Model-based System Development
www.ifi.uio.no/inf5120

Part II
SOA – Service Oriented Architectures

Notes for Course material

“Model Based System Development”

INF5120 – Spring 2008

<table>
<thead>
<tr>
<th>Classification</th>
<th>Notes for course participants</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Responsible :</td>
<td>Arne-Jørgen Berre, SINTEF and University of Oslo</td>
</tr>
<tr>
<td>Authors :</td>
<td>Arne-Jørgen Berre, Brian Elvesæter</td>
</tr>
<tr>
<td>Contributors:</td>
<td>Projects INTEROP, ATHENA, SWING, SHAPE, COIN</td>
</tr>
<tr>
<td>Task</td>
<td>NF5120 Course notes</td>
</tr>
<tr>
<td>Status :</td>
<td>Version 1.00</td>
</tr>
<tr>
<td>Date :</td>
<td>April 28, 2008</td>
</tr>
</tbody>
</table>
# Table of Contents

Part II 1

Notes for Course material ........................................................................................................... 1

“Model Based System Development” .......................................................................................... 1

INF5120 – Spring 2008 .................................................................................................................. 1

Executive Summary ..................................................................................................................... 4

## I Service Oriented Computing ............................................................................................... 5

### 1.1 Introduction ....................................................................................................................... 5

### 1.2 Introduction to SOA ........................................................................................................... 5

### 1.3 Service-oriented model ....................................................................................................... 5

#### 1.3.1 Service provider ............................................................................................................ 6

#### 1.3.2 Service requester ............................................................................................................ 6

#### 1.3.3 Service broker ................................................................................................................ 6

### 1.4 Benefits of service-orientation ........................................................................................... 7

### 1.5 Interoperability analysis ..................................................................................................... 8

### 1.6 Extended Service Model .................................................................................................... 8

### 1.7 SOA platforms and Integrated execution infrastructure ..................................................... 9

### 1.8 Categorises of services ...................................................................................................... 10

### 1.9 Execution platforms .......................................................................................................... 11

### 1.10 Infrastructure Services ..................................................................................................... 12

#### 1.10.1 Types of services .......................................................................................................... 12

### 1.11 Web services ..................................................................................................................... 13

#### 1.11.1 Web Service Descriptions ............................................................................................. 13

#### 1.11.2 WSDL .......................................................................................................................... 14

#### 1.11.3 Representing WSDL using extended UML ................................................................. 15

#### 1.11.4 Representing Web services as a platform-independent model ...................................... 16

### 1.12 Web Service Compositions ............................................................................................... 16

#### 1.12.1 WS-BPEL ....................................................................................................................... 16

### 1.13 Technical aspects of eServices ......................................................................................... 17

#### 1.13.1 Web Services ................................................................................................................ 17

## II The Semantic Web and Web services – RDF, OWL and WSMO ......................................... 25

### II.1.1 The evolution of the Semantic Web ................................................................................ 25

### II.1.2 The Semantic Web ........................................................................................................ 25

### II.1.3 Resource Description Framework (RDF) ...................................................................... 26

### II.1.4 Ontologies ...................................................................................................................... 30

### II.1.5 Web Ontology Language (OWL) .................................................................................. 31

#### II.1.6 1.1 Overview of the OWL Metamodel .......................................................................... 31

#### II.1.7 Web Ontology Language for Web Services (OWL-S) ................................................ 33

#### II.1.8 OWL-S and WSMO .................................................................................................... 34

#### II.1.9 Web Service Modeling Framework (WSMF and WSMO) ........................................... 35

## III Agent-Oriented Computing ............................................................................................... 38

### III.1 Introduction ..................................................................................................................... 38

#### III.1.1 Historical context ........................................................................................................ 38

#### III.1.2 What is an agent? ......................................................................................................... 38

#### III.1.3 Types of agents .......................................................................................................... 39

#### III.1.4 Agents vs Objects ....................................................................................................... 39
III.1.5 What is a multi-agent system? ................................................................. 40
III.2 Multi Agent Systems ..................................................................................... 40
  III.2.1 Agent Societies ....................................................................................... 40
  III.2.2 Coordination in MAS ............................................................................ 41
  III.2.3 Negotiation ............................................................................................ 43
  III.2.4 Communication ..................................................................................... 43
III.3 Platform independent Model for Agents (PIM4Agents) ............................. 43
III.4 Conclusion on Agents ................................................................................. 45

IV Bibliography – References ............................................................................. 47
IV.1 Bibliography – Service Oriented Computing .............................................. 47
IV.2 Bibliography – Semantic Services ............................................................... 53
IV.3 Bibliography – Agent-oriented Computing ................................................... 55
Executive Summary

This document is part I in a series of four part of lecture notes for the course INF5120, Model Based System Development, for the spring of 2008, as follows.

Part I – MDE – Model Driven Engineering
Part II – SOA - Service Oriented Architectures
Part III – MDE for SOA
Part IV – MDI – Model Driven Interoperability

Part I focuses on Model Driven Engineering – with an introduction to principles of metamodeling and related standards and technologies, in particular related to MDA and Eclipse EMF. The relationship between UML profiles and Domain Specific Languages (DSL) is introduced, as well as an overview of various model transformation technologies including model-to-model with ATL and model-to-text with MOFScript. It is shown how method engineering can be supported by the OMG SPEM standard and the Eclipse EPF framework.

Part II focuses on SOA - Service Oriented Architectures – with a basis in concepts for service oriented computing, with a special emphasis on technologies for web services with XML, WSDL and BPEL. The basis technologies for the semantic web is also introduced with RDF and OWL, and semantic web services with OWL-S and WSMO. A last section presents agent-oriented computing with multi agent systems (MAS) and a platform independent model for agents (PIM4Agents).

Part III focuses on MDE for SOA - Model Driven Engineering for Service Oriented Architectures – and applies the principles of model driven engineering to service oriented architectures. The starting point for this approach is the COMET methodology used in previous INF5120 courses, this year enhanced to become COMET-S for Services through the use of new standard UML profiles and metamodels. The Business model uses in particular BMM (Business Motivation Metamodel, BPMN (Business Process Modeling Notation). The Requirements model supports mappings from use cases to services definitions. The service architecture model uses the new UPMS (UML Profile and Metamodel for Services). The platform specific model will vary depending on the target platform. The course has been using the JEE platform as reflected in the Oblig exercises in the course.

