beta_1 \Longrightarrow \dots \Longrightarrow \beta_{k-1} \Longrightarrow \beta_k = w$, and $\forall i \in [1..k]$, $\beta_{i-1} \Longrightarrow \beta_i$ means that $\beta_{i-1} = uX_0v, \beta_i = uX_1X_2 \dots X_m v$ and $X_0 ::= X_1 X_2 \dots X_{m-1} X_m \in \mathcal{P}$.

Definition 6.4.2 A derivation tree t over a context-free grammar G is a tree labeled with symbols from $(N \cup T)$ s.t.:
i. the root of t is labeled with s;
ii. for every interior node n_0 with children $n_1, \dots, n_m$, there exists a production $X_0 ::= X_1 X_2 \dots X_{m-1} X_m$ such that $\forall i \in [0..m]$, $lab(n_i) = X_i$ (in left-to-right order), where $lab(n_i)$ denotes the label of the node n_i;
iii. every leaf node is labeled with a terminal.

We denote by $t_G(w)$ a derivation tree over G, where $w = lab(f_1) \dots lab(f_h)$ and $\forall j \in [1..h]$, f_j is a leaf node in $t_G(w)$. Obviously $L(G) = \{w \in T^* | \exists t = t_G(w)\}$.

Now, we can give the definition of a *First-order attribute (FOA) grammar*.

Definition 6.4.3 A *FOA grammar* G is a context-free grammar $G = \langle N, T, s, \mathcal{P} \rangle$, enriched with the following elements:

i. *Attribute grammar vocabulary*: it is a tuple $\langle At, S, I, A \rangle$:
- At is a finite set of *attributes*;
- S, I, A are functions from $(N \cup T)$ to the powerset of At;
- $\forall X \in (N \cup T)$, $A(X) = I(X) \cup S(X)$ is the set of attributes, $I(X)$ is the set of inherited attributes, $S(X)$ is the set of synthesized attributes of X;
- $\forall X \in N$, $I(X) \cap S(X) = \emptyset$;
- $\forall X \in T$, $S(X) = I(X) = \emptyset$;
- $I(S) = \emptyset$.

ii. *Evaluation rules*: Denoting $\mathcal{D}(A(X))$ the semantic domain of $A(X)$, then $\forall X_0 ::= X_1 ... X_m \in \mathcal{P}$, it holds:
- $\forall A \in S(X_0)$ there is a function

$r_1 : \mathcal{D}(A(X_1)) \times .. \times \mathcal{D}(A(X_m) \times \mathcal{D}(I(X_0)) \longmapsto \mathcal{D}(S(X_0))$;
- $\forall A \in I(X_k)$ $(k \in [1..m])$ there is a function
$r_2 : \mathcal{D}(A(X_0)) \times \mathcal{D}(S(X_1)) \times .. \times \mathcal{D}(S(X_m)) \longmapsto \mathcal{D}(I(X_k))$;
iii. *First-order logical conditions*: $\forall X_0 ::= X_1...X_m \in \mathcal{P}$, there is a set of first-order logical formulas

$\{\phi \mid \phi$ is defined on $\mathcal{D}(A(X_0)) \times .. \times \mathcal{D}(A(X_m))\}$.

Definition 6.4.4 Given a FOA grammar G, a *derivation tree* $t_G(w)$ over G is *valid* if and only if (i) all the attribute values conform to the evaluation rules *and* (ii) all the logical conditions are true.

Definition 6.4.5 The language associated to a FOA grammar G is $\mathcal{L}(G) = \{w \in T^* | \exists t = t_G(w)$ and $t_G(w)$ is valid$\}$.

FOA grammar specifications define the dependencies among attributes. Such specifications must exhibit certain properties - for instance, dependencies must not be circular. However, there are methods transforming an attribute grammar in such a way such properties hold.

6.5 From FOG automata to FOA grammars

It is not surprising that a conversation specification is closely related to the finite state automata formalism: in fact, a conversation can be viewed as a sequence of message symbols accepted from a finite state automaton. At the same time, a guarded automaton can be mapped into a *regular* attribute grammar[4], and also the vice versa: Proposition 6.5.1 is an example of encoding from guarded automata into (regular) attribute grammars.

Regarding the expressive power on boolean logic of guarded automata - i.e. regular attribute grammars - w.r.t. non-regular attribute grammars, the main results are summarized in [117, 116]: (i) guarded automata can accept the same class of languages as Turing machines; (ii) the terminating guarded automata with finite variable domains accept *regular* languages; (iii) simple guarded automata - with unrestricted variable domains - can accept at least some *context sensitive* languages; (iv) guarded automata, where attributes are set of variables without structure, cannot accept *context-free* languages.

If we take into account first-order logic, it is not so clear how regular attribute grammars and non-regular ones are related in term of expressiveness. Anyway, we could think about first-order constrained conversations which

[4]A regular attribute grammar is an attribute grammar where the unconstrained set of productions is regular, in the sense of Chomsky's language hierarchy.

can be easier specified by non-regular attribute grammars than by regular ones: it suffices to take into account any conversation where the sequence of invocations includes either operation trade-off or memory-based properties. For instance, consider the non-regular attribute grammar G with start symbol s and set of productions $\mathcal{P}$ so defined:

$$\{s ::= m_1 B m_2 \;\; g_1(m_1, m_2); B ::= m_1 m_2 \;\; g_2(m_1, m_2)\}$$

where g_1 and g_2 a first-order logical conditions on m_1, m_2.

G models a conversation where (i) two WSDL operations, we say O_1 and O_2, are defined with input parameters respectively m_1 and m_2, (ii) the sequence of O_1 (invocations) has to precede the one of O_2, (iii) the number of O_1 (invocations) equals the one of O_2, and (iv) for any O_1 and O_2 (invocation), constraints g_1 and g_2 have to be satisfied.

Although guarded automata are suitable to specify the most of existing conversations, attribute grammars look as a good compromise in terms of expressivity, constraint complexity and amount of variables - above all for modeling more complex (i.e. non-regular) conversations. For this reason, the input for our monitoring/validation framework is given in the form of attribute grammar. Another reason behind the choice of specifying constrained conversations in the form of attribute grammars is related to constraint aspects: the scope of a conversation constraint includes both message templates and transitions; it follows that a constraint can be put neither only in a conversation document - since here we could only define conditions to able/disable message transitions - nor in a WSDL document - since here we could only define conditions to well-type messages. Logical conditions in attribute grammars are associated to productions (i.e. transitions) and they involve symbol attributes (i.e message attributes): for this reason, they look suitable to encode conversation constraints.

In the following, we define an attribute grammar associated to a conversation $\mathcal{C} = \langle W_c, W_m, \mathcal{G} \rangle$ by encoding a guarded automaton associated to $\mathcal{C}$ into the attribute grammar formalism.

Definition 6.5.1 Let $\mathbf{W} = \langle \mathcal{A}, W_m \rangle$ be stable. A FOA grammar $G_{\mathbf{W}} = \langle N, T, s, \mathcal{P} \rangle$ associated to $\mathcal{A} = \langle S, M, H, q_0, q_n, \delta, \mathcal{G} \rangle$ is defined as follows:

i. $T = \{\texttt{I(m\_i)} | m_i \in M_i\} \cup \{\texttt{O(m\_o)} | m_o \in M_o\}$;

ii. $N = \{< \mathbf{Q\_j} > | \; q_j \in S\}$;

iii. $s = < \mathbf{Q\_0} >$;

iv. $\mathcal{P} = \mathcal{P}_{rq} \cup \mathcal{P}_{rs}$, where:

$\mathcal{P}_{rq} = \;\; \{< \mathbf{Q\_i} > ::= \; \texttt{I(m\_k)} < \mathbf{Q\_j} > g_{(i,k)}\} \cup$
$\{< \mathbf{Q\_l} > ::= \; \texttt{I(m\_h)} \; g_{(l,h)}\}$ s.t.

$q_i, q_j, q_l \in S, q_j \in Q_i (j \neq n)$, $q_n \in Q_l$, $(q_i, (m_k, g_{(i,k)}), Q_i)$, $(q_l, (m_h, g_{(l,h)}), Q_l) \in \delta$;

$\mathcal{P}_{rs} = \{< \mathbf{Q\_i} > ::= \texttt{O(m\_k)} < \mathbf{Q\_j} > g_{(i,k)}\} \cup$
$\{< \mathbf{Q\_l} > ::= \texttt{O(m\_h)}\ g_{(l,h)}\}$ s.t.

$q_i, q_j, q_l \in S, q_j \in Q_i (j \neq n)$, $q_n \in Q_l$, $(q_i, (\overline{m}_k, g_{(i,k)}), Q_i)$, $(q_l, (\overline{m}_h, g_{(l,h)}), Q_l) \in \delta$;

v. *(Attribute grammar vocabulary)*: it is a tuple $\langle At, S, I, A \rangle$:
- $At = H$;
- For every $< \mathbf{Q\_i} > \in N$, $A(< \mathbf{Q\_i} >) = I(< \mathbf{Q\_i} >) = \langle h_1(i), ..., h_{|M|}(i) \rangle$ (i.e. $S(< \mathbf{Q\_i} >) = \emptyset$);
- $I(< \mathbf{Q\_0} >) = \langle h_1(0), ..., h_{|M|}(0) \rangle$, where $\forall j \in [1...|M|]$, $h_j(0) = \lambda$;

vi. *Evaluation rules*: For every $< \mathbf{Q\_i} > ::= \texttt{I/O(m\_k)} < \mathbf{Q\_j} > g_{(i,k)}$ in $\mathcal{P}$, there is the function r_2 such that $r_2(\langle h_1(i), ..., h_{|M|}(i) \rangle) = \langle h_1(j), ..., h_{|M|}(j) \rangle$;

vii. *(First-order logical conditions)*: Given $< \mathbf{Q\_i} > ::= \texttt{I/O(m\_k)} < \mathbf{Q\_j} > g_{(i,k)}$ (respectively $< \mathbf{Q\_i} > ::= \texttt{I/O(m\_k)}\ g_{(i,k)}$) in $\mathcal{P}$, $g_{(i,k)}$ is the CLiX rule associated to the transition $(q_i, (m_k/\overline{m}_k, g_{(i,k)}), Q_i) \in \delta$ ($(q_i., (m_k/\overline{m}_k, g_{(i,k)}), Q_i) \in \delta$ and $q_n \in Q_i$).

Notice that client's and service's productions belong to the same grammar $G_{\mathbf{W}}$; however, there is no ambiguity among productions corresponding to client's requests ($\mathcal{P}_{rq}$) and the ones corresponding to service's responses ($\mathcal{P}_{rs}$).

6.6 A parser for valid sequences of valid invocation stubs

In the following, we describe in detail the monitoring/validation framework, based on attribute grammars.

1. Let $\mathcal{C} = \langle W_c, W_m, \mathcal{G} \rangle$ be a constrained client-server conversation, where W_m is a WSDL document - being W_m and W_c syntactically related by a *stability* relationship - and $\mathcal{G}$ is a CLiX rule document. $\mathcal{C}$ can be in the form of either an XML-formatted guarded automaton or an XML-formatted attribute grammar. In the first case, a *slave parser* worries to translate it in the form of an associated XML-formatted attribute grammar $G_{\mathbf{W}}$.

2. Given $G_{\mathbf{W}}$, the slave parser links by namespaces respectively nonterminals with WSDL operations, terminals with WSDL operation parameters, CLiX rule arguments with WSDL element attributes. It also

marks grammar productions in order to distinguish those ones corresponding to client's requests and those ones corresponding to service's responses.

3. A programmer freely uses $\mathcal{S}$ to write any client's business logic involving any operation invocation sequence. We denote by $\mathcal{S}$ the stub produced by WSDL2Java and by $\mathcal{J}$ a client business logic on $\mathcal{S}$ $\mathtt{j_0}$; $\mathtt{q\_1.Invoke(m\_i1)}$; $\mathtt{j_1[(m\_o1)]}$; $\mathtt{q\_2.Invoke(m\_i2)}$; $\mathtt{j_2[(m\_o2)]}$; ...; $\mathtt{q\_n.Invoke(m\_in)}$; $\mathtt{j_n[(m\_on)]}$.

   $\mathtt{q\_x.Invoke(m\_ix)}$ denotes the invocation of a method `Invoke` on $\mathtt{q\_x}$ with input instance $\mathtt{m\_ix}$, and its effects consist of (i) marshalling and packing a SOAP message to invoke the corresponding WSDL operation o_x with the corresponding input instance m_{ix} - that associated to $\mathtt{m\_ix}$[5], and (ii) activating a *master parser*. $\mathtt{j_0}$ denotes any Java code which does not involve an invocation, and $\mathtt{j_x[(m\_ox)]}$ denotes any Java code which does not involve an invocation and possibly involves an output/fault instance m_{ox} - that one associated to $\mathtt{m\_ox}$[6] - relative to a previous invoked operation o_x.

4. The master parser inputs $G_{\mathbf{W}}$ and W_m. Since (i) nonterminals and terminals in $G_{\mathbf{W}}$ are linked by namespaces to respectively W_m operations and operation parameters, then any grammar production of the form $< \mathbf{Q\_x} > ::= \mathtt{I(m\_ix)}\ [< \mathbf{Q\_j} >]$[7] $g_{(x,ix)}$ corresponds to a possible $\mathtt{q\_x.Invoke(m\_ix)}$. The master parser looks for deriving in $G_{\mathbf{W}}$ $\mathtt{m\_i1[m\_o1]m\_i2[m\_o2]...m\_in[m\_on]}$[8].

   (Request case) The master parser triggers an error when $\mathtt{q\_x.Invoke(m\_ix)}$ in $\mathcal{J}$ is not correct w.r.t. the conversation protocol, i.e. when it cannot apply a production $< \mathbf{Q\_x} > ::= \mathtt{I(m\_ix)}\ [< \mathbf{Q\_j} >]\ g_{(x,ix)}$.

   (Response case) Analogously, it triggers an error whenever a $\mathtt{j_x(m\_ox)}$ in $\mathcal{J}$ is not correct w.r.t. the conversation protocol, i.e. when it cannot apply a production $< \mathbf{Q\_x} > ::= \mathtt{O(m\_ox)}\ [< \mathbf{Q\_j} >]\ g_{(x,ox)}$. Otherwise, it invokes the validator OpenCLiXML.

[5] A SOAP message contains all the WSDL operation, input - and eventually output/-fault - message names involved in the current invocation.

[6] The symbol $[(\mathtt{m\_ox})]$ denotes zero or one occurrences of $(\mathtt{m\_ox})$ - i.e. respectively $\mathtt{j_x}$ and $\mathtt{j_x(m\_ox)}$. The latter denotes the case when the code involves an output/fault instance associated to $\mathtt{m\_ox}$ and relative to a previous invoked operation o_x, where o_x prototype includes an output/fault parameter m_{ox}. We recall that the correspondence is contained into the received SOAP message.

[7] The symbol $[< \mathbf{Q\_j} >]$ denotes zero or one occurrences of $< \mathbf{Q\_j} >$.

[8] The symbol $[\mathtt{m\_ox}]$ denotes zero or one occurrences of $\mathtt{m\_ox}$.

5. OpenCLiXML checks `m_ix` (respectively `m_ox`) w.r.t. the CLiX rule linked to $g_{(x,ix)}$ (respectively to $g_{(x,ox)}$) and an XML history document H, containing the concatenation of all the input and output/fault instances.

   - If the validation is positive:

   (Request case) the SOAP message generated from `q_x.Invoke(m_ix)` is forwarded to the network (i.e. the invocation is fired) and the input instance contained in it is appended in H.

   (Response case) the output instance is extracted from the received SOAP message, it is appended in H, it is unmarshalled and the corresponding object is assigned to `m_ox`.

   - In the opposite case: the master parser catches the OpenCLiXML exception and it returns informations about the error. In the *(Request case)* the invocation is not fired.

The above procedure allows the programmer: (i) to automatically generate a message sequence `m_i1[m_o1]m_i2[m_o2]...m_in[m_on]` which belongs to the conversation protocol associated to the service; (ii) to forward an invocation message only after a validation step (items 5-6) - hence without waiting for service acks, error messages and other additional informations; (iii) to precisely detect which requests(/responses) have triggered an error.

Fig.6.3 shows the monitoring/validation framework in terms of relationships among components. Labels in *boxes* denote documents, those ones in *ovals* denote processes and those ones in *diamonds* denote exceptions.

6.7 A simple authentication service

Suppose to design a simple authentication service as follows: (i) the client is required either to register by a *Registration* form, or to login by a *Login* form; (ii) after filling a *Registration* form, the client can only access to a *Login* one; (iii) after filling a *Login* form, the client is allowed to enter the system only if either it has already registered *in a past session* and login username is valid, or he has just filled a *Registration* form *in the current session* and login username is valid; (iv) the allowed max number of failed logins is 3. In terms of WSDL document, we could define a `Login` operation, with `LoginRQ` as input parameter, `ValidLoginRS` and `InvalidLoginRS` respectively as output and fault parameters, and a `Registration` operation, with `RegistrationRQ` and `RegistrationRS` respectively as input and output parameters.

A FOA grammar $G_{\mathbf{W}}$ is defined in the following.