Part IV focuses on MDI - Model Driven Interoperability – and illustrates how a model driven approach can be applied to the problem domain of interoperability through the use of horizontal mappings and transformations. The approach to this is illustrated with the AIF (ATHENA Interoperability Framework) and the AIM (ATHENA Interoperability Methodology) and a set of articles on MDI.
I Service Oriented Computing

I.1 Introduction

The advent of Service Oriented paradigm for Enterprise Application integration has stimulated great expectation among the developers community. Service Oriented Architectures (SOA) emerged as an evolutionary step from Object and Component based approaches, with the promise to overcome the deficiencies due to which these solutions have fallen short in the past. Still the variety and diversity of implementations and interpretations of SOA causes controversy and scepticism among system architects and developers. Currently there not seems to be a single and consistent agreement on how SOA should be materialized. Moreover the vast amount of emerging standards, which many times expose overlapping applicability, makes it more difficult to understand and utilize the potentials of these technologies.

This part provides a snapshot of current trends, standards and implementations of Service oriented technology and pinpoints some interoperability issues. Toward this, it focuses on three major trends in Service oriented development namely Web Services, Grid Services and peer-to-peer (P2P) services. These three technologies provide the main body of the work that has been achieved the recent past years covering different approaches of distributed programming.

Web Services build upon XML standards to provide a coherent platform for building loosely coupled distributed applications. Grid Services on the other hand originate from the requirement of Grid Computing to standardize the interface mechanism for accessing distributed computational (grid) resources. P2P computing finally although has had many successes till now, still lacks consensus on how applications should be build and what semantics should support, thus rendering the notion of P2P service the vaguest of the three.

I.2 Introduction to SOA

According to W3C, a service-oriented architecture (SOA) specifies a set of components whose interfaces can be described, published, discovered and invoked over a network. SOA aims to promote software development in a way that leverages the construction of dynamic systems which can easily adapt to volatile environments and be easily maintained as well. The decoupling of system constituent parts enables the re-configuration of system components according to the end-user’s needs and the system’s environment. Furthermore, the use of widely accepted standards and protocols that are based on XML and operate above internet standards (HTTP, SMTP, etc) enhances interoperability.

Service-oriented development emerged as an evolution of the component-based development and among its goals is to support the loose coupling of system parts in a far better way than existing component-based technologies. The ramifications of service-oriented development can be observed both at the system and the business level. Having systems composed of services offered by various service providers provides the basis for supporting new business models, such as “virtual organizations”.

I.3 Service-oriented model

Any service-oriented environment is expected to support several basic activities:

1. Service creation
2. Service description
3. Service publishing to Intranet or Internet repositories for potential users to locate
4. Service discovery by potential users
5. Service invocation, binding
6. Service unpublishing in case it is no longer available or needed, or in case it has to be updated to satisfy new requirements.

In addition to these basic activities there are some other activities that need to take place in order to take full advantage of the service-oriented architecture. Such activities include service composition, management and monitoring, billing and security. However, we consider that the service model requires at least the following basic activities: describe, publish/unpublish/update, discover and invoke/bind, and contains 3 roles: service provider, service requester and service broker.

A common service-oriented model contains three roles service provider, service requester and service broker supporting the basic activities publish/unpublish/update, discover and invoke/bind. These roles and basic activities are illustrated in the figure below. It should be noted that in many scenarios the distinction between these roles may be blurred since an entity can play multiple roles, and roles can be interchangeable within the same scenario.

1.3.1 Service provider

A service provider is the party that provides software applications for specific needs as services. Service providers publish, unpublish and update their services so that they are available on the Internet. From a business perspective, this is the owner of the service. From an architectural perspective, this is the platform that holds the implementation of the service.

1.3.2 Service requester

A requester is the party that has a need that can be fulfilled by a service available on the Internet. From a business perspective, this is the business that requires certain function to be fulfilled. From an architectural perspective, this is the application that is looking for and invoking a service. A requester could be a human user accessing the service through a desktop or a wireless browser; it could be an application program; or it could be another service. A requester finds the required services via a service broker and binds to services via the service provider.

1.3.3 Service broker

A service broker provides a searchable repository of service descriptions where service providers publish their services and service requesters find services and obtain binding information for these services. It is like telephone yellow pages. Examples of service brokers are UDDI (Universal Description, Discovery, and Integration) and XMethods. In many cases the role of the service broker is not explicitly needed. Services can be discovered by marketing
channels or by referrals (e.g. a service provider referring to another one, or even a service consumer referring a provider to another provider).

I.4 Benefits of service-orientation

One might ask why we should focus on services for architecting the enterprise and its ICT support. What was wrong with object-orientation, component-based development, and business process engineering? Two of the major benefits are:

1. The service concept applies equally well to the business as it does to software applications. Add to that industry-wide support for Web services standards and for the first time in history, the convergence of the skill sets of the business analyst and the application developer. The analyst is able to specify service interface definitions and business processes, which can be used directly by the application developer as input for the implementation definition. From the point of view of the developer, this approach is sometimes referred to as “the outside-in approach” because it consists in going from the outside representation of a service to its internal representation. The “inside-out approach”, which derives the public interface of a service from its already-existing implementation, is to be avoided since it very often results in polluting the external representation of a service with unnecessary implementation details.

2. Service orientation offers a level of flexibility far exceeding that of component-based development (CBD). A component is built or bought once and integrated into an organisation’s application architecture. Once integrated, the component “disappears” inside the application and becomes “frozen”, i.e. it does not have an existence on its own (cannot be lookup up or accessed directly) and cannot be easily updated. A service is invoked dynamically when required, allowing providers to continuously improve their service and users to select the best available service at any one time. The focus on business processes in business process engineering (BPE) may have given a sense of flexibility, but ICT systems were never process-oriented. A change in the business processes of an organisation could require months to implement in the ERP systems supporting those processes.

Other benefits are listed below:

- Services reduce complexity by encapsulation: a service may be the aggregation of a number of other services. What is important is the type of behaviour a service provides, not how it is implemented. Encapsulation is key to coping with complexity, flexibility, scalability, and extensibility.

- Services provide the ‘units of business’ that represent value propositions within a value chain or within business processes; these services are a natural starting point for flexible outsourcing strategies. This is made possible because services are entirely self-describing and do not rely on any implicit or hidden assumption.

- Services promote interoperability by minimizing the requirements for shared understanding: a service description and a protocol of collaboration and negotiation are the only requirements for shared understanding between a service provider and a service user.

- Services enable interoperability of legacy applications: By allowing legacy applications to be exposed as services, a service-oriented architecture greatly facilitates a seamless integration between heterogeneous systems. New services can be created and dynamically published and discovered without disrupting the existing environment.
I.5 Interoperability analysis

The service-oriented model can be seen as a generic model which can be supported by many different technical platforms. There is a trend towards convergence with respect to all of the different architectures and technologies discussed in this report. Web service technologies such as WSDL and XML are being positioned as the best way of integrating systems implemented using different technologies. By adhering to a set of standards we achieve syntactical interoperability. However, interoperability issues related to semantic understanding of how to effectively and correctly integrate systems are still evident. Here we believe that model-driven development, with amongst other things model transformation technologies, and adaptive architectures such as agents and P2P, coupled with ontology approaches, will ensure better interoperability between software systems.