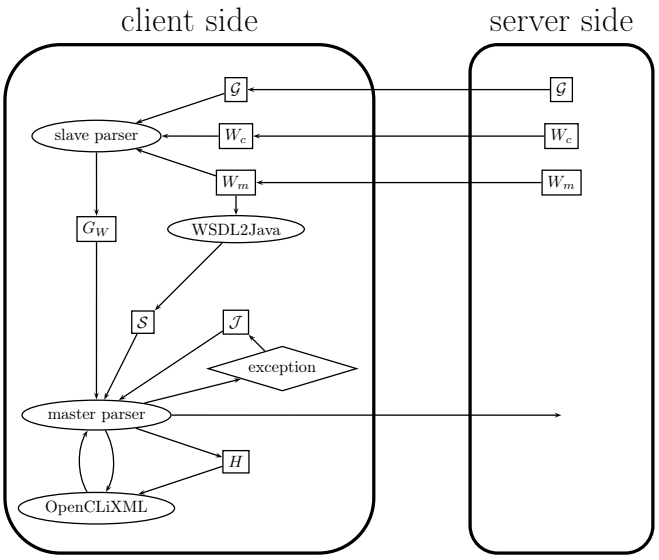

Figure 6.3: The monitoring/validation framework.

$$< \mathbf{Q\_0} > ::= \quad < \mathbf{Q\_1Registration} > \Big| < \mathbf{Q\_2Login} >$$
$$< \mathbf{Q\_1Registration} > ::= \mathtt{I(m\_1)} < \mathbf{Q\_3OutRegistration} >$$
$$< \mathbf{Q\_2Login} > ::= \mathtt{I(m\_2)} < \mathbf{Q\_4OutLogin} > \mathtt{rule1} \wedge \mathtt{rule2}$$
$$< \mathbf{Q\_3OutRegistration} > ::= \quad \mathtt{O(m\_3)} < \mathbf{Q\_2Login} >$$
$$< \mathbf{Q\_4OutLogin} > ::= \quad \mathtt{O(m\_4)} < \mathbf{Q\_1Registration} > \Big|$$
$$\mathtt{O(m\_4)} < \mathbf{Q\_2Login} > \Big| \mathtt{O(m\_5)}$$

$\mathtt{m}_1 = \mathtt{RegistrationRQ}$ $\quad \mathtt{m}_4 = \mathtt{InvalidLoginRS}$
$\mathtt{m}_2 = \mathtt{LoginRQ}$ $\quad \mathtt{m}_5 = \mathtt{ValidLoginRS}$
$\mathtt{m}_3 = \mathtt{RegistrationRS}$

```
rule1 :
context LoginRQ inv :
(self.RegistrationRS− > notEmpty() &&
self.RegistrationRS.RegistrationRQ− > notEmpty()) implies
self.RegistrationRS.RegistrationRQ.login = self.username

rule2 :
context LoginRQ inv :
LoginRQ.allInstances− > count(InvalidLoginRS) <= 3
```

For simplicity, the grammar is not in XML format and constraints are written as OCL formulas[9]. At this aim, Fig.8.3 shows the authentication

[9]It is well-known that the first-order logical fragment of OCL can be encoded into

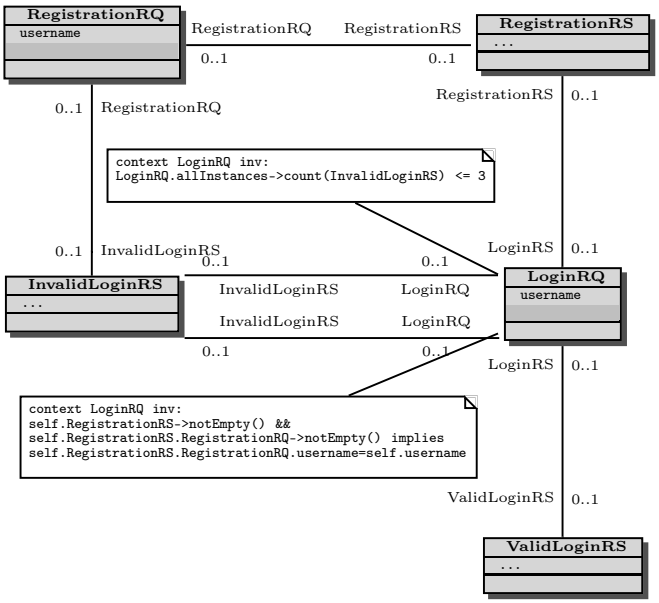

Figure 6.4: The authentication protocol Class Diagram.

protocol above described in term of Class Diagram. Each class models an operation parameter with its WSDL attributes; analogously, each OCL formula translates a constraint. `LoginRQ.allInstances` denotes the set of `LoginRQ` instances, while `LoginRQ.allInstances->count(InvalidLoginRS)` denotes the number of `LoginRQ` instances associated to `InvalidLoginRS` ones.

CLiX.

Part V

Implementation

Chapter 7

Resourceome Knowledge Management

Information systems aim to give support to the individual user for information access, retrieval and store. In order to support knowledge workers during their tasks of information search, location, and manipulation, a system must provide information suitable for a particular user needs, and it must be able to facilitate the information sharing and reusage. The *Knowledge Management* and its associated tools aim to provide an environment where people may create, learn, share, use and reuse knowledge, for the benefit of the organization, the people who works in it, and the organization's customers.

In this chapter, we describe how the *Resourceome* multi-level model can provide the above mentioned features, as well as can capture and share the procedural knowledge. Another aspect of *Resourceome* model is that it can be used for the collaborative ontology development.

7.1 *Resourceome* Knowledge Lifecycle

The *Resourceome* model allows domain-experts to add and share both declarative and procedural proper knowledge in every kind of domain. Now we will describe the *Resourceome* lifecycle and how it can be used to make procedural knowledge explicit. Making procedural knowledge explicit can be done using the business process modeling (BPM) approach. In the literature there is no uniform point of view on the BPM lifecycle number of phases. It varies depending on the chosen granularity to identify the phases. Basically, there are the following phases: process modeling, process implementation, process execution, and process analysis. Usually business analysts or business man-

agers create process models and analyze process models from the business point of view while IT engineers are involved in process implementation and execution phases.

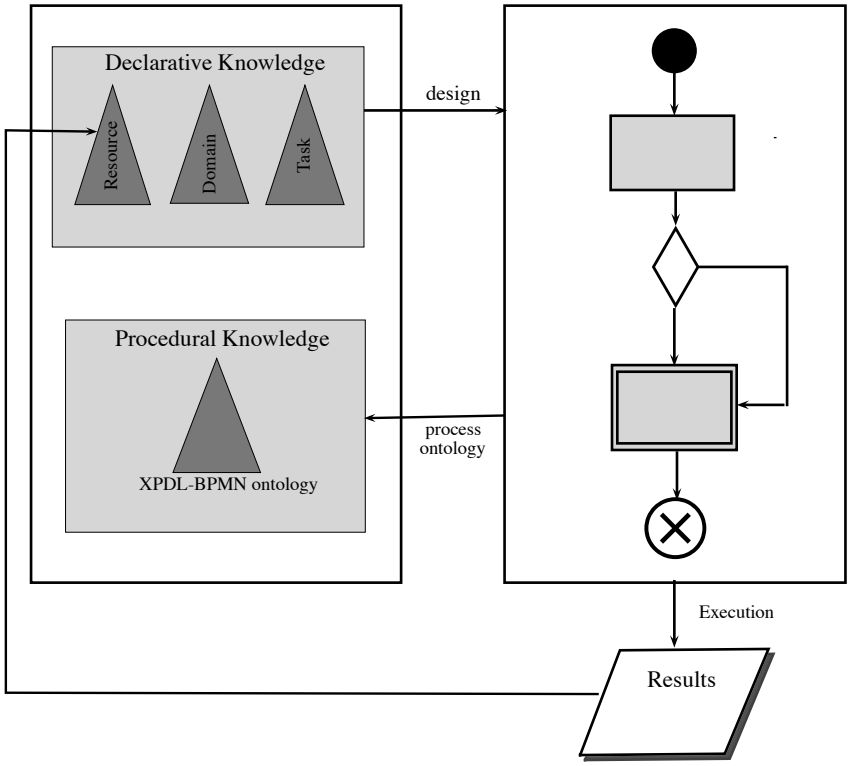

Figure 7.1: $\mathcal{R}esourceome$ Knowledge Lifecycle Idea.

A business process consists of a set of activities that are executed in some organization according to some rules in order to achieve certain goals. It is now well-accepted that there are different aspects of the business processes that can be modeled and investigated independently. The three basic aspects are:

Behavioural: The behavioural aspect defines the order in which the tasks of a business process are instantiated and in which the corresponding activities are executed. Note that the order of the execution does not need to be sequential, it can be a partial order representing the dependencies among the activities.

Organizational: Describes the organization structure and, in particular, the resources and agents.

Informational: All information involved in a business process can be considered to be documents or resource[1], where a document is an artifact representing some piece of information.

As previously introduced, the $\mathcal{R}esourceome$ model can specify the organizational and informational aspect of a business process in some context or domain since:

- it allows us to define activities in the Task Ontology, as well as the relations between them and the relations that describe the resources that are needed by the activities themselves;
- it allows business process behaviour to be specified independently from a specific formalism.

Moreover, $\mathcal{R}esourceome$ makes possible to specify procedural knowledge. It is well-known that a workflow is a partial or total automation of a process, in which a collection of activities must be executed according to certain procedural rules. The main advantages of automated workflows relate to effectiveness, reproducibility, reusability of procedures and of intermediate results and traceability. Sometimes, domain experts are able to define the proper workflow and they would like to share and execute the proper workflow on the fly without involving IT engineers, even if the program could give incorrect output results. This kind of approach allows to specify procedural workflow by domain experts driven by semantics. This specified workflow represents the base knowledge of procedural knowledge, while the running workflow results can enrich the $\mathcal{R}esourceome$ base knowledge adding new resources (individuals) (Fig. 7.1).

7.2 Models and Meta-models

In this section, we discuss the concepts of models, meta-models and (workflow) instances. We focus on the concepts of workflow models, typically based on control-flow captured in procedural process notations.

A model is an abstraction of the real world. In this case, we can use different formalism to define behavioural aspect of workflow, for instances the Petri Nets and Workflow Nets. In order to use a formalism, we need to define its concepts and notation. This can be done by providing a meta-model

[1]In some systems, information is stored in a relational database. In those cases, we can consider the tuples in the database as documents. Other information can be stored or captured by the Web Service meaning.

for the formalism, thus the meta-model defines the language used to express the model. OWL can be used to define this meta-model. In $\mathcal{R}esourceome$ we use XPDL ontology as meta-model and a Web based workflow editor tool based on WF-nets as a model. The instance of the workflow, a XPDL ontology instance represents the procedural base knowledge. XPDL can be also executed by a workflow engine.

7.2.1 XPDL ontology

XPDL has been introduced to allow process model exchange between information systems, most of which are based on proprietary workflow models. A detailed XPDL-ontology description can be found in the appendix B(pag. 131). Now we will focus on some advanced features of ontologized XPDL.

XPDL-ontology represents the informal underlying semantics of XPDL in a formal and expressive Web ontology language. Adding semantics to XPDL using OWL, allows reasoning and sharing of the processes definition. The ontologized process XPDL model can be interlinked and enriched with business knowledge in existing background ontologies, in our case $\mathcal{R}esourceome$. Another feature is the interchanging resource in the workflow execution whenever some resources are not available. In this scenario the workflow engine can use an equivalent resource involved in the activity if the equivalent resource is defined in the ontology. The basic aspect defined above, *mbehavioural*, *rganisational* and *informational* can be mapped in XPDL ontology as follows:

Behavioural: (Control Aspect) *Process type* constitutes the primary modelling element. It groups related activities, data, and resources together. *Activities* in XPDL represents the reusable task behaviour in a process and takes one of the following types, a triggered event, a route activity that constrains the ordering of activities or a block activity;

Organisational: The organisational aspect in an XPDL document is only weakly defined; a process participant is simply a token of the following types: resource set, resource, organizational unit, role, human, or system. Anyway resources, as well as the other types, can be linked to the $\mathcal{R}esourceome$ axiomatized formal ontology, providing in this way formal semantics.

Informational: As the organizational aspect, we provides semantics using $\mathcal{R}esourceome$.

7.2.2 $\mathcal{R}esourceome$ KMS architecture

The main actors, which are involved in $\mathcal{R}esourceome$ KMS, are:

- the $\mathcal{R}esourceome$ knowledge model as a registry of available resources and activities;
- a Web-based graphical interface for composing, entering data, watching execution, displaying results;
- an archive to store and share workflow descriptions, results of executions and related traces;
- a set of programming interfaces able to dialogue with remote activities;
- a set of visualization capabilities for displaying different types of results.

A workflow specification is translated into a workflow engine by means of a compilation process involving three main components:

The graphical interface: it enables the definition of workflow as (primitive and complex) activity in the XML Process Definition Language (XPDL) [194] and XPDL ontology, as well as the execution of existing or previous saved workflows, the monitoring of their execution state and the management of the produced results.

An XPDL compiler: it is an $\mathcal{H}ermes$ special component which translates workflows specifications into interactive component-based specifications and generates the code to be executed on $\mathcal{H}ermes$ middleware (see below). The associated workflow specification is the coordination model that describes how the generated agents cooperate to reach a particular goal.

$\mathcal{H}ermes$ middleware: it provides the run-time environment for executing of workflows as mobile and distributed code. In particular, it enables, transparently to users, the interaction with the external resources, i.e. invoked applications, and the migration of workflow executors to different sites.

$\mathcal{R}esourceome$ tool is a software 3-layer software architecture as shown in Fig. 7.2: *user layer*, *system layer* and *runtime-layer*.

- At the user layer, it allow managing both declarative and procedural knowledge by means of adding new instance resource and and specify their application as a workflow of activities using the graphical notation.

- At the system layer, it provides a context-aware compiler to generate a pool of user mobile agents from the workflow specification.
- At the run-time layer, it supports the activation of a set of specialized service agents, and it provides all necessary components to support agent mobility and communication.

The main difference between the run-time layer and the system layer is how agents function in each. ServiceAgents in the run-time layer are localized to one platform to interface with the local execution environment. UserAgents in the system layer are workflow executors, created for a specific goal that, in theory, can be reached in a finite time by interacting with other agents. Afterwards that agent dies. Furthermore, for security UserAgents can access a local resource only by interacting with ServiceAgent that is the "guard" of the resource.

It follows a detailed description of the main components and functionalities of each layer.

User Layer

The user layer is based on both ontology and workflow technologies. While the $\mathcal{R}esourceome$ knowledge management system, using a graphical interface, provides the knowledge based of the application domain, the $\mathcal{R}esourceome$ workflow management system provides to users a set of programs for interacting with the workflow management system. In particular, there are two main groups of programs: programs for specifying, managing and reusing existing workflow specifications, and programs enabling administration and direct interaction with the workflow management system. The workflow editor is the program that supports the workflows specification by composing activities in a graphical environment. Activities used in a workflow are configured by specifying input parameters and their effects are recognizable as modification of state variables or modification on the environment's status.

The workflow editor enables the composition of both primitive and complex activities. A primitive activity is an activity that can be directly executed. Users can specify primitive activity without knowing the real implementation. A complex activity is an activity that must be specified before it can be used.

By using complex activities the specification of workflows is simplified because they enhance both hierarchical specification and reuse: we can use an already existing complex activity without caring of its specification.

Users can use complex activities and stored workflows to increase productivity when specifying new workflows. Moreover, large libraries of both

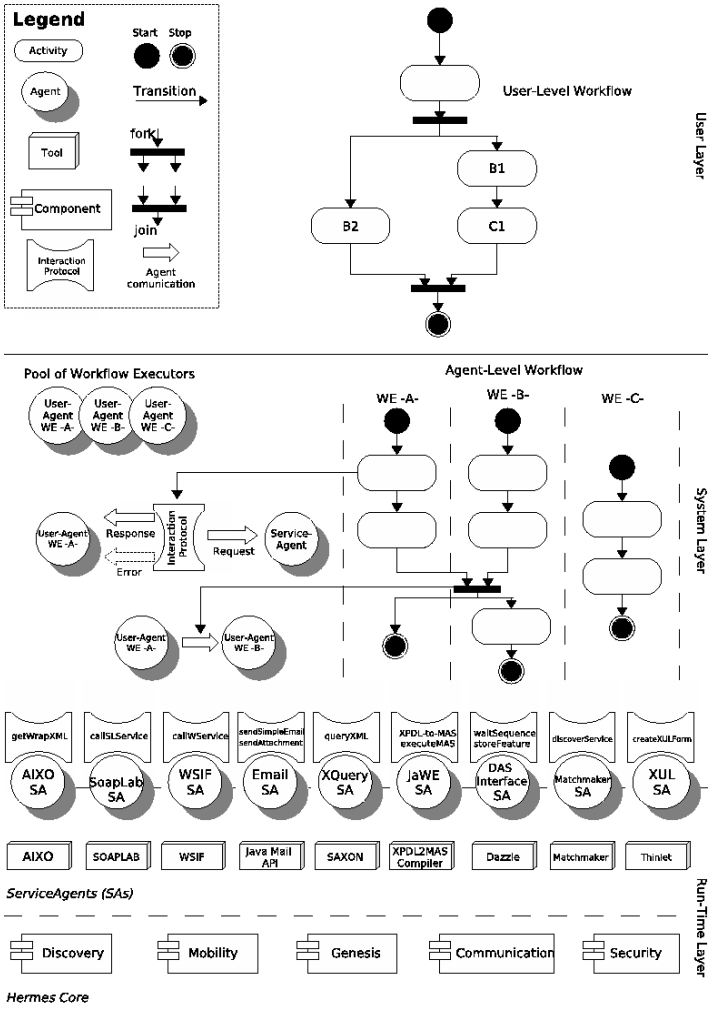

Figure 7.2: $\mathcal{H}ermes$ Software Architecture

domain specific primitives and complex activities can be loaded to specialize the editor for a specific application domain.