I.6 Extended Service Model

The basic set of operations do not suffice for the development of a system within a business context. The construction of a complex system that supports business processes requires enhancements of the basic set of operations that give added value to the basic model. These enhancements mechanisms that address the composition of services to more complex ones, and mechanisms that deal with issues like transactions and security. Furthermore, there is also a need for mechanisms that support the quality of service and semantic aspects.

Higher level mechanisms that can handle higher level issues such as monitoring and contracting are also required. This set of extensions can be organized into layers settled one above the other. Such an organization scheme was specified by Papazoglou and Georgakopoulos in [PG 2003] and is presented in Figure 1.

![Figure 1: Extended Service Oriented Architecture](image-url)
I.7 SOA platforms and Integrated execution infrastructure

The purpose of this document is to discuss a generic service-oriented integrated execution infrastructure that will be able to support different execution platforms. We refer to this integrated execution environment as the **Integrated Execution Infrastructure**.

The architecture of the Integrated Execution Infrastructure is shown in Figure 1.

- Infrastructure services
- Model Execution Platform
- Process Execution Platform
- Goal-oriented Adaptive Execution Platform
- Active Model Platform
- Adaptive Distributed Resource Management Platform

The *service wrapper*, the *evaluation & negotiation of available functionality* and the *enhanced service interconnection bus* are components of the service bus.*service wrapper evaluation & negotiation of available functionality enhanced service interconnection bus*

- The Composed Web Service Platform
- The Registry
- The Repository

![Figure 1: Architecture of the Integrated Execution Infrastructure](image)

The components of the environment could be further detailed using a modelling formalism such as UML. Figure 2 is a first attempt to identify the dependencies and interfaces provided...
by these components. The final specification of the Integrated Execution Infrastructure and its infrastructure services.

Figure 2: UML model of the core components of the Integrated Execution Infrastructure

1.8 Categorises of services

Services are an abstraction and an encapsulation of the functionality provided by an autonomous entity, e.g. a software component. Services are typically provided through well-defined interfaces or contracts that specify their usage and behaviour.

Of the many ways to categorise services implemented by information and communication technology (ICT) the following four have been chosen for the structure of the A5 and A6 work. The four categories reflect an operational picture of a running ICT environment within an enterprise.

1. **User-communication services** provide interaction front-ends which reflect different user views of the businesses within the enterprise. Various ICT services customised for different user groups are offered for managing these businesses. User-interaction services can be realised in task- or process-oriented execution languages, implemented as web or client applications, or developed and run as active knowledge models.

2. **Value services (business services)** are network-accessible software services that can be invoked in a service-oriented architecture and that provide value to the business under consideration. These services are also referred to as business services from an ICT point of view, since they provide core business functions within an enterprise. Value services can be realised in process- or agent-oriented execution languages, implemented as web or service components, or developed and run as active knowledge models.

3. **Infrastructure services** provide support functionality for managing deployed user-interaction services and business services in a heterogeneous and distributed ICT environment. These services, which also are referred to as middleware services, provide the information communication infrastructure of the enterprise.

4. **Registry/repository services** provide functionality for managing models, services, task executions and data. These services could be seen as a special kind of infrastructure services.
Figure 3 depicts how service realisations within these four categories are interconnected through a service bus which allows distributed services to interoperate. The service bus should be considered as an architectural pattern. The enterprise service bus [3] represents a similar approach defined by IBM.

**I.9 Execution platforms**

There exist several technical execution platforms that can be used to realise the service architecture outlined in Figure 3. In a heterogeneous environment, services may be deployed on different execution platforms, depending on which technical platform best support the realisation of the services in question. Figure 4 illustrates some execution platforms and the kinds of services they are capable of running. The illustrated platforms are categorised as user-oriented or business-oriented depending based on whether they primarily support user-interaction or business services.
Figure 4: Execution platforms

We might note that there are cases where a combination of these execution platforms is viewed as a single platform, e.g. the WebSphere Business Integration Server Foundation (WBI SF) [4] is a platform for J2EE, Web services and processes. The J2EE itself can be seen as an integrated platform comprising several execution platforms. Furthermore, some platforms may fall into both categories, e.g. the Active Model Platform where active models are used to represent both types of services.

While the complete picture was outlined in this section, the rest of the document describes different approaches for business service execution platforms in relation to the Integrated Execution Infrastructure. Section 7 introduces peer-to-peer (P2P) concepts that increase the adaptiveness and robustness of the underlying execution infrastructure.

I.10 Infrastructure Services

The infrastructure services, as the name suggests, provide the basic infrastructure. They provide support functionality for managing deployed user-interaction services and business services in a heterogeneous and distributed ICT environment. These services, which also are referred to as middleware services, provide the information communication infrastructure of the enterprise.

Most of the services in this category are things that are “natural” in software infrastructures today, while others are more advanced and are subject to research.

I.10.1 Types of services

- **Discovery services:** These services provide mechanisms that allow the finding of services based on their name or functionality. Typical examples of such services are UDDI for Web services, the CORBA Naming and Trader services in the world of CORBA.

- **Adaptive and dynamic composition of services:** Services that allow the dynamic and adaptive composition of a set of services to provide a “new” complete service is something that would be beneficial. Such adaptive and dynamic composition may be based on the use of semantic lookup and discovery with other mechanisms that actually compose the services so that they appear as one. Providing such support run-time is a challenge.

- **Semi-automated mapping of used terms:** Run-time mapping of terms corresponding to ontologies.

- **Provide & consume services:** The basic infrastructure needs to provide services that allow clients and services to communicate and to provide and consume services. CORBA and RMI are basic examples of such.

- **Search and semantic matchmaking:** In order to find services and match requests in a “smarter” way than just using names one can provide services that allow for the search and matchmaking based on semantics. This can involve the semantic annotation of services. This annotation will then be used by the infrastructure to provide matches for requests. The requests will then hold semantic information about the need.

- **Dynamic binding & invocation:** The discovery services allow clients or components to find services at run time. Having found the right service it is essential that one can bind to the service and invoke the operations it provides.

- **Brokering, mediation, and negotiation:** Intelligent brokering for negotiating service usage at run-time, automatically ensuring data mediation as part of the establishment of service contracts.

- **Peer-to-Peer business resource management:** Virtualization and de-central management of business objects, services, and processes
- **Self-descriptive properties for distributed enterprise networks**: Annotation of resources in a distributed network.
- **Intelligent agents**: Pro-active event-driven communication & situated semantic reasoning nodes support interoperability in de-centrally managed, distributed enterprise networks.
- **Monitoring**: Services to monitor service-level agreements, eContracts, QoS contracts etc.

  - **Testing and validation**: Services to test and validate software interactions on a model level.

### I.11 Web services

The service-oriented model can be seen as a generic model which can be supported by many different technical platforms. There is a trend towards convergence with respect to all of the different architectures and technologies discussed in this report. Web service technologies such as WSDL and XML are being positioned as the best way of integrating systems implemented using different technologies.

There are many definitions for what constitutes a Web service. In this report we adopt the definition given by the World Wide Web Consortium (W3C) [W3C 2004]:

> "A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."

Service-oriented architectures can be said to describe an architectural style towards system design. The Web service stack is an enabling technology that supports the design of a service-oriented model. Web services can be designed to adhere to the set of roles and operations specified by the service-oriented model and they have also managed to establish a standardized protocol stack. SOAP, WSDL and UDDI are the most well-known standards used for the execution of the basic set of operations i.e. bind, publish and find.

Web services aim to support enterprise application integration (EAI). They enable the integration and interoperability of heterogeneous systems and components which may be geographically dispersed.

In practise Web services have been regarded as web interfaces to components. However, Web services are not just interfaces to components. Web services intend to expose higher level business processes whereas components tend to expose lower level business objects and processes. Nevertheless, the level of granularity addressed by each technology is not the only difference among Web services and component technologies such as CORBA, EJBs and COM+.

#### I.11.1 Web Service Descriptions

The **Web Services Description Language** (WSDL) [20, 24] is one of the cornerstones of how Web services are used in the industry today. WSDL provides a convenient way to group together all the different pieces of metadata that together describe a service from both a functional (operations, data types and faults) and a non-functional (specifications supported, policies with respect to non-functional aspects such as security or reliability) view.
I.11.2 WSDL

WSDL 1.1 [20] is used today to describe and publish the public interfaces of Web services. WSDL is an XML format for describing network services as a set of endpoints operating on messages. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services).

A WSDL document is simply a set of definitions. There is a definitions element at the root, and definitions inside. Services are defined using six major elements:

- **Types**, which provides data type definitions used to describe the messages exchanged.
- **Message**, which represents an abstract definition of the data being transmitted. A message consists of logical parts, each of which is associated with a definition within some type system.
- **Port type**, which is a set of abstract operations. Each operation refers to an input message and output messages.
- **Binding**, which specifies concrete protocol and data format specifications for the operations and messages defined by a particular port type.
- **Port**, which specifies an address for a binding, thus defining a single communication endpoint.
- **Service**, which is used to aggregate a set of related ports.

Figure 7 depicts the WSDL 1.1 metamodel which shows the relationship between these elements. A detailed description of the metamodel is given in Annex [2]. The annex also gives a good insight into the metamodel of WSDL 2.0 [24] which will eventually replace WSDL 1.1.

Figure 7: WSDL 1.1 metamodel
1.11.3 Representing WSDL using extended UML

The specification framework described in this report promotes visual models such as UML to describe Web services. When modeling Web services in such an approach a WSDL profile is defined with special constructions for the WSDL language. This is done by using the UML extension mechanisms to:

- define stereotypes for the specific WSDL and XML Schema types such as `<<wsdl:portType>>`, `<<wsdl:service>>` and `<<xs:complexType>>`. `wsdl:portTypewsdl:servicexs:complexType`
- define taggedValues for representing bindings and access URLs.

Examples of proposed UML extensions for WSDL are described in [25] and [26]. Figure 8 shows a WSDL-dependent UML model of the Google service according to the extensions described in [25]. The `GoogleSearchService` represents the Web service. It contains a `GoogleSearchPort` with access URL, which in turn refers to the concrete binding represented by `GoogleSearchBinding`. `GoogleSearchBinding` defines the transport protocol to be used with its associated tagged values (`binding`, `style` and `transport`) and realizes the `GoogleSearchPortType`. The `GoogleSearchPortType` defines the three available operations. Note that each of these operations has exactly one input parameter and one output parameter that directly reflect the WSDL programming model of sending an input message and receiving an output message as answer. To retrieve the actual details of these messages, it is necessary to look at the `<<wsdl:message>>` stereotyped classes. `GoogleSearchServiceGoogleSearchPortGoogleSearchBindingGoogleSearchPortTypeGoogleSearchPortType`

Figure 8: Example of a specific model for WSDL
I.11.4 Representing Web services as a platform-independent model
You could also envision a platform-independent modelling approach that abstracts from the
details of the Web services platform. There are two major advantages of using platform-
independent UML models:

- The same model may be used as a basis for conversion to more than one platform.
- The modeller is not disturbed by technical details when designing a conceptual model.

I.12 Web Service Compositions
Web service composition can encompass a lot of different approaches. The evolution of the
WS-* stack favoured control flow-based approaches such as the use of business process
engines for orchestrating service invocations. In the last years, the industry seems to have
standardized on WS-BPEL orchestrating [22] as the specification of choice for describing
business processes that feature both humans and services. In addition to WS-BPEL we have
also investigated the ACE-GIS Web service composition profile described in [27] and JACK
BDI [28] as an agent-oriented approach to service composition. Lastly, service composition
can also be done according to the planning from first principle approach.

I.12.1 WS-BPEL
The Business Process Execution Language for Web Services (WS-BPEL) Business Process
Execution Language for Web Services [22] provides a language for the formal specification of
business processes and business interaction protocols. By doing so, it extends the Web services
interaction model and enables it to support business transactions. WS-BPEL defines an
interoperable integration model that should facilitate the expansion of automated process
integration in both the intra-corporate and the business-to-business spaces.

The language covers interoperability through concepts for interaction with partners based on
Web services, supporting conversations of process instances with partners, and specifying
business protocols through the external behaviour of partners. Partner interaction is based on
the notion of peer-to-peer interaction between Web services. WS-BPEL introduces concepts to
express the peer-to-peer conversational relationships between services.
I.13 Technical aspects of eServices

I.13.1 Web Services

By definition, a Web service is a self-content, self-describing, loosely coupled, reusable software component that can be published, discovered/located, and invoked via Internet protocols. A Web service is agnostic of operating systems, programming models, and languages. It provides an interface describing how other systems can interact with it using messages. Web services perform functions, which can be anything from simple requests (transformation, storage and/or retrieval of data) to complicated business processes (aggregation, composition, orchestration).

The basic technological infrastructure for Web services is structured around XML-based standards and Internet protocols. These standards provide building blocks for service description, discovery, and interaction. Web service technologies have clearly influenced positively the development of integrated systems by providing programmatic access to Web services. They are evolving toward being able to solve critical integration issues including security, transactions, collaborative processes management, semantic aspects, and seamless integration with existing middleware infrastructures.