System Layer

System Layer provides the needed environment to map a user-level workflow into a set of primitive activities. The execution of such activities is coordinated by a suitable model: it implements them at user level by autonomous software entities UserAgent, able to react to the environment changes where they are executed; moreover, it hides any implementation details at the execution environment. Finally, a compiler generates a pool of user mobile

agents from the workflow specification.

Run-time Layer

Run-time Layer, at the bottom of the architecture, provides primitives and services which are essential for agent mobility and resource access. The kernel is the $\mathcal{H}ermes$ platform for mobile computing, which provides primitives for discovery, mobility, communication, and security. As already described, the overall structure of the system is very complex, it supports abstract specifications that are mapped into a complex distributed and coordinated flows of activities over a large-scale distributed system.

7.2.3 XPDLCompiler

XPDLCompiler receives an XPDL document and generates the Java bytecode implementing Interactive Components. A lexical and syntax analyzer performs the validation and the parsing of the XPDL document using the Java Architecture XML Binding [58]. After this first phase, the compiler checks if the activities used in the workflow specification have a corresponding implementation in UAIR (User Implementation Activity Repository). Each role is translated in an Agent skeleton, an extension of $\mathcal{H}ermes$ UserAgent Java class. As Figure 7.3 shows, a UserAgent provides the needed communication methods to interact with other UserAgents. Then, for each activity, the corresponding implementation code in UAIR is plugged into an Agent skeleton and each internal scheduler is set. The Java code generation is performed using Apache Velocity (http://jakarta.apache.org/velocity/) template engine. Finally, using the Java compiler, the generated bytecode can be loaded into $\mathcal{H}ermes$ middleware.

In order to handle this complexity and to support the reusability of existing artifact during the development of a middleware system for a specific application domain, we designed $\mathcal{H}ermes$ kernel following a component-based [33] approach.

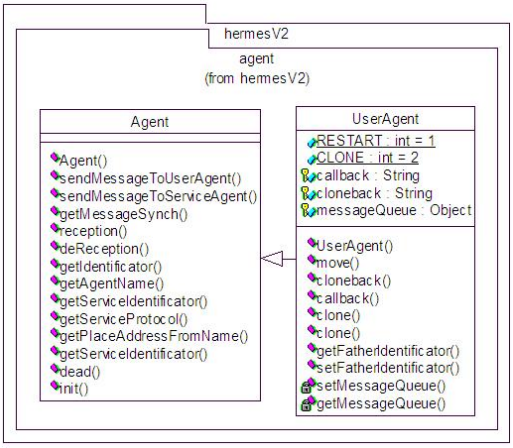

Figure 7.3: UserAgent and Agent main methods

Part VI

Biology Application Domain

Chapter 8

Application Domain

8.1 Bioinformatics Resources

We introduce the reader to the universe of bioinformatics, concentrating our attention on bioinformatics resources, like e.g. databases, articles and programs.

8.1.1 A Universe in Expansion

Bioinformatics can be seen as a very dynamic and turbulent universe in continuous expansion. Its expansion speed is increasing and new regions are suddenly created. Bursts of new topics [167] can be often observed as well as the deadening of other ones. Buzzwords quickly enter the scene [163]. Metaphors are frequently used in teaching and research as well as in divulgation of science. One of the most famous metaphors in life science paints the genome as the "book of life" [174].

Bioinformatics originates from the encounter of the two universes of computer science and life sciences. The growing amount of knowledge acquired in molecular biology, coupled with the technological evolutions (constant improvements and miniaturizations; the first impacts from the nanotechnologies) initiated to produce unimaginable quantity of data. Therefore the marriage with a science able to organize, manage, elaborate and represent such heaps of information was an unavoidable one. On the other hand, also the (much more recent than biology) science of informatics is strongly characterized by constant technological advancement and exponential growth. The empirical Moore's law[1] (the number of transistors on a chip doubles about

[1] http://www.intel.com/technology/mooreslaw/

every two years) can naturally be transferred from semiconductor industry to many other fields (e.g. growth of DNA sequence database, number of internet Web sites). The whole system evolves as a positive feedback circuit. From data, new information and knowledge are derived, in turn catalyzing new technologies which consequently yield higher amount of data. Scientists working in this area are becoming used to data deluges and overflows [131]. An impressing logarithmic graph at GenomeNet[2] reports the growth of the main molecular databases. Concerning DNA sequences the new generation of sequencers has hundreds times higher production rates with a cost per Megabase of a tenth or a hundredth in comparison to the previous generation. So we can expect a further hike in the production of DNA sequences and in the number of sequenced organisms. In general it is also increasing the variety of the data. The anarchy in microarray experiments at the beginning of the new millennium required the definition of standards (i.e. MIAME [134]). Similarly, to bring order in the - now maybe too abused - field of systems biology, standards have been defined for data in metabolic modeling (MIRIAM [166]), proteomics experiments (MIAPE [186]), molecular interaction experiments (MIMIx [173]).

The pioneering computational analysis of molecular data and information started even before the 1970s [175]. The term "bioinformatics" was first used by Hwa Lim in the late 1980s [177]. But the community, after all these years is still asking "what is bioinformatics?" [150] and "who are the bioinformaticians?". We suggest to identify three different levels of bioinformaticians with respect to their relations to resources.

At the basic level we can collocate the simple users of bioinformatics resources. In the pre-WWW age, resources were databases (composed of a few records) and programs, to be installed in local computers. The WWW contributed undoubtedly to the affirmation of bioinformatics. The fact that every data could be freely accessed on a database in the internet from everybody, from everywhere in the world and at every time, revolutioned science. HTML interfaces rendered simpler the execution - online - of algorithms on generally remote, huge and complex data. It was not necessary anymore to bother about programs and databases installation, license, versioning, operating systems, hardware requirements and system management skills. Nowadays, for every molecular biologists and every scientist involved in "omics" disciplines, bioinformatics simply offers the common daily tools. The suffix "omics" became very famous up from the late 1990s to indicate the paradigm shift of research: from the study of a single molecule (e.g. a gene, a protein) to the study of the entire population of such molecules as a whole (e.g. the

[2]http://www.genome.ad.jp/en/db_growth.html

genome, the proteome). Probably with the announcement of the completion of the first draft of the human genome in the year 2000, bioinformatics reached the status of "big science", becoming popular in the media and to the common people.

At the second level we can find the users which assemble resources. In the decade around the year 2000, complex repetitive bioinformatics analyses were automatized in form of programs written with script languages (e.g. Perl). Besides being very compact and permitting rapid programs development, such languages are mostly ideal also for performing sophisticated text analysis and pattern matching. Some of them are even endowed with dedicated bioinformatics libraries (e.g. Bioperl[3]). Now, most of the bioinformatics programs as well as database queries, can be invoked remotely as Web Services available somewhere in the internet. Web pages with HTML form have been developed for human beings, but being not standards and without any precise semantics and syntax, they could not be tackled automatically by programs. Also the affirmation of XML[4] as a standard for data interchange contributed firmly to acquire a complete machine interoperability. Remote programs sparse over the internet can talk - and understand - with each other using standard protocols. Scripts to execute "in-silico" bioinformatics experiments have been now substituted by workflows of Web Services, which can be composed in a very simple visual manner. Therefore, also users without any programming skill can easily assemble them.

At the higher level we pose the developers of resources. Either they could be the computational biologists, which develop the algorithms, subsequently implemented as stand alone programs or Web applications, accessible through HTML forms or Web Services. Or they could be the database developers, which expose their data or information, probably obtained also through complex computations and filtering. Here still emerges the foundational binomial of bioinformatics *bioinformatics = database + computational biology*. The latter could in some way remember the notable *algorithms + data structure = programs* [189].

To give an idea of the dynamicity of bioinformatics we do list the titles of the articles in the "trends guide to bioinformatics" edited by Mark Boguski in 1998 [133]: Text-based database searching, Fundamentals of database searching, Practical database searching, Computational genefinding, Multiple-alignment and -sequence searches, Protein classification and functional assignment, Phylogenetic analysis and comparative genomics, Functional genomics, Databases of biological information. We can make a com-

[3]http://www.bioperl.org/
[4]http://www.w3.org/XML/

parison with the categories adopted from 2005 - only seven years later - by the journal Bioinformatics for classifying its articles: Genome analysis, Sequence analysis, Structural bioinformatics, Gene expression, Genetics and population analysis, Data and text mining, Databases and ontologies, Systems biology. Gene Expression, with its inflation of microarray experiments - but now already in a descending phase - could maybe have been glimpsed in 1998's Functional genomics. Text mining, ontologies and systems biology arose instead rather "unexpectedly" and today constitute probably some of the most active research area in bioinformatics. A good introductions to biological literature mining can be found in [163]. The "manifesto" of computational systems biology was launched by Kitano at the dawn of the new millennium [164, 165]. Evolving research trends in bioinformatics have been analyzed in 2006 also by Wren and collaborators [176], through a statistical analysis of the occurence of MeSH[5] terms in the Bioinformatics Journal abstracts. Unfortunately MeSH terms concern biomedicine and do not actually provide a proper classification of bioinformatics tasks.

Now it has become extremely hard for a scientist to be a "plenary" bioinformatician, following the development of the whole area. The galaxies of the bioinformatics universe are too many, too big and too distant among each other. Being aware of all the resources available in the whole domain is a "mission impossible". For this reason, adopting the same "ome" suffix used in indicating the wide amount of a molecular population, we termed "$\mathcal{R}esourceome$" the full set of bioinformatics resources. In this chapter we describe how an intuitive semantic index of bioinformatics resources could be built for being understood (and reasoned) also by computers.

A Resource Ontology

It is not easy to build a universally accepted classification schema for bioinformatics resources, as the assignment of semantics to the concepts of actor and artifact is not a trivial task. A shared ontology is needed to properly annotate resources and for this reason here we give an example of how this task can be done. We describe a Resource Ontology that formalizes the basic entities with which scientists interact every day. This essential Resource Ontology - by definition built on a *is-a* taxonomy - allows to classify bioinformatis resources.

Besides expressing particular concepts identifying specific kinds of resources, we introduced also non-hierarchical semantic relationships, to improve automatic inference processes and to understand the dependences be-

[5] http://www.nlm.nih.gov/mesh/

tween different resources.

The first step consists in identifying the main classes of bioinformatics resources that we want to represent. Some domain independent methodologies and directions to properly design high quality and consistent biomedical ontologies can be found in [127], [159] and [192].

In a Resource Ontology, a generic *Resource* can be represented by two general kinds of concepts: *Actor* and *Artifact* (see Figure 8.1).

Intuitively *Actor* represents resources where humans have a protagonist role, e.g. *Person*, with sub-concept *Author*, and *Institution*, which in turn includes *Research Group*, *Firm*, *Organization*, *University*, *Institute*, *Department*, *Division*.

As *Artifact* we mean concepts like *Event*, *Educational Artifact*, *Project*, *Informatics Artifact*, *Methodology*, *Literature Resource*.

In particular as *Event* we consider *Seminar*, *Meeting* and *Conference*. *Educational Artifact* resources include *Tutorial*, *Course* and *Learning Object* (for e-learning).

Literature Resource considers *Book*, *Conference Proceeding*, *Article* and *Journal*. Besides literature resource, another important class is that of *Informatics Artifact*. This concept includes *Artifact for Data Representation*, *Artifact for Knowledge Representation* and *Computational Resource*.

Under *Artifact for Data Representation* we have *File* and *Database* which undoubtedly have an historical place in bioinformatics. *Artifact for Knowledge Representation* includes the *Classification System*, such as *Ontology*, and *Semantic Web Prototype*.

Another very important branch is *Computational Resource*, which includes *Algorithm* and *Program*. The latter can be specialized into *Stand Alone* and *Web Based* programs, which includes the *Web Service* class of programs.

We felt the necessity to consider also some auxiliary concepts. In particular they permit to take into account data features like format (an instance could be e.g. "FASTA format") and type (e.g. "DNA sequence" or "integer"). To maintain the ontology as generic and reusable as possible, we hypothesized a domain-specific *Domain Data Type* class. Features like data format and type are becoming essential in bioinformatics data and processes integration. For example, in a bioinformatics workflow management system, it would be very helpful to have formally defined the input and output data type of programs and web services. This will allow to correctly compose and enact them or to suggest to the user the available tools for a specific data type.

In the Resource Ontology some attributes should be assigned to the root concept, in order to give more specific information on the instances belonging

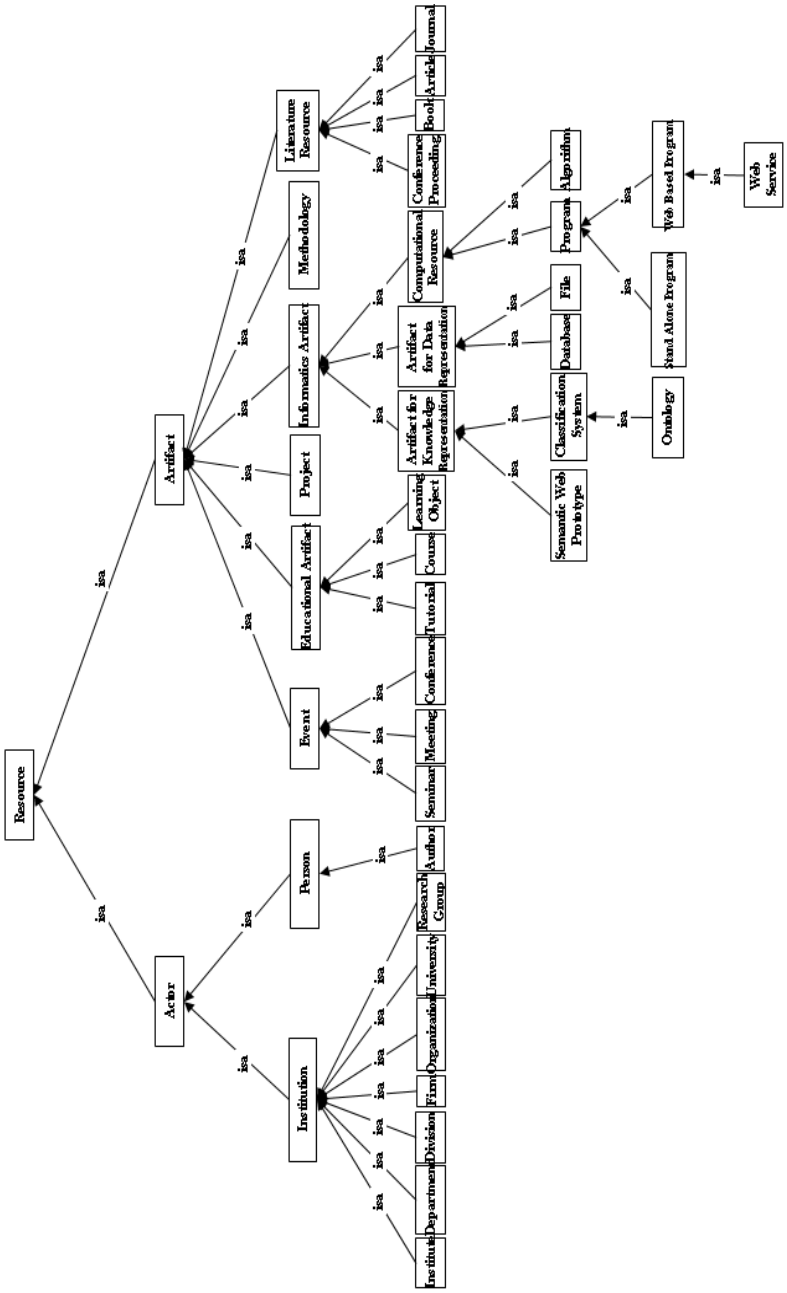

Figure 8.1: The is-a hierarchy of the resource ontology

to its sub-concepts. Some examples can be:

- name: the name of the instance. This is a functional attribute, i.e. it is possible to have only one name for each instance;

- description: a string allowing to give a precise description of the instance. It is possible to have just one description for each instance;
- author: the creator of the instance in the $\mathcal{R}esourceome$;
- creation date: the date of creation of the instance. Clearly it is possible to have only one creation date for each instance;
- URI: the URI of the instance. If it is an URL than the resource can be accessed directly on the Web.

Besides these general attributes, sometimes there is the need of defining other attributes for the description of a particular resource, e.g. abstract (which must be functional) for an *Article* resource, or license for an *Informatics Artifact*.

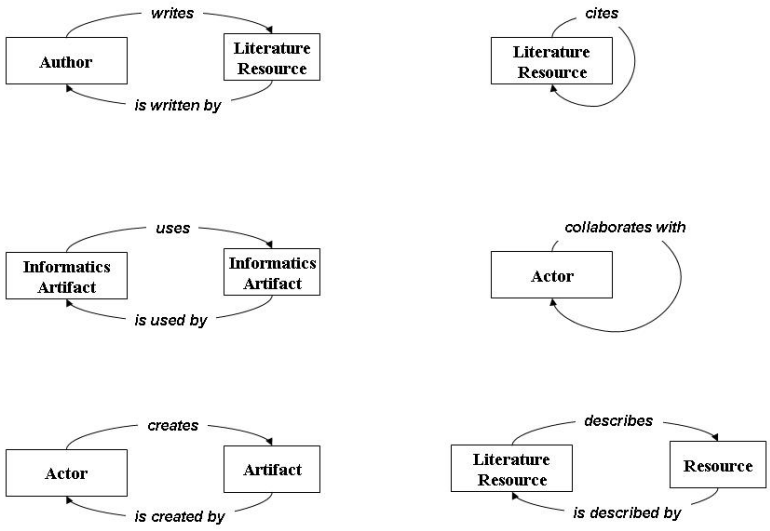

Figure 8.2: Some examples of non-hierarchical relationships in the Resource Ontology

We also suggest some useful non-hierarchical relationships for the Resource Ontology (see fig. 8.2): *Informatics Artifact* `uses` *Informatics Artifact*; *Actor* `collaborates with` *Actor* and `creates` *Artifact*; *Author* `writes` *Literature Resource*.

One of the most important relations is *Literature Resource* `cites` *Literature Resource*. Usually, in the bioinformatics domain *Literature Resource* `describes` *Resource*.

For most of the relations just described, it is possible to define inverse ones, e.g. *Literature Resource* `is cited by` *Literature Resource* or *Literature Resource* `is written by` *Author*. The `collaborates with` relation is symmetric.

8.2 Biology Domain

The biological domain and operational universe recalls in some way a global and distributed data structure and algorithm space, on which in-silico experiments could be thought as transparent and automatic computational procedures, i.e. independent from semantics, heterogeneity and distribution of the involved resources and tools. It follows that the bioinformatic metaphor, which aims to identify the whole biological space to a virtual biological desktop, implies in-silico experiments to be able to automatically integrate (and access to) heterogeneous and distributed databases and computational tools.

A concrete and functional implementation of this vision should necessarily resort on (i) a lower semantic labeling mechanism of all existing resources-data and applications-algorithms, (ii) an upper semantic-driven mechanism for building workflows on this labeled biological space, and (iii) an intelligent technology which allows mobile, distributed and well-balanced computations to be supported.

8.3 $\mathcal{R}esourceome$ for semantic-driven in-silico experiments

In-silico experiments are described as processes of activities - laboriously executed in a large, distributed and dynamic environment - to test hypotheses, derive a summary and search for patterns. As a consequence, they are naturally specified as workflows of activities that implement data and control analysis processes in standardized but dynamic environments.

It is well-known that a workflow is a partial or total automation of a process, in which a collection of activities must be executed according to certain procedural rules. The main advantages of automated workflows relate to effectiveness, reproducibility, reusability of procedures and of intermediate results and traceability. which are also fundamental features to validate a scientific experiment. These are the main reasons why the $\mathcal{R}esourceome$

environment for assembling, managing and executing in-silico experiments recalls - in its structure and in some functionalities - a Workflow Management System (hence the acronym WMS).

Similarly to [196], $\mathcal{R}esourceome$ WMS permits to define any in-silico experiment specification as an activity workflow and to translate it into mobile code supported by Hermes middleware[6]. However, differently from [196] and other traditional WMSs, the $\mathcal{R}esourceome$ knowledge model allows each workflow activity to be automatically linked to its context, roles, objects and documents, i.e. it enables a fully semantic-driven mechanism for realizing in-silico experiments on a biological domain. As a consequence, $\mathcal{R}esourceome$ WMS can naturally capture not only the experimental method behind an in-silico experiment, but also the associated constraints and goals.

In particular, $\mathcal{R}esourceome$ WMS provides two interfaces - user and administrator - with different privileges about definition, selection and execution of experiments related to specific goals. In the first case, simple users can only select experiments from a finite list of goals, eventually specify goal parameters driven by $\mathcal{R}esourceome$ KMS interface (i.e. navigating in a control way on Domain and Resource Ontologies, as well as adding new OWL-DL resource individuals and relationships and new SKOS concepts respectively on the Resource and Domain Ontology application level) and visualize the obtained results. In the latter case, the administrator can edit experiments, selecting the appropriate and involved domain, resource and activity concepts (and eventually accessing to any knowledge model level), associate specific goals and store them.

8.4 An ontology-driven design of an in-silico experiment

In this section we provide an example of a process data retrieval in $\mathcal{R}esourceome$. The goal is to edit a workflow of activities in order to obtain and to visualize all crystallographic structures related to a swissprot protein identificator. In order to accomplish the goal of the proposed workflow, we need to infer all the activities that belong to the class *Database_Retrieval* concerning the concept *Protein*.

Fig. 8.3 shows the semantic search in $\mathcal{R}esourceome$. The syntax used is a triple *Concept*, *Relation*, *Target*. In our case we have *Concept=Database_Retrieval*, *Relation=corcern* and *Target=Protein*. Using the conceptual map as shown

[6]Due to the lack of space, middleware architecture is not discussed here and we refer to [202] for further details.

in Fig. 8.4 is possible to get more informations about the activities found as the input needed and the produced output.

After the user has chosen the proper activity, can drag and drop it directly in an activity element of the workflow editor as the Fig. 8.5 shows.

The activities used in a workflow can be configured with several parameters, in this way it is possible to reuse the same activity for different workflows. In this case the chosen activity takes in input a protein identificator. A user can specify the order of activities execution using special control-flow patterns.

The *Sequence* pattern allows to execute an activity after another. It is generally used when the output of an activity must be piped as input of the subsequent. In the workflow example in Fig. 8.6, the SWISSPROT entry file obtained by *Activity_0* is piped as input of the *Activity_1* that extracts the cross-references of the SWISSPROT entry to the other database. The *If* pattern defines a conditional routing where the choice of the activity to be executed is case-driven. An error or exception can be caught and considered as special case, so that the workflow becomes fault tolerant and the execution can select an alternative path when something goes wrong.

The *Iteration* is a pattern enabling the cyclic execution of one or more activities. When a special case occurs the control-flow leaves the cycle and the workflow execution continues. The arrow on the left hand side of the graphical notation addresses the direction of the execution. In the proposed workflow *Iteration_0* control-flow allows to evaluate each the cross-reference *Activity_2*, to choose through *If_0* control-flow those that refer to crystallographic structures, to fetch (*Activity_3*) them from the protein data bank and to store (*Activity_4*) as results.

Even if is not used in our example another possible control-flow is the *Concurrence* pattern that enables the parallel execution of two or more activities. It is possible to assign each of these concurrent activities to different performers corresponding to the Workflow Executors. In this case, the code mobility, supported by Hermes middleware [202], can be exploited to improve the computational load balancing of a workflow.

As Fig. 8.7 shows, during the execution of the workflow, it is possible to monitor the obtained results and/or to interact with it whenever the in-silico experiment requires a conditional input from the user at run-time. Resourceome provides also several plugins to visualize in a proper way the results. An example is the integration of the Jmol applet that, as Fig. 8.8 illustrates, is used in this case for the 3D visualization of the crystallographic structures fetched. Once the workflow is defined and tested by a bioinformatic user, through $\mathcal{R}esourceome$ is possible to publish it and make it available to a biologist that can use it without explicitly knowing its imple-

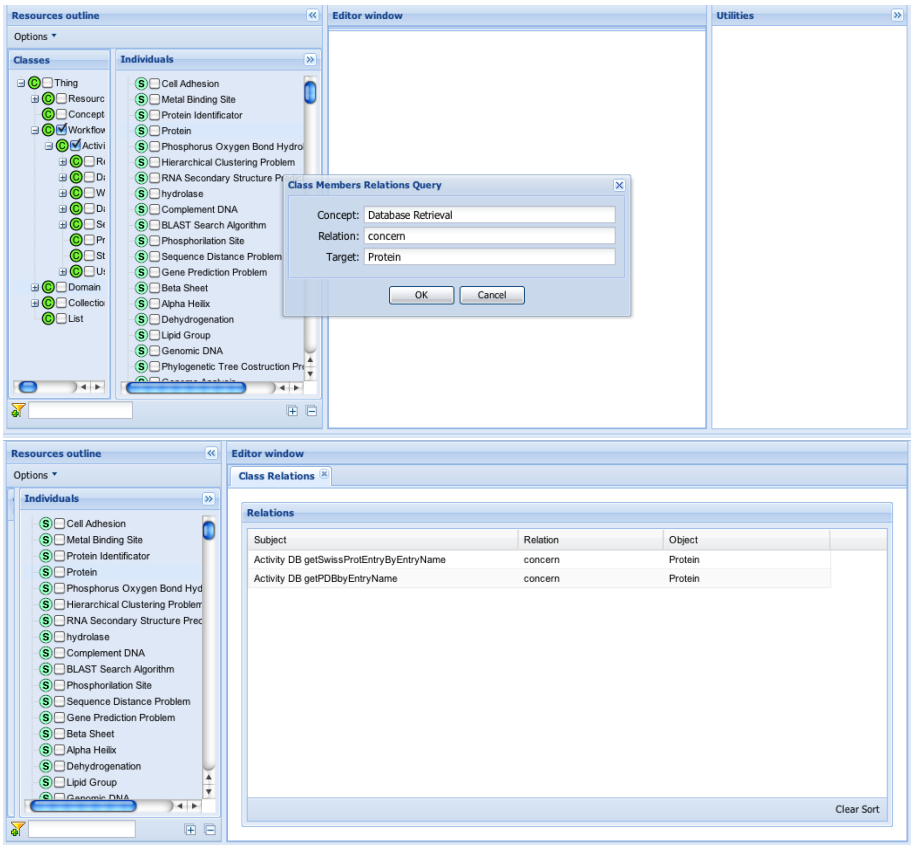

Figure 8.3: Semantic search of a Database_Retrieval activity concerning Protein.

mentation as Fig. 8.9 shows.

8.5 Related Work

Regarding KMSs in somewhat similar to $\mathcal{R}esourceome$, we can cite iTools [203] and BioNavigation [201]. The iTools framework aims at the classification and integration of the resources developed at the seven US' National Center for Biomedical computing. It is characterized by a taxonomy-like user friendly interface for browsing the managed resources, which are tools. The updating of resource metadata is delegated through interfacing (XML, SOAP and WSDL) external softwares. In the present version, iTools does not manage, differently from $\mathcal{R}esourceome$, any semantic relationship (Object Property) between resources, but only the properties (Data Property) of the visualized

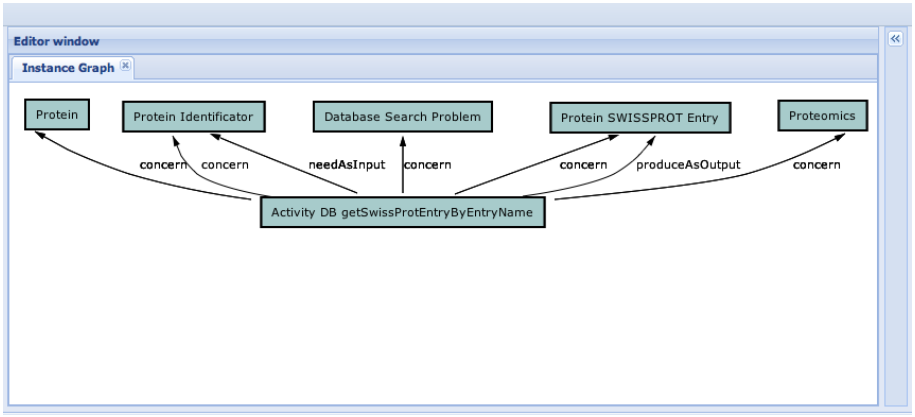

Figure 8.4: Conceptual Map of Activity getSwissProtEntryByEntryName

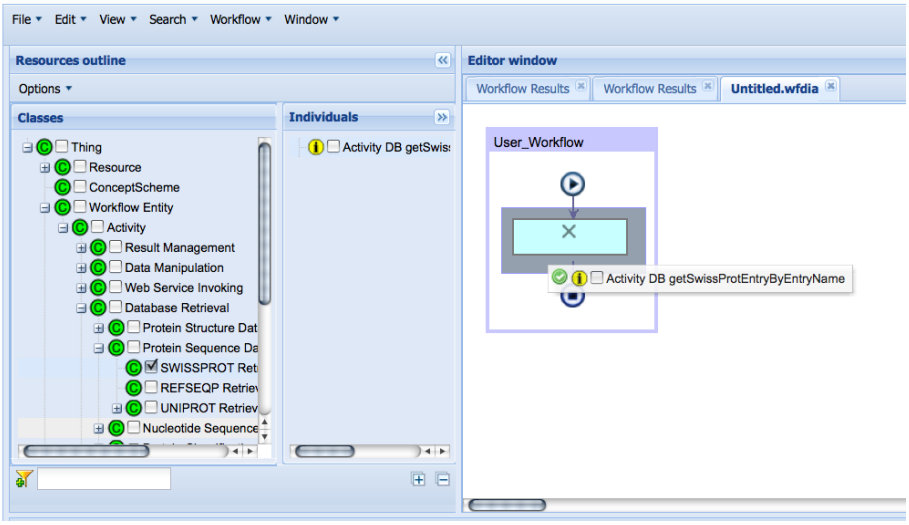

Figure 8.5: Drag and Drop of an activity into the workflow.

individuals. iTools relies on three user classes: expert editors, registered and general users. Only editors may update the iTools resource descriptions. Registered users may submit new resource descriptions (or updates), retrieve or comment on resources and their descriptions. General users may only browse the iTools resources. The main difference with $\mathcal{R}esourceome$ is the metadata representation model, which in iTools is a simple resource taxonomy.

BioNavigation metadata representation philosophy is similar to that of $\mathcal{R}esourceome$. BioNavigation's physical graph connecting data sources can be compared with our Resource Ontology and its conceptual graph can be

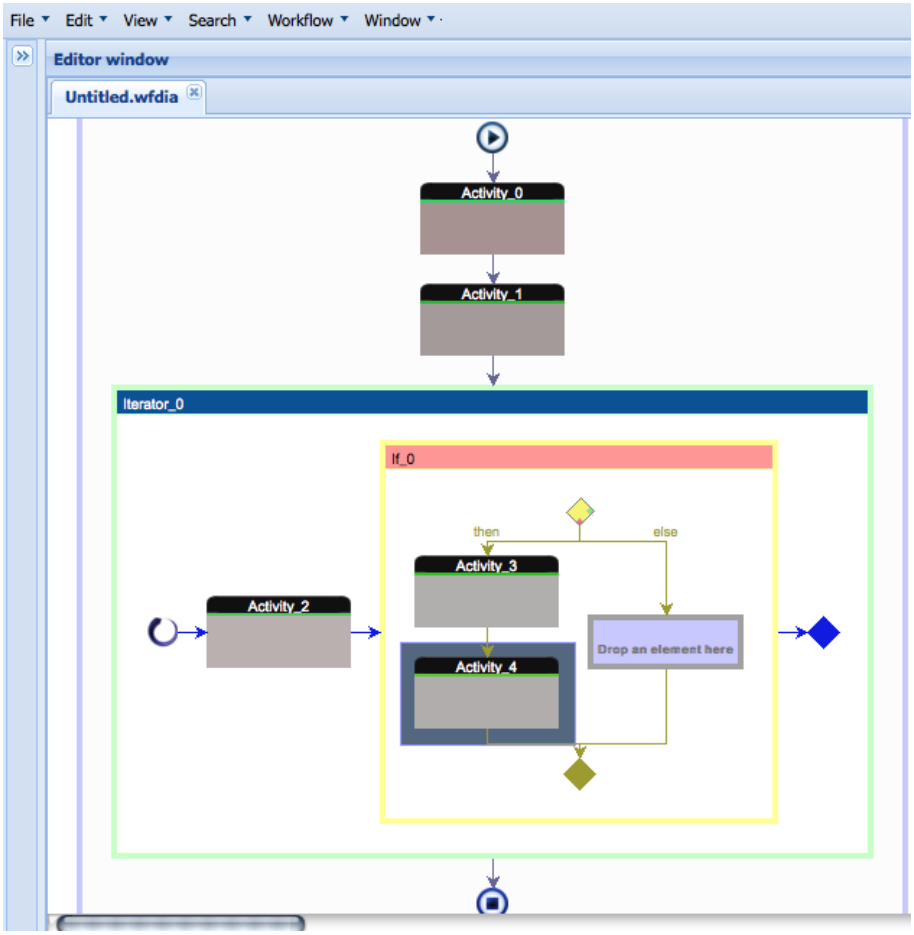

Figure 8.6: Editing a workflow in $\mathcal{R}esourceome$.

compared with our Domain Ontology. The mapping between the two graphs is in our system formalized by the relation "*concerns*" and its sub-relations. However, differently from BioNavigation, the $\mathcal{R}esourceome$ knowledge representation allows one to manage also individuals and its ontologies are published on the Semantic Web.

Regarding WMSs for creating and enacting workflows in the biological domain, Taverna [207], Wildfire [210], Pegasys [208], Kepler [195], MS-Analyzer [198] are some of the most representative platforms. Taverna - a part of MyGrid project [209] - has mainly the aim to integrate Web Services by workflows specified in a choreography language: XML Simple conceptual unified flow language (XScufl [207]). Recently it has been equipped with BioMoby [205] plugin that allows the user to access BioMoby [211], a well-

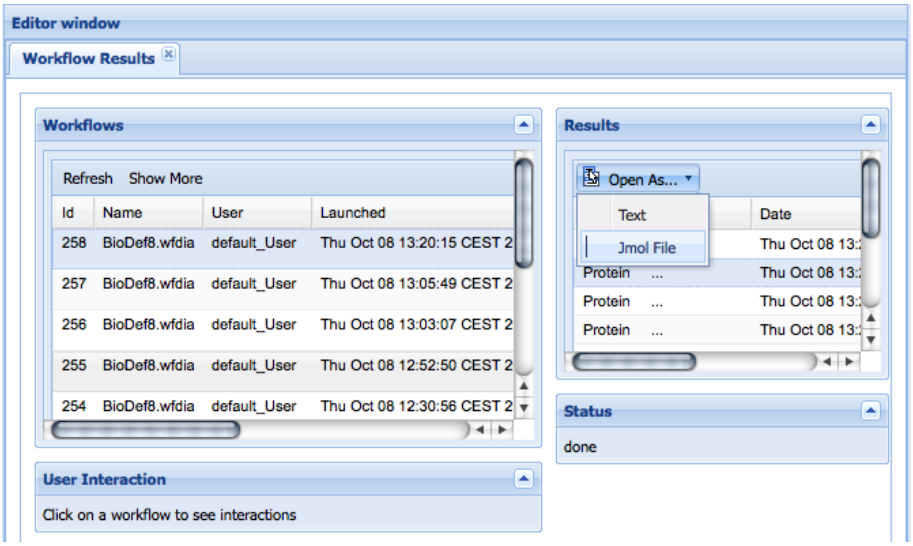

Figure 8.7: Results obtained during workflow execution.