Figure 2 provides an overview of existing Web service specifications organized in terms of the issues that they address.

![Figure 2: Overview of the Web services stack]

**Composition/Choreography**
- BPEL4WS, WSCI
- ebXML BPSS

**Description**
- WSDL, WS-Policy
- ebXML CPP/CPA

**Messaging**
- SOAP, WS-Security, WS-ReliableMessaging, WS-Routing
- ebXML Messaging Service

**Transport**
- HTTP, HTTPS, SMTP

**Transactions**
- WS-Coordination/Transaction
- OASIS BTP

**Advertisement/Discovery**
- UDDI, WS-Inspection
- ebXML Registry

**Format and Encoding**
- Unicode, XML, XML Schema

I.13.1.1 Web Service Description / Publication

In this section, we first describe the use of SOAP (Simple Object Access Protocol), WSDL (Web Services Description Language), and UDDI (Universal Description, Discovery, and Integration) as building blocks for Web services-enabled applications [Alonso et al. 2003, Curbera et al. 2002]. Then, we give a brief overview of other Web service standards.
**Simple Object Access Protocol (SOAP)**

SOAP provides an XML-based protocol for structured message exchanges. It relies on existing transport protocols such as HTTP and MQSeries. SOAP features document-based communication among Web services. Document-based communication allows the integration of loosely coupled services. A SOAP message contains two parts: the header and the body. The header includes information such as intended purpose (e.g., service invocation, invocation results), sender's credentials, response type, and so on. The body contains an XML representation of a service invocation request (i.e., name of operation to be invoked, values of input parameters) or response (i.e., results of service invocation). SOAP implementations exist for several programming languages including Java and C. SOAP implementations provide mappings between SOAP messages and formats understood by service implementations (e.g., Java classes). SOAP implementations typically automatically generate the SOAP header, and provide mappings between the contents of SOAP message bodies, and data structures in the host language (e.g., Java objects).

**Web Service Description Language (WSDL)**

WSDL [W3 Consortium 2001a.] is an XML-based language for describing functional properties of Web services. It aims at providing self-describing XML-based service definitions that applications, as well as people, can easily understand. In WSDL, a service consists of a collection of message exchange end points. An end point contains an abstract description of a service interface and implementation binding. The abstract description of a service contains: (i) definitions of messages which are consumed and generated by the service (i.e., input and output messages) and (ii) signatures of service operations. The implementation binding provides a means to map abstract operations to concrete service implementations. It essentially contains information about the location of a binding, the communication protocol to use (e.g., SOAP over HTTP) for exchanging messages with the service, and mappings between the abstract description of a service and the underlying communication protocol message types (i.e., how interactions with service occur over SOAP).

**I.13.1.2 Web Service Discovery**

**Universal Description Discovery and Integration (UDDI)**

UDDI is a specification of an XML-based registry for Web services. It defines an interface for advertising and discovering Web services. The UDDI information model, defined through an XML schema, identifies three types of information: white pages, yellow pages, and green pages.

White pages contain general information such as business name (i.e., service provider's name) and contact information (e.g., provider's phone numbers). Yellow pages contain meta-data that can used to effectively locate businesses and services based on classification schemes. For instance, UDDI uses the following standard taxonomies to facilitate businesses/services discovery: NAICS (North American Industry Classification System), UNSPSC (Universal Standard Products and Services Code System), and ISO 3166 (The ISO geographical classification system).

The green pages contain service access information including service descriptions and binding templates. A binding template represents a service end point (i.e., a service access interface). It refers to an entity called the tModel. A tModel describes the compliance of a service with a technical specification (e.g., WDSL document, RMI interface, CORBA IDL). For instance, a WSDL document can be registered as a tModel in the UDDI registry and used in the description of a WSDL-complaint service end point to provide access to service operations. The current stable version of UDDI is version 3.
I.13.1.3 Web Service Composition (Coordination / Transactions)

Web service composition refers to the development of new Web services by interconnecting existing ones according to some business logic, expressed (for example) as a business process model. For example, a composite Web service for travel arrangement could bring together a number of Web services for flight booking, accommodation booking, attractions search, car rental, events booking, etc. in order to provide a “one-stop shop” for its users. Web service composition is a key element of the Web services paradigm, as it provides a means to integrate heterogeneous enterprise applications and to realize business-to-business collaborations.

Orchestration deals with implementation management (what happens behind interfaces, i.e. process execution). This means that orchestration is a private process, controlled by one party, and it defines steps of an executable workflow. Propositions such BPEL and BPML are clearly at this level. Choreography is more about what happens between interfaces. It can involve static or dynamically negotiated protocols. In this sense, choreography is a public abstract process, where conversations are made up of equals, and they define sequences of observable messages [Peltz 2003]. In this section, we describe a representative sample of ongoing efforts in service composition, orchestration and choreography standardization.

Business Process Execution Language for Web Services (WS-BPEL)
The Business Process Execution Language for Web Services (WS-BPEL [Thatte2003]) is a language to model Web service based business processes. The core concept is the representation of peer-to-peer interactions between a process and its partners using Web services and an XML-based grammar. It is built on top of WSDL (both the processes and partners are modelled as WSDL services).

BPEL4WS – BPEL for short – is a language based on XML that allows for controlling the process flow (states, coordination and exceptions handling) of a set of collaborating Web services. For that, it defines interactions that exist within and between organisation processes. The language uses either a graph based or algebraic representation, and offers the ability to manage both abstract and executable processes. It provides constructs to handle long running transactions (LRTs), compensation and exception using related standards WS-AtomicTransaction, WS-BusinessActivity and WS-Coordination.

BPEL offers an interesting feature that allows having an independent representation of the interactions between the partners. The interaction protocols are called abstract processes, and they are specified in business protocols. This concept separates the external behaviour of the partners (public and visible message exchange behaviour) from their private internal behaviour and implementation. Executable processes are represented using the BPEL meta-model to model the actual behaviour using the three classical flows: the control flow, the data flow and the transactional flow. It also includes support for the message flow.

As in traditional flow models, the control flow defines the execution flow as a directed acyclic graph. The language is designed to combine the block oriented notation and the graph oriented notation. It contains powerful constructors for modeling structured activities: aggregation, branching, concurrency, loops, exceptions, compensations, and time constraints. Links are used to define control dependencies between two block definitions: a source activity and a target activity. Activities can be grouped within a scope, and associated with a scope are three types of handlers: fault handlers, compensation handlers, and event handlers. When an error occurs, the normal processing is terminated and the control is transferred to the corresponding fault handler. Then, a process is terminated when it completes normally, when a terminate activity is called (abnormal termination), when a fault reaches the process scope or when a compensation handler is called.