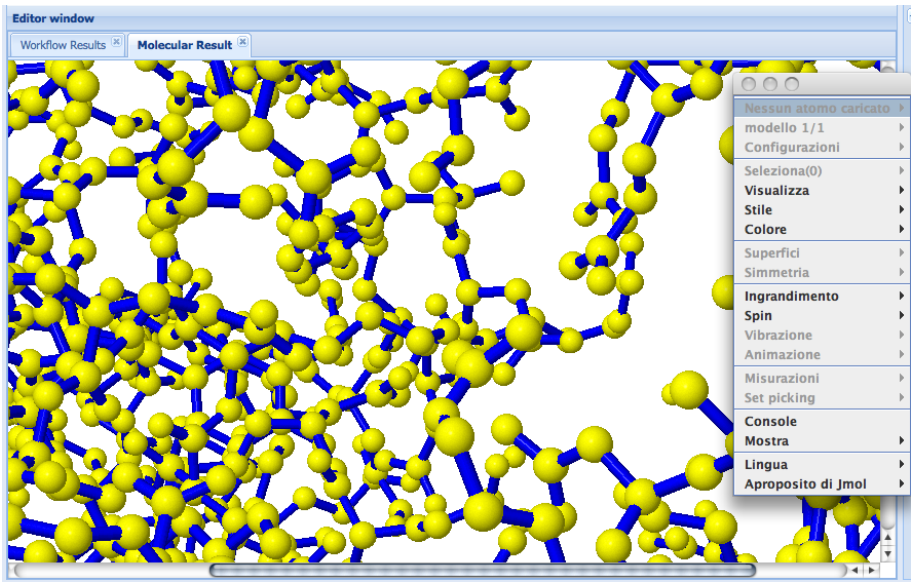

Figure 8.8: Visualization of a PDB file using Jmol in *Resourceome*.

known registry of Web services used in bioinformatics.

Being Taverna editor embedded with its engine in a Java stand-alone application, it is quite heavy, for an end-user, to download it.

Wildfire [210] is another WMS that provides an integrated environment

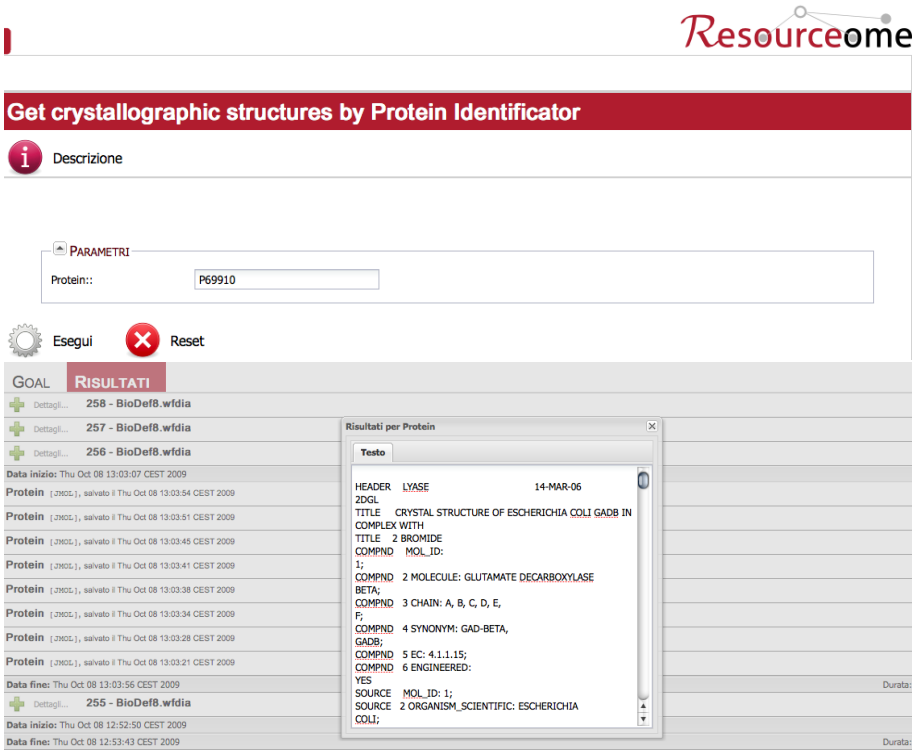

Figure 8.9: Publishing of a workflow.

for the construction and execution of workflows based only on Jemboss [199] applications. Pegasys system enables bioscientists to create and manage sequence analysis workflows. It includes numerous analytical tools and provides database capacities to maximize information captured during the execution of a workflow.

All the above mentioned WMSs describe, often with different expressive power, an in-silico experiment without using standard workflow specification languages. Moreover, they are characterized by a monolithic architecture, which does not allow the execution of workflows, or parts of them, over a distributed and heterogeneous environment due to lack of flexibility, scalability and fault tolerance, features required for a distributed cross-organizational workflow.

Kepler and MS-Analyzer attempt to overtake the above-mentioned limits. Kepler is a workflow tool based on a extension of the MoML language [206]: it is obtained by introducing the concept of a Director to define execution models and monitor workflows, where Web and Grid services, Globus Grid jobs and GridFTP can be used as components.

MS-Analyzer is the nearest software platform for bioinformatics experiments to $\mathcal{R}esourceome$. However, differently from the latter, it is strongly customized for proteomic domain: it allows the integrated preprocessing, management and data mining analysis of proteomic data and it provides various services that implement spectra management and preprocessing. In particular, the composition and execution of such services is carried out through an ontology-based workflow editor and scheduler that uses specific domain ontologies, namely WekaOntology and ProtOntology[7].

[7]WekaOntology is an ontology of the data mining domain that is used to describe the tools of the Weka suite [212] and has been enriched by the description of relevant datasets and preprocessing algorithms. ProtOntology models concepts, methods, algorithms, tools and databases relevant to the proteomic domain, and provides a biological background to the data mining analysis.

Part VII

Conclusion

Chapter 9

Conclusions

In this Thesis, a semantic framework for declarative and procedural knowledge, called $\mathcal{R}esourceome$, has been proposed. $\mathcal{R}esourceome$ is an ontology-based and domain-independent formal model for semantic annotation of *resources* and *activities*, in according to Guarino's approach [11]. It was born as an extension of the knowledge model proposed in [34] to organize the amount of resources in the biological domain.

It keeps the same name of the solution proposed in [34] but, differently from it, can be adopted in any application domains (as well as the biological one) and can also handle the procedural knowledge. As a consequence, business process modeling as well as workflow specification and execution are topics which have been investigated in the definition of $\mathcal{R}esourceome$. In particular, distributed workflow execution using multi-agent systems, together with the Web Service paradigm, have been studied and taken into account.

Furthermore, a Web-based tool, namely a semantic knowledge management system together with a semantic-driven workflow specification and execution environment, was built on the top of $\mathcal{R}esourceome$.

Finally, the biology domain has been investigated as a case study and the Web-based tool has been customized as an environment for semantic-driven in-silico experiments.

In conclusion, we believe that the whole $\mathcal{R}esourceome$ semantic framework (model and tools) can be useful - but it is not limited - to any kind of organization, research group or SIG:

- to specify and to share the proper know-how;
- to identify relevant information and exploit that information (domain and procedural knowledge) in the context of specific knowledge processing tasks;

- to support the user in a variety of knowledge-based activities. To give an idea, we can cite discovery and organization of information resources; extraction of task-relevant information using information harvesting and knowledge editing techniques; retrieval, integration and transformation of domain-relevant knowledge using semantic query capabilities; use of semantic reasoning techniques to support domain-relevant business processes.

Part VIII

Appendix

Appendix A

Resourceome Web-tool

A.1 *Resourceome* Web-tool

This section will provide a deeper view of the *Resourceome* tool, with a detailed analysis of its key features. In particular this part will attempt to explain the recommended usage of the software, giving an idea of its high potential. The *Resourceome* framework mainly consists of two components strictly interacting each other:

- *Resourceome* KMS, a graphical interface for the *Resourceome* ontology-based Knowledge Management System
- The Workflow Editor, Monitoring and Publishing Tools, constituting together the *Resourceome* WMS.

A.1.1 Workspace

The workspace is the piece of software responsible for the knowledge management and the place where the knowledge itself is stored in form of ontology. Consequently, the user can load a specific workspace according to the specific domain in which he wants to operate. The content of a workspace is graphically organised in two trees:

- **The Class Tree:** each node is an ontology class and can be expanded to display subclasses.
- **The Individuals Tree:** shows the individuals belonging to the selected concept in the class tree.

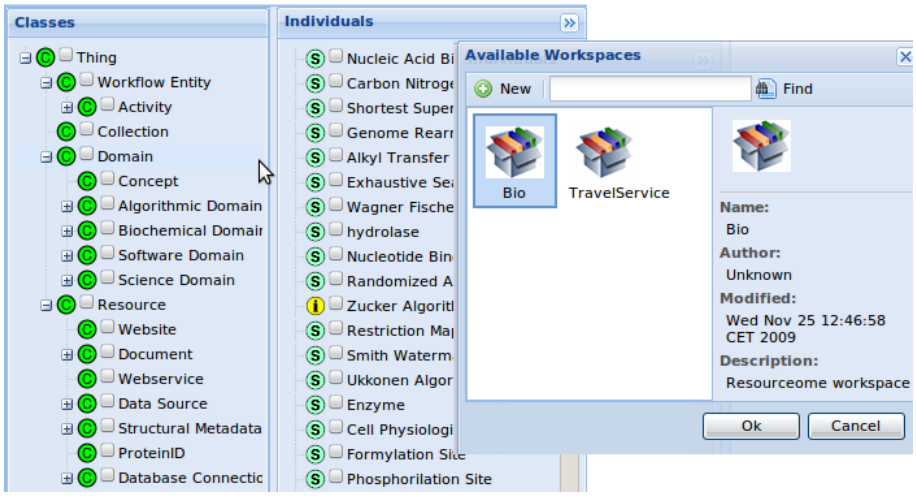

Figure A.1: Load workspace dialog and the panel displaying classes and inviduals.

One of the most interesting features of the software is the innovative visualization component, designed to provide a graphical view for the relations and classes of an individual; the resulting conceptual map, based on the HTML5 canvas tag, is provided in vector format, so can be zoomed without losing any detail. The tool also supports different visualization algorithms: Dot, Circo, TwoPi, Neato and Fdp.
In fig. A.2, it's possible to observe the conceptual maps for the individual 'Enzyme': on the left, the class diagram is rendered with the Dot engine, while, on the right, the Circo engine is responsible for the visualization of the relation diagram.
Moreover, *Resourceome* allows to perform changes to the underlying ontology at runtime; such operations are:

- **Add class/subclass**
- **Add relation:** the user has the possibility to specify if the relation is Functional, Inverse Functional, Transitive, Symmetric, Asymmetric, Reflexive or Irreflexive.
- **Add data property**

A.1.2 Workflow Editor

The workflow editor can be considered the core of the *Resourceome* WMS; this graphical tool makes possible to design complex workflows of activities

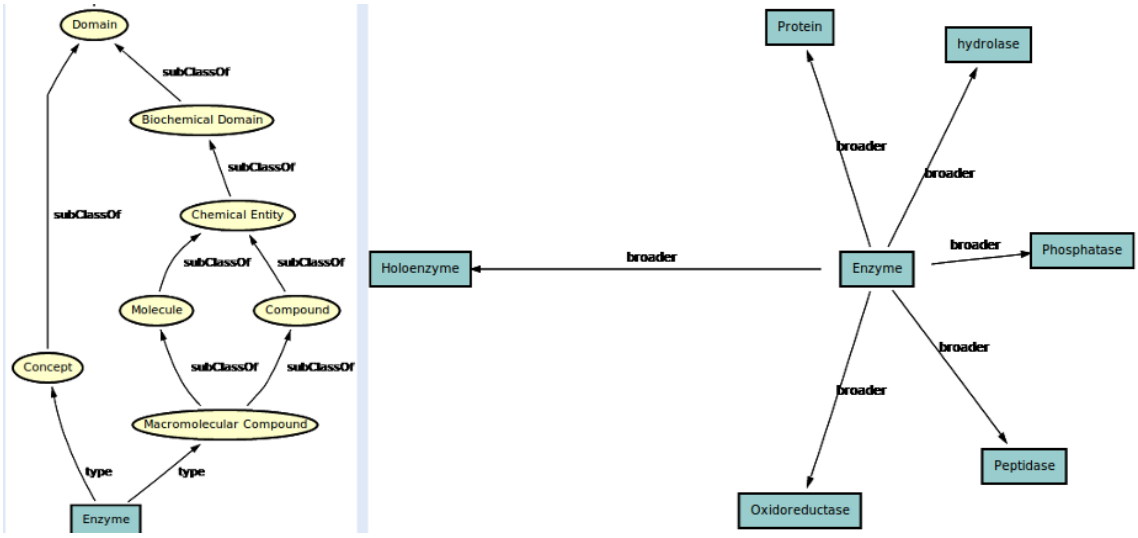

Figure A.2: Conceptual maps of the individual Enzyme

with the minimum effort for the user, and besides, differently from other traditional WMSs, the *Resourceome* knowledge model allows each workflow activity to be automatically linked to its context, roles, objects and documents; as a result, the process definition is strongly ontology-driven.
In other words, the user can find the activities as individuals in the workspace ontology, and simply can drag and drop them in the workflow editor; an activity can be configured with several input/output parameters, each of them has associated its own datatype. *Resourceome* allows the following advanced operations on parameters:

- **Piping:** this operation is used when, for example, it's necessary to pipe the output parameter, named *out_1*, of an activity as the input parameter, named *in_1*, for another activity; this can be easily performed with a drag and drop of the parameter reference for *out_1* in the textbox used to set the value for *in_1*.

- **Resources Fetching:** parameters can not only be set manually or with a link to another parameter, but *Resourceome* enables getting the required data also from the knowledge base located in the workspace; after having chosen an individual, the software prompts the user to select among the individual's data properties and the involved parameter will be set with the value of the chosen property.

A user can specify the order of activities execution using the following special control-flow patterns:

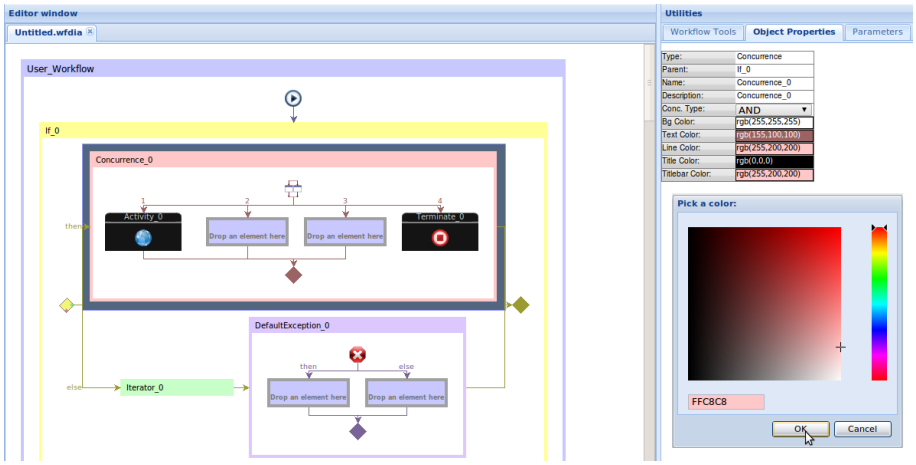

Figure A.3: The workflow editor

- Sequence: allows to execute an activity after another.
- If: defines a conditional routing where the choice of the activity to be executed is case-driven.
- Iterator: enables the cyclic execution of one or more activities until the boolean condition is true.
- Concurrence: enables the parallel execution of two or more activities.
- Default Exception: allows to define the activity to be executed if an exception or an error is previously caught.
- Terminate: allows the forced termination of the workflow.

As mentioned before, the If and the Iterator patterns needs a boolean condition to be specified, and *Resourceome* also provides a graphical editor for building a boolean formula; this editor allows the composition of logical ports (and, or, not), the comparisation between string or integer values and the inputting of boolean values.

A.1.3 Workflow Monitoring Tool

This component of the *Resourceome* WMS makes possible the monitoring of the running workflows as well as the online visualization or the download of the results produced by a specific workflow.
In particular, this tool doesn't only provide the list of the running workflows,

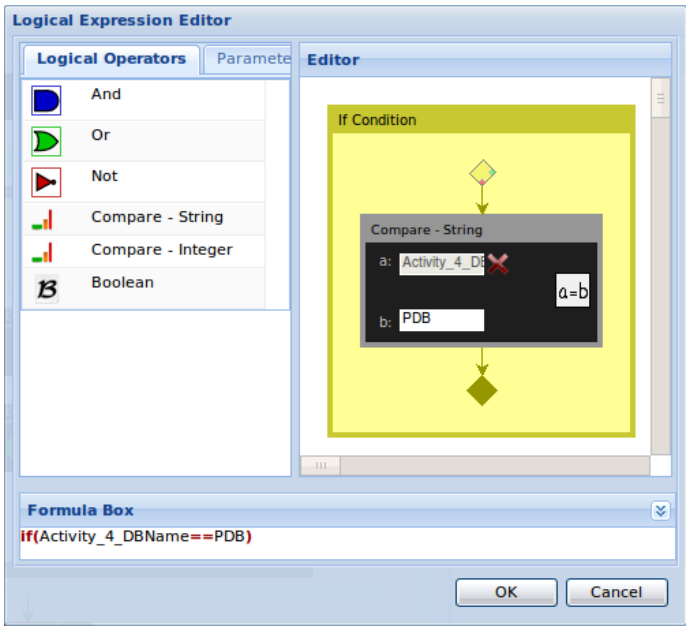

Figure A.4: The boolean formula editor

but also the whole execution hystory: for each listed workflow, the following informations are rendered:

- **Workflow Id**
- **Workflow Name**
- **Launch timestamp**
- **Termination timestamp**
- **Duration**
- **User**
- **Status** (Compiling, Running, Finished, Compilation Error, Runtime Error)

The Results panel updates automatically when selecting a workflow and shows the data outputted during its execution. Each result has its own datatype, so that *Resourceome* can associate the right plugins to visualize the data: for example an XML result can be rendered as plain text, as HTML or as a tree, while a molecular result can be opened as plain text or with the Jmol plugin.

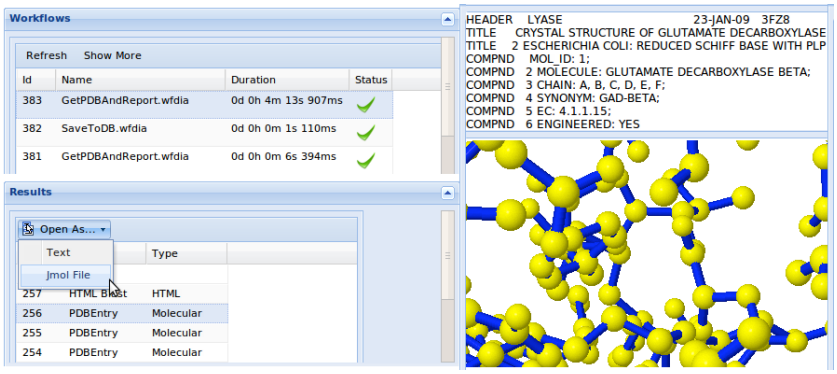

Figure A.5: Execution history, results panel and visualization plugins.

A.1.4 Workflow Publishing Tool

It is evident that *Resourceome* is a software suited to users with basic informatic skills, so this component has been designed to make the workflows available to the common user through a simplified interface. With the publishing tool, the expert user, after having designed and tested his workflow, can publish it, so that the end user has only to set some possible parameters and launch the execution of the **goal**, without explicitly knowing the implementation of the underlying workflow.
On the other hand, the following informations must be provided in order to publish a goal:

- Unique name
- Description (optional)
- A set of bindings

'Binding' is the term denoting an activity parameter intented to be set by the end user. For each binding it's necessary to specify a name to display in the simplified interface and if the parameter is optional or required. Besides, it's possible to associate a concept from the workspace to help the input choice of the end user: for example, if the user is expected to enter the name of a city to launch a goal, *Resourceome* allows the workflow designer to drag and drop the concept City from the class tree to the binding panel; in the resulting goal, the field City will have associated a set of acceptable values, obtained from the individuals belonging to the City class.

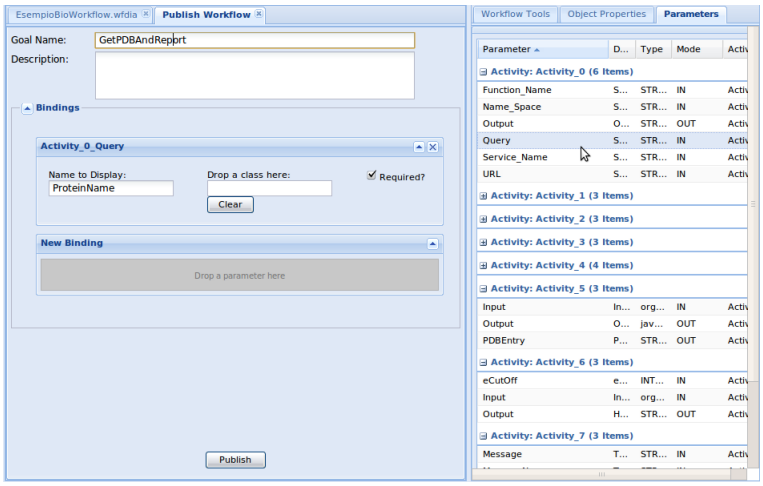

Figure A.6: A view of the publishing interface

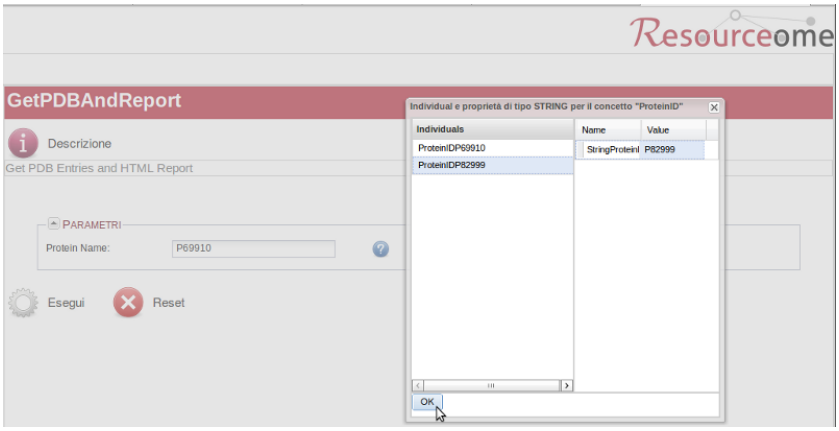

Figure A.7: A list of ontology-driven values in the simplified interface

Appendix B

XPDL-ontology

In this Chapter we describes the XPDL ontology specified through Web onltology language (OWL-DL) and then how to link it with a organisational and informational resources and activities described in Resourceome ontology.

B.1 XPDL introduction

The XML Process Definition Language (XPDL[1]) is a format standardized by the Workflow Management Coalition (WfMC[2]) to interchange Business Process definitions between different workflow products, ie between different modeling tools and management suites. XPDL defines an XML schema for specifying the declarative part of workflow / business process.

B.2 Creating XPDL semantics

In the next pages is described how to create semantic for XPDL and link it with an ontology that describes activities and resources in a specific domain. XPDL just defines the sintax (an XML schema) for specifying the declarative part of workflow / business process, so the first step is to create an ontology that describes the semantic of the XPDL format.

B.2.1 Basic modelling approach

The first step of creating an ontology over the XPDL format consists of mapping every single elmenent to its respective OWL rappresentation. The basic

[1] http://www.wfmc.org/xpdl.html
[2] http://www.wfmc.org/

```
<xsd:element name="Application">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="xpdl:Description" minOccurs="0"/>
      <xsd:choice>
        <xsd:element ref="xpdl:FormalParameters"/>
        <xsd:element ref="xpdl:ExternalReference" minOccurs="0"/>
      </xsd:choice>
      <xsd:element ref="xpdl:ExtendedAttributes" minOccurs="0"/>
    </xsd:sequence>
    <xsd:attribute name="Id" type="xsd:NMTOKEN" use="required"/>
    <xsd:attribute name="Name" type="xsd:string"/>
  </xsd:complexType>
</xsd:element>
```

Figure B.1: XML definition of Application element

approach is:

Each XML complex type element is mapped to an OWL class and each XML simple type subelement or attribute is converted to a data property.

For each XML complex type subelement an object property that descibes what relationship the parent has with it is created.

Each XML group element is mapped to a superclass of all its subelements (see figure B.5 and B.4).

Following we report how some interesting elements were mapped with this approach.

Complex element: Application

Application is a complex element defined as shown in figure B.1. It has at least the attribute *Name* and have a subelement *FormalParameters* or a subelement *ExternalReference*.

Name is an attribute so we need to create a data property that represent the relationship between an entity of type Application and a it's name. We created the *hasName* property that has as domain *Application* and as range a string.
Application is a complex type so it's mapped to an OWL class as shown in figure B.2.

We can assert that it must have exactly one data property *hasName* of type String but we cannot assert that it has exactly one object property *hasFormalParameters* or exactly one object property *hasExternalReferece* because the minOccurs value of the second one is 0. In fact we can have an

```
<SubClassOf>
  <Class URI="&XPDL-Ontology;Application"/>
    <DataExactCardinality cardinality="1">
    <DataProperty URI="&XPDL-Ontology;hasName"/>
    <Datatype URI="&xsd;string"/>
  </DataExactCardinality>
</SubClassOf>
```

Figure B.2: OWL definition of Application element

application like this:

```
<xsd:element name="Applications">
  <xsd:complexType>
    <xsd:sequence>
     <xsd:element ref="xpdl:Application" minOccurs="0"
                  maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure B.3: XML definition of Applications

So we have to define an object property *hasApplication* that is not functional (we can have more than one *Application* in each *Applications* element) that have as domains entities of *Applications* class and as range entity of *Application* class.

Group element: Datatypes

```
<xsd:group name="DataTypes">
 <xsd:choice>
    <xsd:element ref="xpdl:BasicType"/>
    <xsd:element ref="xpdl:DeclaredType"/>
    <xsd:element ref="xpdl:SchemaType"/>
    <xsd:element ref="xpdl:ExternalReference"/>
    <xsd:element ref="xpdl:RecordType"/>
    <xsd:element ref="xpdl:UnionType"/>
    <xsd:element ref="xpdl:EnumerationType"/>
    <xsd:element ref="xpdl:ArrayType"/>
    <xsd:element ref="xpdl:ListType"/>
  </xsd:choice>
</xsd:group>
```

Figure B.4: Datatypes declaration in XPDL schema

Datatypes is defined as an *xsd:group* composed of a choice between other xml elements (as shown in figure B.4) i.e. it is one of *BasicType, DeclaredType, Schematype, ExternalReference, RecordType, UnionType, EnumerationType, ArrayType* or *ListType*.

In our ontology we defined it as the union of it's subelements as shown in figure B.5