BPEL basic activities are handled by three types of messages: <invoke> to invoke an operation on a partner, <receive> to receive an invocation from a partner and <reply> to send reply message in
partner invocation. For each message, one must associate a partner, which prohibits the message exchange between two internal components for instance. Furthermore, there is no ability to associate a timeout to the <invoke> activity. This can block the system if no response is returned.

Data flow management is ensured using scoped variables. Input and output of activities are kept in variables, and data is transferred between two (or more) activities thanks to shared data spaces that are persistent across Web services and global to one scope. The <assign> activity is used to copy data from one variable to another.

BPEL also proposes a compensation protocol to handle the transaction flow, and particularly long running transactions. One can define either a fault handler or a compensation handler. Handlers are associated with a scope, and a fault handler defines alternate execution paths within the scope, while the compensation handler is used to reverse the work performed by an already completed scope.

On collaboration aspects, BPEL is able to model several types of inter-actions from simple stateless interactions to stateful, long running, and asynchronous interactions. Partner Link Types are used to model partner relationships and correlation sets represent the conversations, maintaining the state of the interaction. The choreography of the collaborative business processes is defined as an abstract process.

**Web Service Choreography Interface (WSCI)**

The WSCI specification [BEA et al. 2002] proposed by Sun, SAP, BEA and Intalio, is an XML-based language for describing the observable behaviour of a Web service during a message exchange in the context of a collaborative business process. This language gives the ability for describing the sequence of Web service invocations, i.e. the conditions under which an operation can be invoked. The specification is mainly concerned with public message exchanges among Web services and supports message correlation, sequencing rules, exception handling and transactions.

As WSCI defines the flow of messages exchanged by a stateful Web service describing its observable behaviour, it does not directly address the issue of supporting executable business processes, as BPEL does. A WSCI document defines only one partner’s participation in a message exchange, including the specification of temporal constraints and logical dependencies using constructs for expressing the flow chart and conditional correlation. Thus, other Web services can unambiguously interact with it according to the intended collaboration. This means that a collaboration is described using a set of WSCI documents, one for each partner. There is neither private workflow nor global cooperation business process.

A WSCI interface is built on top of a WSDL interface which defines stateless operations that are supplied by a Web service. Therefore, a WSCI interface can be regarded as an augmented WSDL interface that includes operation abstraction, simple sequencing (call, delay, empty, fault, and spawn), message correlation and properties based on message contents. An action in WSCI maps to a WSDL operation and a role to perform it. This corresponds to a basic activity in BPEL. A second level aims at defining exceptions, transactions and compensating transactions, and offers rich sequencing rules: loops, branches, joins and nested activities (all, choice, foreach, sequence, switch, until, and while). Thus, a stateless WSDL description can be transformed in a stateful message exchange using WSCI. This corresponds to structured activities in BPEL. However, WSCI does not define a transactional protocol, but only expose the transactional capacities of Web services in the context of a collaboration. An extensibility feature of WSCI suggests using RDF to annotate a WSCI interface definition with additional semantics.

**Business Process Management Language (BPML)**

BPML [BPMI2002] from BPMI (Business Process Management Initiative) is a language that provides an abstract model and grammar for describing business processes. BPML allows both
abstract and executable processes, Web services orchestration and multi-partners collaboration choreography to be defined. BPML can be used to develop a private implementation of already existing WSCI collaborations. In fact, BPML is more or less at the same level as BPEL and can be used to define a series of activities a business process performs using a block-structured language. An activity is a component performing a specific function and atomic activities can be composed into complex activities. A BPML specification extends WSCI activity types adding assign, raise and synch. A process is a complex activity which can be invoked by other processes.

The language includes three process types: nested processes (a process that is defined to execute within another process, such as WfMC nested processes), exception processes to handle exceptional conditions and compensation processes to support compensation logic. An activity executes within a context which is similar to a BPEL scope. A context is an environment for the execution which allows two activities to (1) define a common behaviour e.g. coordination of the execution using signals (such as raise or synchronize signal) and (2) share properties (data flow exchange between activities). A context is transmitted from a parent to a child and it can be nested. The language includes a logical process model to express concurrency, loops or dynamic tasks. The process instantiation is based either on the receipt of a message, either in response to a system event and scheduling, or invoked from an activity (called or spawned).

**ebXML and the Business Process Specification Schema (BPSS)**

ebXML (Electronic Business using eXtensible Markup Language) is a global electronic business standard envisioned to define an XML based framework that will allow businesses to find each other and conduct business using well-defined messages and standard business processes [ebXML 2007]. The ebXML Business Process Specification Schema (BPSS) is part of the ebXML framework B2B suite of specifications aimed at representing models for collaborating e-business public processes. Using XML syntax, BPSS describes public business processes as collaborations between roles, where each role is an abstraction of a trading partner. It also defines relationships and responsibilities. Being abstract, a definition is reusable as it only defines the exchange of information between two or more partners – business documents and business signals. A business process includes business collaborations, which are a choreographed set of business transaction activities. There are two types of collaborations: binary collaborations between two roles, and multi-party collaborations between three or more roles. Multi-party collaborations are decomposed into binary collaborations.

BPSS does not use WSDL to describe services. Instead, BPSS process models contain service interface descriptions and capabilities for each role. A partner can declare its support for a given role (services interfaces) in a ebXML CPP – Collaboration Protocol Profile which serves two purposes. Firstly, it supports messaging exchange capabilities i.e. specific asynchronous request and response operations, each with a defined message content. ebXML uses SOAP with attachments to manage XML document types and MIME attachments. Secondly, it supports generic acknowledgement and exception messages. This allows for reliable and secure messaging service management e.g. authorization, encryption, certification and delivery.

In BPSS, there is no explicit support for describing how data flows between transactions. Instead, BPSS assigns a public control flow (based on UML activity graph semantics) to each binary collaboration. The control flow describes the sequencing of business transactions between the two roles. It can specify sequential, parallel, and conditional execution of business transactions.

In addition, BPSS supports a long-running business transaction model based on transaction patterns. A business transaction consists of a request and an optional response. Each request or response may require a receipt acknowledgement. Time constraints can be applied on messages and/or acknowledgements. If a transaction fails, the opposite side is notified so that both sides can decide on the actions that need to be taken.
Transactions are not nested and there is no support for specifying compensating transactions so a business transaction either succeeds or fails completely. BPSS handles exceptions by defining a number of possible exceptions and prescribing how these are communicated and how they affect the state of the transaction. Then, BPSS provides explicit support for specifying quality-of-service semantics for transactions such as authentication, acknowledgements, non-repudiation, and timeouts.