```
<EquivalentClasses>
  <Class URI="&XPDL-Ontology;DataTypes"/>
  <ObjectUnionOf>
    <Class URI="&XPDL-Ontology;ArrayType"/>
    <Class URI="&XPDL-Ontology;BasicType"/>
    <Class URI="&XPDL-Ontology;DeclaredType"/>
    <Class URI="&XPDL-Ontology;EnumerationType"/>
    <Class URI="&XPDL-Ontology;ExternalReference"/>
    <Class URI="&XPDL-Ontology;ListType"/>
    <Class URI="&XPDL-Ontology;RecordType"/>
    <Class URI="&XPDL-Ontology;SchemaType"/>
    <Class URI="&XPDL-Ontology;UnionType"/>
  </ObjectUnionOf>
</EquivalentClasses>
```

Figure B.5: OWL rappresentation of the Datatypes group

B.2.2 Linking XPDL ontology with another ontology

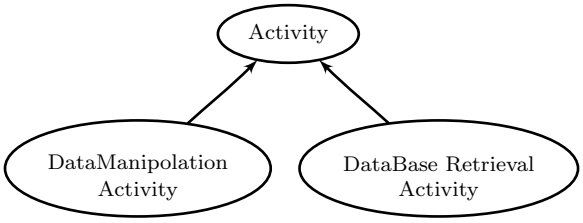

Figure B.6: Graph representation of the Activity class in $\mathcal{R}esourceome$ ontology

Now that we have defined a consistent ontology that represent an XPDL we have to link it with the one representing our resources and activity. In the following example we show how we can link them by simply making equivalent the classes of the two ontologies that have the same meaning. In figure B.6 is showed a graph representation of the class Activity in the $\mathcal{R}esourceome$ ontology.

We can assert that *Application* is an equivalent class of Activity.

As shown in figure B.7 *Activity* is now equivalent to *Application* that inherit all its subclasses. We can now define an *Activity*, add a name to it and, of course, the hasFormalparamters properties (if present) and state that this is an *Application*. In this way we added knowledge to the meaningless *Application* element in XPDL.

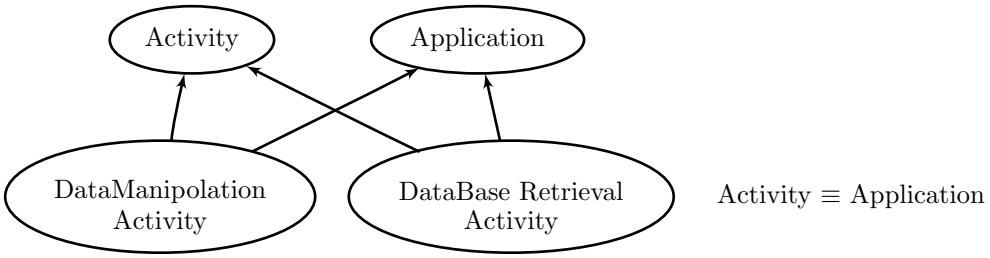

Figure B.7: Graph representation of the Activity class in $\mathcal{R}esourceome$ ontology after linking with the XPDL ontology

Bibliography

[1] SKOS Core Guide. http://www.w3.org/TR/2005/WD-swbp-skos-core-guide-20051102.

[2] SKOS Core. http://www.w3.org/2004/02/skos/core.rdf

[3] SKOS Mapping. http://www.w3.org/2004/02/skos/mapping.rdf

[4] SKOS Extensions. http://www.w3.org/2004/02/skos/extensions.rdf

[5] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F.k Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business Process Execution Language for Web Services Version 1.1, July 2002. 49, 81, 86, 111

[6] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowl. Acquis.*, 5(2):199–220, 1993.

[7] E. Bartocci, F. Corradini, E. Merelliand L. Vito, "Model driven design and implementation of activity-based applications in Hermes", Proc. of the 7th WOA 2006 Workshop, From Objects to Agents (2006).

[8] A. Bernstein, and M. Deanzer, "The next system: Towards true dynamic adaptions of semantic web service compositions (system description)", Proc. of the 4th European Semantic Web Conference (ESWC'07), Springer (2007).

[9] F. Corradini, and E Merelli, "Hermes: agent-based middleware for mobile computing", Mobile Computing, LNCS, 3465, pp. 234–270 (2005).

[10] T.R. Gruber, "A translation approach to portable ontology specifications", Knowledge Acquisition, 5(2), pp. 199-220 (1993).

[11] N. Guarino, "Formal Ontology in Information Systems: Proceedings of the 1st International Conference June 6-8, 1998, Trento, Italy" (1998).

[12] F. Baader, I. Horroks, and U Sattler, "In Festschrift in honor of Jorg Siekmann Hutter and Stephan" , in W3C Recommendation.

[13] C. Masolo, S. Borgo, A. Gangemi, Guarino, N., Oltramari, A. and L. Schneider, "Wonderweb deliverable d17. The wonderweb library of foundational ontologies and the dolce ontology" (2002).

[14] C. Masolo, G. Guizzardi, L. Vieu, E. Botazzi and R. Ferrario, "Relational roles and qua-individuals", Proc. AAAI Fall Symposium on Roles, an Interdisciplinary Perspective, pp. 103–112 (2005).

[15] K. Morik, and M. Scholz, "The miningmart approach to knowledge discovery in databases", Proc. of the International Conference on Machine Learning, Springer, pp. 47–65 (2004).

[16] OWLViz. www.co-ode.org/downloads/owlviz (15.07.2009)

[17] The Protégé Ontology Editor and Knowledge Acquisition System. http://protege.stanford.edu (15.07.2009)

[18] C.P Gomes, H. Kautz, A Sabharwal, and B Selman. 2008. Satisfiability solvers. In van Harmelen, F., Lifschitz, V., and Porter, B., eds., "Handbook of Knowledge Representation" . Elsevier.

[19] P. Borst, H. Akkermans, and J. Top "Engineering ontologies", Int. J. Human-Computer Studies 46 (1997), 365-406.

[20] A.W Scheer. "ARIS - Vom Geschftsprozess zum Anwendungssystem", Springer (1998).

[21] P. Mika, J.M. Akkermans, "Towards a New Synthesis of Ontology Technology and Knowledge Management, Knowledge Engineering Review" 19:4 (2004), 317-345

[22] SKOS Parser. http://oaei.ontologymatching.org/2008/skos2owl.html (15.07.2009)

[23] Suggested upper merged ontology (sumo). http://www.ontologyportal.org (15.07.2009)

[24] SUPER Project. http://www.ip-super.org/ (15.07.2009)

[25] A Suyama, N. Negishi, and T Yamaguchi, "CAMLET: A platform for automatic composition of inductive learning systems using ontologies", Proc. of PRICAI'98, LNCS, 1531, pp. 205–215, (1998).

[26] R Wirth, C. Shearer, U. Grimmer, T.P Reinartz, J Schloesser, C Breitner, R. Engels, and G Lindner, "Towards process-oriented tool support for knowledge discovery in databases", Proc. of the First European Symposium on Principles of Data Mining and Knowledge Discovery, LNCS, 1263, pp. 243–253 (1997).

[27] M. Angeletti, A. Baldoncini, N. Cannata, F. Corradini, R. Culmone, C. Forcato, M. Mattioni, E. Merelli, and R. Piergallini. Orion: A spatial multi agent system framework for computational cellular dynamics of metabolic pathways. In *Proceedings of Bioinformatics ITalian Society (BITS) Meeting*, Bolgna, Italy, 2006.

[28] E. Bartocci, D. Cacciagrano, N. Cannata, F. Corradini, E. Merelli, and L. Milanesi. A GRID-based multilayer architecture for bioinformatics. In *Proceedings of NETTAB'06 Network Tools and Applications in Biology*, Santa Margherita di Pula, Cagliari, Italy, 2006.

[29] E. Bartocci, F. Corradini, and E. Merelli. BioWMS: A web based workflow management system for bioinformatics. In *Proceedings of Bioinformatics ITalian Society (BITS) Meeting*, Bologna, Italy, 2006.

[30] E. Bartocci, F. Corradini, and E. Merelli. Building a multiagent system from a user workflow specification. In *Proceedings of Workshop From Objects to Agents - WOA*, 2006.

[31] T. Bellwood, L. Clément, D. Ehnebuske, A. Hately, M. Hondo, Y. L. Husband, K. Januszewski, S. Lee, B. McKee, J. Munter, and C. von Riegen. UDDI version 3.0. Published specification, Oasis, 2002.

[32] D. Bonura, F. Corradini, E. Merelli, and G. Romiti. Farmas: a MAS for extended quality workflow. In *2nd IEEE International Workshop on Theory and Practice of Open Computational Systems*. IEEE Computer Society Press, 2004.

[33] D. Bonura, L. Mariani, and E. Merelli. Designing modular agent systems. In *Proceedings of NET.Object DAYS, Erfurt*, pages 245–263, September 2003.

[34] N. Cannata, E. Merelli, and R. B. Altman. Time to organize the bioinformatics resourceome. *PLoS Comput Biol.*, 1(7) e76, 2005.

[35] F. Corradini and E. Merelli. Hermes: agent-based middleware for mobile computing. In *Mobile Computing*, volume 3465, pages 234–270. LNCS, 2005.

[36] Enhydra. Jawe. http://jawe.enhydra.org/, 2003.

[37] A. Fuggetta, G. Picco, and G. Vigna. Understanding code mobility. *IEEE Transaction of Software Engineering*, 24(5):352–361, May 1998.

[38] D. Hollingsworth. The Workflow Reference Model, January 1995.

[39] E. Merelli, R. Culmone, and L. Mariani. Bioagent: a mobile agent system for bioscientists. In *NETTAB Workshop on Agents Nd Bioinformatics*, Bologna, July 2002.

[40] WfMC. Xml process definition language (xpdl). WfMC standard, W3C, October 2005.

[41] W. Aalst. Putting Petri nets to work in industry. *Computers in Industry*, 25(1):45–54, 1994.

[42] T. Andrew, F. Curbera, H. Dholakia, Y. Goland, and et al. Business process execution language (bpel) for web services version 1.1. Technical report, IBM, 2003.

[43] D. Benson, I. Karsch-Mizrachi, D. Lipman, J. Ostell, and D. Wheeler. Genbank. *Nucleic Acids Res.*, 34:D16–20, 2006.

[44] H. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. Bhat, W. H., S. I.N., and B. P.E. Wildfire: distributed, grid-enabled workflow construction and execution. *Nucleic Acids Res.*, 28(1):235–42, 2000.

[45] A. Bertolino, F. Corradini, P. Inverardi, and H. Muccini. Deriving test plans from architectural descriptions. In *ICSE*, pages 220–229, 2000.

[46] A. Bertolino, P. Inverardi, and H. Muccini. An explorative journey from architectural tests definition downto code tests execution. In *ICSE*, pages 211–220. IEEE Computer Society, 2001.

[47] A. Cichocki. *Migrating workflows and their transactional properties.* PhD thesis, 1998. Adviser-Marek Rusinkiewicz.

[48] F. Corradini and E. Merelli. Hermes: agent-based middleware for mobile computing. In *Mobile Computing*, volume 3465, pages 234–270. LNCS, 2005.

[49] J. Ferber. *Multi-Agent System: An Introduction to Distributed Artificial Intelligence.* Addison-Wesley, 1999.

[50] I. Foster and C. Kesselman. *The Grid: Blueprint for a Future Computing Infrastructure.* Morgan Kaufmann Publishers, San Francisco, CA, 1998.

[51] I. T. Foster, N. R. Jennings, and C. Kesselman. Brain meets brawn: Why grid and agents need each other. In *AAMAS*, pages 8–15. IEEE Computer Society, 2004.

[52] A. Group. Vtl reference guide. http://jakarta.apache.org/velocity/docs/vtl-reference-guide.html.

[53] T. Hey and A. E. Trefethen. Cyberinfrastructure for e-Science. *Science*, 308(5723):817–821, 2005.

[54] N. R. Jennings. An agent-based approach for building complex software systems. *Commun. ACM*, 44(4):35–41, 2001.

[55] K. Jensen. *Coloured Petri Nets. Basic concept, analysis methods and practical use.* EATCS monographs on Theoretical Computer Science. Springer-Verlag, Berlin, 1996.

[56] T. Murata. Petri nets: Properties, analysis and applications. In *Proceedings of the IEEE*, volume 77, pages 541–580, April 1989.

[57] C. A. Petri. *Kommunikation mit Automaten.* PhD thesis, Institut für instrumentelle Matematik, Bonn, 1962.

[58] Sun. Java architecture for xml binding (jaxb). http://java.sun.com/webservices/jaxb/.

[59] W. van der Aalst. The application of petri nets to workflow management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.

[60] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.

[61] WfMC. Workflow management coalition terminology and glossary. Technical Report WFMC-TC-1011, Workflow Management Coalition, 1999.

[62] K. v. H. W.M.P. van der Aalst. *Workflow Management Models, Methods and Systems.* MIT Press, Cambridge, 2002.

[63] WfMC. Xml process definition language (xpdl). WfMC standard, W3C, October 2005.

[64] H. Boley, S. Taber, G. Wagner: Design Rationale of RuleML: A Markup Language for SemanticWeb Rules. Proc. of SWWS'01 (2001).

[65] G. Wagner, S. Tabet, H. Boley: MOF-RuleML: The Abstract Syntax of RuleML as a MOF Model. OMG Meeting. Boston (2003).

[66] M. Paolucci, N. Srinivasan, K. Sycara: Adding OWL-S to UDDI, implementation and throughput. First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004).

[67] P. Haase, L. Stojanovic: Consistent Evolution of OWL Ontologies. Proc. of the Second European Semantic Web Conference (ESWC 2005). LNCS **3532** (2005) 182–197.

[68] M. Richters, M. Gogolla: On Formalizing the UML Constraint Language OCL. Proc. of 17th Int. Conf. Conceptual Modeling (ER'98). LNCS **1507** (1998) 449–464.

[69] I. Horrocks, P. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, M. Dean: SWRL: A Semantic Web Rule Language: Combining OWL and RuleML. W3C Member Submission (2004). `http://www.w3.org/Submission/SWRL`.

[70] B. Grosof, R. Volz, S. Decker: Description logic programs: Combining logic programs with description logic. Proc. of the Twelfth International World Wide Web Conference (WWW 2003).

[71] J. Warmer, A. Kleppe: The Object Constraint Language: Precise Modeling with UML. Addison-Wesley (1998).

[72] XlinkIt: A Consistency Checking and Smart Link Generation Service. ACM Transactions on Software Engineering and Methodology (2002) 155–185.

[73] CLiX: Constraint Language in XML. `www.clixml.org/clix/1.0`.

[74] Open CLiX: an open source CLiXML Schema Validator. `http://clixml.sourceforge.net`

[75] RuleML. The Rule Markup Initiative. Date 22nd October 2004. `www.ruleml.org`.

[76] The Schematron Assertion Language. http://www.ascc.net/xml/schematron

[77] UML: Unified Model Language. www.uml.org

[78] XML Path Language (XPath) Version 2.0. W3C Recommendation. http://www.w3.org/TR/xpath20

[79] WSDL: Web Service Definition Language. www.w3.org/TR/wsdl.

[80] WSDL2Java. http://ws.apache.org/axis/java/user-guide.html

[81] W3C Web Services Activity. www.w3.org/2002/ws

[82] OWL-S, DAML Web Service Ontology. http://www.daml.org/services/owl-s

[83] Klasse Objecten. OCL Center: OCL Tool. http://www.klasse.nl/ocl/index.htm

[84] Object Constraint Language Specification. version 2.0. www.klasse.nl/ocl/ocl-subm.html

[85] Web Service Semantics: WSDL-S. www.w3.org/Submission/WSDL-S

[86] W3C XML Schema. www.w3.org/XML/Schema

[87] The Alloy analyzer, http://alloy.mit.edu/.

[88] BPML, http://www.bpmi.org/.

[89] OCL 1.4 syntax checker, http://www.klasse.nl/ocl/.

[90] OCL V. 2.0, http://www.klasse.nl/ocl/ocl-subm.html/.

[91] OCLE 1.0, http://lci.cs.ubbcluj.ro/ocle/.

[92] W3C XML Schema, http://www.w3.org/xml/schema/.

[93] WS-CDL, http://www.w3.org/tr/ws-cdl-10/.

[94] WSCI, http://www.w3.org/tr/wsci/.

[95] WSCL, http://www.w3.org/tr/wscl10/.

[96] WSDL, http://www.w3.org/tr/wsdl/.

[97] SOAPhttp://www.w3.org/TR/soap/

[98] XPath V. 2.0, http://www.w3.org/tr/xpath20/.

[99] K. Anastasakis, B. Bordbar, G. Georg, and I. Ray. UML2Alloy: A challenging model transformation. pages 436–450. 2007.

[100] D. B. Aredo. A framework for semantics of UML Sequence Diagrams in PVS. *Journal of Universal Computer Science*, 8(7):674–697, 2002.

[101] A. D. Brucker and B. Wolff. A proposal for a formal OCL semantics in Isabelle/HOL. In *TPHOLs '02: Proceedings of the 15th International Conference on Theorem Proving in Higher Order Logics*, pages 99–114, London, UK, 2002. Springer-Verlag.

[102] T. Bultan, X. Fu, R. Hull, and J. Su. Conversation specification: a new approach to design and analysis of e-service composition. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 403–410, New York, NY, USA, 2003. ACM Press.

[103] D. Cacciagrano, F. Corradini, R. Culmone, and L. Vito. Dynamic constraint-based invocation of Web Services. In M. Bravetti, M. Nez, and G. Zavattaro, editors, *WS-FM*, volume 4184 of *Lecture Notes in Computer Science*, pages 138–147. Springer, 2006.

[104] H. Foster, S. Uchitel, J. Magee, and J. Kramer. Model-based verification of Web Service compositions. pages 152–163. IEEE Computer Society, 2003.

[105] X. Fu, T. Bultan, and J. Su. Analysis of interacting BPEL Web Services. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 621–630, New York, NY, USA, 2004. ACM.

[106] M. D. M. Gallardo, P. Merino, and E. Pimentel. Debugging UML designs with model checking. *Journal of Object Technology*, 1:101–117, 2002.

[107] G. Georg, J. Bieman, and R. France. Using Alloy and UML/OCL to specify runtime configuration management: A case study. In *Practical UML-Based Rigorous Development Methods - Countering or Integrating the eXtremists. Volume P-7 of LNI., German Informatics Society*, pages 128–141, 2001.

[108] G. J. Holzmann. The model checker SPIN. *Software Engineering*, 23(5):279–295, 1997.

[109] H. Hussmann, B. Demuth, and F. Finger. Modular architecture for a toolset supporting OCL. In A. Evans, S. Kent, and B. Selic, editors, *UML 2000 - The Unified Modeling Language. Advancing the Standard. Third International Conference, York, UK, October 2000, Proceedings*, volume 1939 of *LNCS*, pages 278–293. Springer, 2000.

[110] R. Marcano and N. Levy. Transformation rules of OCL constraints into B formal expressions. In J. Jürjens, M. V. Cengarle, E. B. Fernandez, B. Rumpe, and R. Sandner, editors, *Critical Systems Development with UML – Proceedings of the UML'02 workshop*, pages 155–162. Technische Universität München, Institut für Informatik, 2002.

[111] M. Richters and M. Gogolla. Validating UML models and OCL constraints. pages 265–277. Springer, 2000.

[112] D. Cacciagrano, F. Corradini, R. Culmone, and L. Vito, Dynamic constraint-based invocation of Web Services, Proc. of WSFM 2006, LNCS, 4184, pp. 138-147, 2006.

[113] X. Fu, T. Bultan, and J. Su, Analysis of interacting BPEL Web Services, in the proc. of WWW 04 (the 13th international conference on World Wide Web), ACM, pp. 621-630, 2004.

[114] H. Foster, S. Uchitel, J. Magee, and J. Kramer, Model-based verification of Web Service compositions, IEEE Computer Society, pp. 152-163, 2003.

[115] D. Knuth, Semantics of context-free languages, Theory of Computing Systems, 2(2), pp. 127-145, 1968.

[116] M. Meriste and J. Penjam, Attributed models of executable specifications, Proc. of PLILPS 95 (the 7th International Symposium on Programming Languages: Implementations, Logics and Programs), pp. 459-460, 1995.

[117] J. Penjam, Attributed Automata: A formal model for protocol specification, technical report ISRN KTH/IT/R94/30SE, 1994.

[118] G.J. Holzmann, The model checker SPIN, Software Engineering, 23(5), pp. 279-295, 1997.

[119] C. Nentwich, L. Capra, W. Emmerich, and A. Finkelstein, Xlinkit: a consistency checking and smart link generation service, ACM Trans. Interet Technol., 2(2), pp. 151-185, 2002.

[120] BPML, http://www.bpmi.org/20.02.2009

[121] WS-CDL, http://www.w3.org/tr/ws-cdl-10/20.02.2009

[122] WSCI, http://www.w3.org/tr/wsci/20.02.2009

[123] WSCL, http://www.w3.org/tr/wscl10/20.02.2009