**WSCL**

Web Services Conversation Language (WSCL) is a proposition from Hewlett-Packard related to previous work on e-Speak. WSCL is an XML vocabulary that offers the ability to define the external behaviour of the services by specifying the business level conversations between Web services. One of the main design goals of WSCL is simplicity. As such, WSCL provides a minimal set of concepts necessary for specifying the conversations. A WSCL document specifies three parts: the XML schemas that correspond to the XML documents being exchanged as part of the conversation, the conversation description (order in which documents are exchanged), and the description of the transactions from one conversation to another. In contrast with BPEL or BPML, WSCL does not specify how the content of the messages that are exchanged is created. The specification states that typically the conversation description is provided from the perspective of the service provider. This can also be used to determine the conversation from the perspective of the user. Although the conversation is defined from the service provider's perspective, WSCL separates the conversational logic from the application logic or the implementation aspects of the service.

**WS-Coordination, WS-AtomicTransaction and WS-BusinessActivity**

Since ACID transactions are not suitable for loosely-coupled environments like the Web, OASIS BTP and WS-AtomicTransaction/WS-BusinessActivity/WS-Coordination are proposals for dealing with specific WS aspects of coordination.

WS-Coordination [Microsoft et al. 2003a.] defines a generic framework that can support various coordination protocols. Each protocol is intended to coordinate a different role that a Web service plays in the activity. Some examples of coordination protocols are Completion (a single participant tells the Coordinator to either try to commit the transaction or force a rollback), 2PC – Two Phase Commit (a participant such as a resource manager registers for this, so that the Coordinator can manage a commit/abort decision across all resource managers), and PhazeZero (the Coordinator notifies a participant just before a 2PC protocol begins).

A Coordination Service propagates and coordinates activities between services. The messages exchanged between participants carry a Coordination Context that contains critical information for linking the various activities within the protocol. A Coordination Service consists of several components: an Activation Service that allows a Coordination Context to be created, a Registration Service that allows a Web service to register its participation in a Coordination Protocol, and a set of Coordination Protocol Services for each supported Coordination Type (e.g., Completion, 2PC).

WS-AtomicTransaction and WS-BusinessActivity [Microsoft and al. 2003b, Microsoft and al. 2004] are two specifications released in September 2003 and January 2004 by Microsoft, IBM and BEA Systems. It specifies transactional properties of Web Services independently of coordination aspects. An Atomic Transaction is used to coordinate activities having a short duration and executed within limited trust. It has the classical atomicity property (“all or nothing” behaviour) from ACID properties. A Business Activity provides flexible transaction properties (relaxing Isolation and Atomicity) and is used to coordinate activities that are long in duration and aimed at applying business logic to handle business exceptions. Actions are applied immediately and are permanent. This is because the long duration nature of the activities prohibits locking of data
resources. A Web Service application can include both Atomic Transactions and Business Activities.

1.13.1.4 Web Service Security, Reliability & Policy

**WS-Security**

WS-Security [Kaler and al. 2002] aims at integrating several existing security-related technologies in a coherent model and providing an XML syntax for this model. This is achieved by defining header elements to be included in SOAP messages. WS-Security does not provide a complete security framework for Web services; however it does provide mechanisms for ensuring single-message security within SOAP.

Three mechanisms are supported in the current specification:

- Propagation of unsigned and signed security tokens in both text and binary formats. Examples of unsigned security tokens include usernames and passwords, while signed tokens include X.509 certificates and Kerberos tickets. Recent extensions provide support for SAML (Security Assertions Markup Language) assertions and XrML (eXtensible rights Markup Language) licenses.
- Message integrity of SOAP messages is provided using the XML Signature specification together in conjunction with security tokens.
- Message confidentiality using XML Encryption specification in conjunction with the security tokens.

**WS-Reliability**

WS-Reliability [Evans and al. 2003] and WS-ReliableMessaging [Langworthy 2003] are two competing standards which aim at defining SOAP header elements for addressing three issues:

- Guaranteed message delivery through retries
- At most once message delivery through duplicate elimination
- Guaranteed message ordering by attaching sequence numbers to messages within a message group.

**WS-Policy**

WS-Policy [HK 2002] provides a framework with an XML-syntax for defining capabilities and requirements of Web services in the form of policy assertions. Policy assertions are statements about an XML element or a Web service description that provide indications regarding the text encoding and natural language used in an XML element, the version of a given standard specification used by a Web service, and the mechanisms used for Web service security (e.g. authentication scheme) with reference to the WS-Security specification (see above). A related specification, namely WS-PolicyAttachment, provides a mechanism for associating policy assertions expressed in WS-Policy to WSDL descriptions and UDDI entries.

1.13.1.5 Web Service Billing

Web service billing concerns service brokers and service providers. Service brokers create and manage taxonomies, register services and offer rapid lookup for services and companies. They may also offer value-added information for services, such as statistical information for the service usage and QoS data. One key issue for brokers is how to make profit out of these provided services. On the other hand, service providers need to charge the users of a Web Service. Unlike today’s software packages, which are typically made available through licenses based on a per-user or site pricing model, Web Services will be most likely be licensed according to a “pay-as-you-go” subscription based pricing model. This has the potential to significantly reduce the IT-related costs.
for supporting software within an enterprise. Rather than having to buy monolithic software packages, wherein users might only use a fraction of the whole package, the Web Service model allows users to pick and choose the precise bits of functionality for the time interval that are needed to perform a specific task. This means that the use of the service should be monitored and billed. Standards do not provide any answers to these questions and research results are still minimal. An initial research contribution is the meter and accounting model described in [EK 2001] that operates on a base assumption that services with a high degree of value are contracted via Service Level Agreements (SLA’s) or their equivalent. This implies that both the provider and the requester agree to the contract. This model has been developed as a service itself. The service stores the contract details, such as type of contract (long term, ad hoc, etc.), start and expiration dates of contract, the time model to be used (everyday, once weekly, etc.) and security details. A number of alternative business models such as pay-per-click/fee-for-use model, subscription and lease model can be used by the meter. HP Web Services platform continuously tracks service usage, and allows the service provider to bill only for the time the service was actually used. None of the two previous solutions fully address the semantic aspects of billing.
II The Semantic Web and Web services – RDF, OWL and WSMO

II.1.1 The evolution of the Semantic Web
The semantic web has evolved from its initial architecture with RDF in 2001 to an extended set of standards with RDF-S, OWL, SPARQL and RIF in 2006 and onwards.

II.1.2 The Semantic Web
The main challenge of the Semantic Web is to introduce a language for describing web resources, so also Web Services.
This section aims giving a simple overview on the present semantic languages scenario. All the languages, as shown in the figure below, use XML as common syntax.