[124] Christopher J.O. Baker, Arash Shaban-Nejad, Xiao Su, Volker Haarslev, and Greg Butler. Semantic web infrastructure for fungal enzyme biotechnologists. *J. Web Sem.*, 4(3):168–180, 2006.

[125] Nicola Cannata, Emanuela Merelli, and Russ B. Altman. Time to organize the bioinformatics resourceome. *PLoS Comput Biol.*, 1(7):e76, 2005.

[126] PG Baker, CA Goble, S Bechhofer, NW Paton, R Stevens, and A Brass. An ontology for bioinformatics applications. *Bioinformatics*, 15(6):510–520, 1999.

[127] Jonathan B. L. Bard and Seung Y. Rhee. Ontologies in biology: design, applications and future challenges. *Nature Reviews Genetics*, 5(3):213–222, 2004.

[128] Ezio Bartocci, Diletta Cacciagrano, Nicola Cannata, Flavio Corradini, Emanuela Merelli, Luciano Milanesi, and Paolo Romano. An Agent-based Multilayer Architecture for Bioinformatics Grids. *IEEE transactions on Nanobioscience*, 6(2):142–148, 2007.

[129] Sean Bechhofer, Robert D. Stevens, and Phillip W. Lord. Gohse: Ontology driven linking of biology resources. *J. Web Sem.*, 4(3):155–163, 2006.

[130] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Sci Am.*, 284:34–43, 2001.

[131] Judith A. Blake and Carol J. Bult. Beyond the data deluge: Data integration and bio-ontologies. *Journal of Biomedical Informatics*, 39(3):314–320, 2006.

[132] Olivier Bodenreider and Robert Stevens. Bio-ontologies: current trends and future directions. *Brief Bioinform*, 7(3):256–274, 2006.

[133] M.S. Boguski. Bioinformatics–a new era. *Trends Guide to Bioinformatics, Trends Supplement*, pages 1–3, 1998.

[134] Alvis Brazma, Pascal Hingamp, John Quackenbush, Gavin Sherlock, Paul Spellman, Chris Stoeckert, John Aach, Wilhelm Ansorge, Catherine A. Ball, Helen C. Causton, Terry Gaasterland, Patrick Glenisson, Frank C.P. Holstege, Irene F. Kim, Victor Markowitz, John C. Matese, Helen Parkinson, Alan Robinson, Ugis Sarkans, Steffen Schulze-Kremer, Jason Stewart, Ronald Taylor, Jaak Vilo, and Martin Vingron. Minimum information about a microarray experiment (miame)- toward standards for microarray data. *Nat Genet*, 29(4):365–371, 2001.

[135] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30(1-7):107–117, 1998.

[136] Kenneth H. Buetow. Cyberinfrastructure: Empowering a "Third Way" in Biomedical Research. *Science*, 308(5723):821–824, 2005.

[137] Declan Butler. Mashups mix data into global service. *Nature*, 439(7072):6–7, 2005.

[138] Nicola Cannata, Flavio Corradini, and Emanuela Merelli. A resourceomic grid for bioinformatics. *Future Generation Comp. Syst.*, 23(3):510–516, 2007.

[139] Nicola Cannata, Flavio Corradini, Emanuela Merelli, Andrea Omicini, and Alessandro Ricci. An agent-oriented conceptual framework for systems biology. In Corrado Priami, Emanuela Merelli, Pedro Pablo Gonzalez, and Andrea Omicini, editors, *T. Comp. Sys. Biology*, volume 3737 of *Lecture Notes in Computer Science*, pages 105–122. Springer, 2005.

[140] Mario Cannataro et al. Algorithms and databases in bioinformatics: Towards a proteomic ontology. In *ITCC (1)*, pages 322–328. IEEE, 2005.

[141] Yi-Bu Chen, Ansuman Chattopadhyay, Phillip Bergen, Cynthia Gadd, and Nancy Tannery. The Online Bioinformatics Resources Collection at the University of Pittsburgh Health Sciences Library System–a one-stop gateway to online bioinformatics databases and software tools. *Nucl. Acids Res.*, 35(suppl_1):D780–785, 2007.

[142] Sarah Cohen-Boulakia, Susan B. Davidson, Christine Froidevaux, Zoé Lacroix, and Maria-Esther Vidal. Path-based systems to guide scientists in the maze of biological data sources. *J. Bioinformatics and Computational Biology*, 4(5):1069–1096, 2006.

[143] The Gene Ontology Consortium. Gene ontology: tool for the unification of biology. *Nature Genet.*, 25.

[144] Remko de Knikker, Youjun Guo, Jin-long Li, Albert Kwan, Kevin Yip, David Cheung, and Kei-Hoi Cheung. A web services choreography scenario for interoperating bioinformatics applications. *BMC Bioinformatics*, 5(1):25, 2004.

[145] David De Roure and James A. Hendler. E-science: The Grid and the Semantic Web. *IEEE Intelligent Systems*, 19(1):65–71, 2004.

[146] David De Roure, Nicholas R. Jennings, and Nigel R. Shadbolt. The Semantic Grid: A future e-science infrastructure. *Grid Computing*, 2003.

[147] R.P. Dellavalle, E.J. Hester, L.F. Heilig, A.L. Drake, J.W. Kuntzman, M. Graber, and L.M. Schilling. INFORMATION SCIENCE: Going, Going, Gone: Lost Internet References, 2003.

[148] Andreas Doms and Michael Schroeder. GoPubMed: exploring PubMed with the Gene Ontology. *Nucl. Acids Res.*, 33(suppl_2):W783–786, 2005.

[149] Zhuo Fang, Jiong Yang, Yixue Li, Qing ming Luo, and Lei Liu. Knowledge guided analysis of microarray data. pages 401–411, 2006.

[150] David A. Fenstermacher. Introduction to bioinformatics. *JASIST*, 56(5):440–446, 2005.

[151] Ian Foster and Carl Kesselman. *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann Publishers, San Francisco, CA, 1998.

[152] Ian T. Foster, Nicholas R. Jennings, and Carl Kesselman. Brain meets brawn: Why grid and agents need each other. In *AAMAS*, pages 8–15. IEEE Computer Society, 2004.

[153] Joanne A. Fox, Scott McMillan, and B. F. Francis Ouellette. Conductin Research on the Web: 2007 Update for the Bioinformatics Links Directory. *Nucl. Acids Res.*, 35(suppl_2):W3–5, 2007.

[154] Michael Y. Galperin. The Molecular Biology Database Collection: 2007 update. *Nucl. Acids Res.*, 35(suppl_1):D3–4, 2007.

[155] Yong Gao, June Kinoshita, Elizabeth Wu, Eric Miller, Ryan Lee, Andy Seaborne, Steve Cayzer, and Tim Clark. Swan: A distributed knowledge infrastructure for alzheimer disease research. *J. Web Sem.*, 4(3):222–228, 2006.

[156] Yolanda Gil. On agents and grids: Creating the fabric for a new generation of distributed intelligent systems. *J. Web Sem.*, 4(2):116–123, 2006.

[157] L. Grivell. Mining the bibliome: searching for a needle in a haystack? *EMBO Reports*, 3(3):200–203, 2002.

[158] N. Guarino. Formal ontologies and information systems. In *Proceedings of FOIS*, pages 3–15, 1998.

[159] Nicola Guarino and Christopher A. Welty. Evaluating ontological decisions with ontoclean. *Commun. ACM*, 45(2):61–65, 2002.

[160] James Hendler. Science and the semantic web. *Science*, 299(5606):520–521, 2003.

[161] Tony Hey and Anne E. Trefethen. Cyberinfrastructure for e-Science. *Science*, 308(5723):817–821, 2005.

[162] NR Jennings and M. Wooldridge. Applications of intelligent agents. In *Agent technology: foundations, applications, and markets table of contents*, pages 3–28. Springer-Verlag New York, Inc. Secaucus, NJ, USA, 1998.

[163] Lars Juhl Jensen, Jasmin Saric, and Peer Bork. Literature mining for the biologist: from information retrieval to biological discovery. *Nat Rev Genet*, 7:119–129, 2006.

[164] H. Kitano. *Foundations of systems biology*. MIT Press Cambridge, 2001.

[165] Hiroaki Kitano. Computational systems biology. *Nature*, 420(6912):206–210, 2002.

[166] Nicolas Le Novere, Andrew Finney, Michael Hucka, Upinder S Bhalla, Fabien Campagne, Julio Collado-Vides, Edmund J Crampin, Matt Halstead, Edda Klipp, Pedro Mendes, Poul Nielsen, Herbert Sauro, Bruce Shapiro, Jacky L Snoep, and Barry L Spence, Hugh D Wanner. Minimum information requested in the annotation of biochemical models (miriam). *Nat Biotech*, 23(12):1509–1515, 2005.

[167] Ketan K. Mane and Katy Borner. Mapping topics and topic bursts in PNAS. *PNAS*, 101(suppl_1):5287–5290, 2004.

[168] Herve Menager, Zoe Lacroix, and Pierre Tuffery. Bioinformatics services discovery using ontology classification. In *Proceedings of IEEE Congress on Services (Services 2007)*, pages 106–113, Los Alamitos, CA, USA, 2007. IEEE Computer Society.

[169] Emanuela Merelli, Giuliano Armano, Nicola Cannata, Flavio Corradini, Mark d'Inverno, Andreas Doms, Phillip Lord, Andrew Martin, Luciano Milanesi, Steffen Moller, Michael Schroeder, and Michael Luck. Agents in bioinformatics, computational and systems biology. *Brief Bioinform*, 8(1):45–59, 2007.

[170] Eric Neumann. A life science semantic web: Are we there yet? *Sci. STKE*, 2005(283):pe22, 2005.

[171] Eric K. Neumann and Denis Quann. Biodash: A semantic web dashboard for drug development. In Russ B. Altman, Tiffany Murray, Teri E. Klein, A. Keith Dunker, and Lawrence Hunter, editors, *Pacific Symposium on Biocomputing*. World Scientific, 2006.

[172] Tom Oinn, Matthew Addis, Justin Ferris, Darren Marvin, Martin Senger, Mark Greenwood, Tim Carver, Kevin Glover, Matthew R. Pocock, Anil Wipat, and Peter Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, 2004.

[173] Sandra Orchard, Lukasz Salwinski, Samuel Kerrien, Luisa Montecchi-Palazzi, Matthias Oesterheld, Volker Stumpflen, Arnaud Ceol, Andrew Chatr-aryamontri, John Armstrong, Peter Woollard, John J Salama, Susan Moore, Jerome Wojcik, Gary D Bader, Marc Vidal, Michael E Cusick, Mark Gerstein, Anne-Claude Gavin, Giulio Superti-Furga, Jack Greenblatt, Joel Bader, Peter Uetz, Mike Tyers, Pierre Legrain, Stan Fields, Nicola Mulder, Michael Gilson, Michael Niepmann, Lyle Burgoon, Javier De Las Rivas, Carlos Prieto, Victoria M Perreau, Chris Hogue, Hans-Werner Mewes, Rolf Apweiler, Ioannis Xenarios, David Eisenberg, Gianni Cesareni, and Henning Hermjakob. The minimum information required for reporting a molecular interaction experiment (mimix). *Nat Biotech*, 25(8):894–898, 2007.

[174] Christos Ouzounis and Pierre Mazire. Maps, books and other metaphors for systems biology. *Biosystems*, 85(1):6–10, 2006.

[175] Christos A. Ouzounis and Alfonso Valencia. Early bioinformatics: the birth of a discipline–a personal view. *Bioinformatics*, 19(17):2176–2190, 2003.

[176] Carolina Perez-Iratxeta, Miguel A. Andrade-Navarro, and Jonathan D. Wren. Evolving research trends in bioinformatics. *Brief Bioinform*, 8(2):88–95, 2007.

[177] Corrado Priami and Paola Quaglia. Modelling the dynamics of biosystems. *Brief Bioinform*, 5(3):259–269, 2004.

[178] D. A. Quan and R. Karger. How to make a semantic web browser. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 255–265, New York, NY, USA, 2004. ACM Press.

[179] Alan Ruttenberg, Tim Clark, William Bug, Matthias Samwald, Olivier Bodenreider, Helen Chen, Donald Doherty, Kerstin Forsberg, Yong Gao, Vipul Kashyap, June Kinoshita, Joanne Luciano, M Scott Marshall, Chimezie Ogbuji, Jonathan Rees, Susie Stephens, Gwendolyn Wong, Elizabeth Wu, Davide Zaccagnini, Tonya Hongsermeier, Eric Neumann, Ivan Herman, and Kei-Hoi Cheung. Advancing translational research with the semantic web. *BMC Bioinformatics*, 8(Suppl 3):S2, 2007.

[180] Satya S. Sahoo, Amit P. Sheth, Blake Hunter, and William S. York. Sembowser - semantic biological web services. In Christopher J.O. Baker and Kei-Hoi Cheung, editors, *Semantic Web. Revolutionizing Knowledge Discovery in the Life Sciences*, pages 317–340. Springer US, 2007.

[181] Michael Schroeder, Albert Burger, Patty Kostkova, Robert Stevens, Bianca Habermann, and Rose Dieng-Kuntz. Sealifeeee: A Semantic Grid Browser for the Life Sciences Applied to the Study of Infectious Diseases. In *Proceedings of HealthGRID 2006, Valencia, Spain (7th–9th June 2006)*, volume 120 of *Studies in Health Technology and Informatics*, pages 167–178, 2006.

[182] Michael Seringhaus and Mark Gerstein. Publishing perishing? towards tomorrow's information architecture. *BMC Bioinformatics*, 8(1):17, 2007.

[183] S. Sild, U. Maran, A. Lomaka, and M. Karelson. Open computing Grid for molecular science and engineering. *Journal of Chemical Information and Modeling*, 46(3):953–959, 2006.

[184] Robert Stevens et al. Tambis: Transparent access to multiple bioinformatics information sources. *Bioinformatics*, 16(2):184–186, 2000.

[185] Robert D. Stevens, Alan J. Robinson, and Carole A. Goble. myGrid: personalised bioinformatics on the information grid. *Bioinformatics*, 19(suppl_1):i302–304, 2003.

[186] Chris F Taylor, Norman W Paton, Kathryn S Lilley, Pierre-Alain Binz, Randall K Julian, Andrew R Jones, Weimin Zhu, Rolf Apweiler, Ruedi Aebersold, Eric W Deutsch, Michael J Dunn, Albert J R Heck, Alexander Leitner, Marcus Macht, Matthias Mann, Lennart Martens, Thomas A Neubert, Scott D Patterson, Peipei Ping, Sean L Seymour, Puneet Souda, Akira Tsugita, Joel Vandekerckhove, Thomas M Vondriska, Julian P Whitelegge, Marc R Wilkins, Ioannnis Xenarios, John R Yates, and Henning Hermjakob. The minimum information about a proteomics experiment (miape). *Nat Biotech*, 25(8):887–893, 2007.

[187] G. WEIß. Agent orientation in software engineering. *The Knowledge Engineering Review*, 16(04):349–373, 2002.

[188] Mark D. Wilkinson and Matthew Links. BioMOBY: An open source biological web services proposal. *Brief Bioinform*, 3(4):331–341, 2002.

[189] N. Wirth. *Algorithms+ Data Structures= Programs*. Prentice Hall PTR Upper Saddle River, NJ, USA, 1978.

[190] Jonathan D. Wren. 404 not found: the stability and persistence of urls published in medline. *Bioinformatics*, 20(5):668–672, 2004.

[191] Jonathan D. Wren. Engineering in genomics: the emerging in-silico scientist; how text-based bioinformatics is bridging biology and artificial intelligence. *Engineering in Medicine and Biology Magazine, IEEE*, 23(2):87–93, 2004.

[192] Alexander C. Yu. Methods in biomedical ontology. *Journal of Biomedical Informatics*, 39(3):252–266, 2006.

[193] Enhydra JaWE, http://www.enhydra.org/workflow/jawe/index.html

[194] Xml process definition language, http://xml.coverpages.org/XPDL20010522.pdf

[195] I. Altintas, C. Berkley, E. Jaeger: Kepler: an extensible system for design and execution of scientific workflows. In: Proceedings 16th International Conference on Scientific and Statistical Database Management: 21-23 June 2004; Santorini Island; Greece (2004)

[196] E. Bartocci, F. Corradini, E. Merelli, L. Scortichini: BioWMS: a web-based Workflow Management System for bioinformatics. BMC Bioinformatics 8(Suppl 1) (2007), http://www.biomedcentral.com/1471-2105/8/S1/S2

[197] D. Cacciagrano, F. Corradini, E. Merelli, L. Vito, G. Romiti: Resourceome: a multilevel model and a Semantic Web tool for managing domain and operational knowledge. In: P. Dini, J. Hendler, J. Noll, (eds.) The Third International Conference on Advances in Semantic Processing (SEMAPRO 2009). IEEE Computer Society (2009)

[198] M. Cannataro,P.H. Guzzi, T. Mazza, P. Veltri: MS-Analyzer: Intelligent preprocessing, management, and data mining analysis of mass spectrometry data on the grid. International Conference on Semantics, Knowledge and Grid (2005).

[199] T. Carver, A. Bleasby: The design of jemboss: a graphical user interface to emboss. Bioinformatics 19(14) (Sep 2003).

[200] A. Cichocki: Migrating workflows and their transactional properties. Ph.D. thesis, University of Houston (1999).

[201] S. Cohen-Boulakia, S.B. Davidson, C. Froidevaux, Z. Lacroix, M.E. Vidal, : Path-based systems to guide scientists in the maze of biological data sources. J. Bioinformatics and Computational Biology 4(5), 1069-1096 (2006).

[202] F. Corradini, E. Merelli: Hermes: Agent-Based Middleware for Mobile Computing. In: SFM. pp. 234–270 (2005).

[203] I.D. Dinov, D. Rubin, W. Lorensen, J. Dugan, J. Ma, S. Murphy, B. Kirschner, W. Bug, M. Sherman, A. Floratos, D. Kennedy, H.V. Jagadish, J. Schmidt, B. Athey, A. Califano, M. Musen, R. Altman, R. Kikinis, I. Kohane, S. Delp, D.S. Parker, A.W. Toga: itools: a framework for classification, categorization and integration of computational biology resources. PLoS ONE 3(5) (2008).

[204] N. Guarino: Formal Ontology in Information Systems: Proceedings of the 1st International Conference June 6-8, 1998, Trento, Italy. IOS Press, Amsterdam, The Netherlands, The Netherlands (1998).

[205] E. Kawas, M. Senger, M.D. Wilkinson: Biomoby extensions to the taverna workflow management and enactment software. BMC Bioinformatics 7, 523 (2006), http://www.biomedcentral.com/1471-2105/7/523.

[206] A. Lee, S. Neuendorffer: MoML - a modeling markup language in xml - version 0.4. Tech. rep., University of California at Berkeley (2000).

[207] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. Glover, M. Pocock, A. Wipat, P. Li: Taverna: a tool for the composition and enactment of bioinformatics workflows. Bioinformatics 20(17) (Nov 2004).

[208] S.P. Shah, D.Y.M. He, J.N. Sawkins, J.C. Druce, G. Quon, D. Lett, G.X.Y. Zheng, T. Xu, B.F.F. Ouellette: Pegasys: Software for executing and integrating analyses of biological sequences. BMC Bioinformatics 5(40) (2004).

[209] R. Stevens, A. Robinson, C. Goble: myGrid: personalised bioinformatics on the information grid. Bioinformatics 19 Suppl 1 (2003).

[210] F. Tang, C. Chua, L. Ho, Y. Lim, P. Issac, A. Krishnan: Wildfire: distributed, grid-enabled workflow construction and execution. BMC Bioinformatics 6(1) (2005).

[211] M.D. Wilkinson, M. Links: Biomoby: An open source biological web services proposal. Briefings in Bioinformatics 3(4), 331–341 (2002).

[212] I.H. Witten, E. Frank: Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann, San Francisco (2005).

PhD in Information Science and Complex Systems
Collection of theses

XXI-09-1 Ezio Bartocci.*A Formal Framework for Modeling, Simulating and Analyzing Network of Excitable Cells.* January 2009.

XXI-09-2 Francesco De Angelis. *Interoperability in e-Government Services.* January 2009.

XXI-09-3 Oliviero Riganelli. *Online Public Service Delivery for Small and Medium Size Organizations.* January 2009.

XXII-10-4 Roberta Alfieri. *Computational approaches to model the cell cycle, a biological complex system.* January 2010.

XXII-10-5 Andrea Cristofaro. *Some constrained control problems with applications in industrial engineering.* January 2010.

XXII-10-6 Barbara Re. *Quality of (Digital) Services in e-Government.*January 2010.

XXII-10-7 Leonardo Vito. *A Semantic Framework for Declarative and Procedural Knowledge.* January 2010.