XML-Based Ontology Exchange Language (XOL)
XOL is a language for ontology exchange, developed by the US bioinformatics community to facilitate the creation of shared ontologies. Originally the language was developed for
bioinformatics use and now it is intended to get used as an intermediate language on transferring ontologies between different applications and tools.

The XOL syntax is based on XML and the semantic is based on OKBC-Lite (Open Knowledge Base Connectivity), a simplified form of an API for accessing knowledge representation systems such as object databases, relational databases and so on. However, XOL is not intended to be used for the development of ontologies but only to integrate different programs and databases.

**Simple HTML Ontology Extension (SHOE)**

SHOE language had developed at the University of Maryland as an extension of HTML, to include a machine readable semantic knowledge in the Web documents. Recent releases has adapted the SHOE syntax to XML.

SHOE adds new tags to the HTML standard useful for declaring and extending ontology description, defining classification rules, relationship rules, inference rules, instance and it includes also data type definition.

SHOE aims to raise an agents architecture in which the agents classify the web contents with semantic constructors. It prevents the possibility of logical contradictions permitting only assertions, no retractions and negations, and relations with only one value or a fixed set of values.

**Ontology Markup Language (OML)**

OML is a partially XML serialization of SHOE based on conceptual graph. It is divided in four different levels: Core is related to logical aspects of the language and it is included in the others levels, Simple is a direct mapping to RDF language, Abbreviated includes conceptual graphs features and Standard is the most comprehensive and expressive level of the language.

Each of these versions is designed for a specific purpose, from the most simple to the most expressive and natural. The OML earlier releases were basically translations of the SHOE language into XML but the news versions are compatible with RDF language and RDF Schema, and include expressiveness of conceptual graphs.

**II.1.3 Resource Description Framework (RDF)**

RDF (Resource Description Framework) is a framework that enables describing and interchanging metadata with a few simple constructors. It has been extended from a description of a Schema with which it composes RDF(S). RDF(S) is the most important semantic language and it is the baseline for the new ones, such as DAML, OIL and OWL.

RDF provides a model for metadata. It’s a complement of XML enabling knowledge-base applications and ontology languages. It’s based on the idea of identifying “things”, such as Web Services, in terms of properties and related values.

The most important feature of RDF is simplicity, so that provides a very well understood metadata structure for information modeling, based on three assumptions:

- Resource: Anything that can have an URI such as a web site or a Web Service.
- Property: A property of the thing that the statement describes.
- Statement: A link between a resource and its property.

Practically each statement is composed of three terms:

- Subject: An URI that indicates a resource.
- Predicate: The property of the subject.
- Object: The value of the property.

It can describe a single statement with a graphical representation or with XML.
The model language is based on three simple artifacts. The resources are represented from an oval, a value from a rectangle and the predicates from an arrow that links the subject with the object. RDF Schema extends the language with a new vocabulary and defines a semantic extension of the basic language that provides the necessary instruments to describe groups of related resources and relationship between resources.

The RDF Schema is very important for the understanding of every semantic language descending from RDF, such as DAML and OWL, because it introduces many capabilities used for describing ontologies. It adds some very important basic concepts such as properties, classes, sub-classes, data types, constraints, containers and collections.

The first level at which a concrete data model is defined on XML is the Resource Description Framework (RDF). Actually, RDF as a data model is independent of XML, but we consider it as a layer extending the XML because of the widely practiced XML serialization of RDF in semantic web applications.

The basic structure of RDF is a triple consisting of two nodes and a connecting edge. These basic elements are all kinds of RDF resources, and can be variously described as <things> <properties> <values> (Manola & Miller, 2004), <object> <attribute> <value> (Broekstra, Kampman, & van Harmelen, 2003), or <subject> <predicate> <object> (Powers, 2003). This relatively simple basic model has several features that make it a powerful data model for integrating data in dispersed locations. The following are quotes from (Butler, 2002).

1. RDF is based on triples, in contrast to arguably the most commonly encountered metadata format, attribute-value pairs. The advantage of using triples is that this makes the subject of the attribute value pair explicit.
2. RDF distinguishes between resources and properties that are globally qualified, i.e., are associated with a URI, and those that are locally qualified. The advantage of a globally qualified resource or property is it can be distinguished from other resources or properties in different vocabularies that share the same fragment name. This is also possible in XML via XML namespaces, but many XML documents do not use this.
3. As a result of the first two properties, RDF can be used to make statements about resources, including documents on the Web, such as relating one URI to another.
4. It is easy to encode graphs using RDF as it is based on triples, whereas XML documents are trees so encoding graphs is more complicated and can be done in several different ways.
5. RDF has an explicit interpretation or model theory; there is an explicit formal interpretation of an RDF model (Hayes, 2004). XML documents also have interpretations, but they are often implicit in the processor or parser associated with that particular type of XML document.
Interestingly, in spite of the seeming usefulness of RDF, there is relatively slow adoption of RDF compared with XML (Batzarov, 2004). There are many possible reasons for this slow adoption. (Daconta, Obrst, & Smith, 2003) take an optimistic position and attribute the long lead-in time to poor tutorials, minimal tool support, and poor demonstration applications, arguing that once the practical limitations have been overcome, adoption will grow rapidly. However, we must not ignore the presence of dissatisfaction with RDF in both practitioner and research communities. Some of the challenges for RDF in light of this dissatisfaction are as follows:

1. RDF / XML integration needs improvement. For instance it is not possible to validate RDF embedded in other XML (or XHTML) documents because of RDF's open grammar. Without an agreed process of validation, RDF/XML documents can contain unchecked errors which make the goal of shared, trusted knowledge problematical. The W3C RDF Working Group is working on solutions for successfully embedding RDF within XML, and tools such as SMORE already mix RDF and XML. But it remains to be seen if solutions can be found for the problems of validation.

2. The RDF data model can be complex and confusing because it mixes metaphors and introduces new concepts that can be tricky to model. For instance the standard notion of RDF as composed of subject-predicate-object is linguistically derived, but its relationship to concepts in other representations is somewhat unclear, e.g., class-property-value (object-oriented), node-edge-node (graph theory), source-link-destination (web link), entity-relation-entity (database), and can cause confusion. One of the tricky constructs is reification, which introduces an unproven modeling construct that is foreign to most data modeling communities. Reification can cause confusion because it can be used to arbitrarily nest statements, possibly negating the stated truth value of statements (Daconta, Obrst, & Smith, 2003).

3. The RDF/XML serialization is confusing and difficult to work with, especially in the absence of proper tool support. The striped syntax (Brickley, 2001) can make it difficult to understand the proper interpretation of statements. For instance it is often impossible to tell whether an XML element in the RDF serialization represents an edge, or a node.

4. The RDF/XML serialization makes it difficult to perform many tasks that are significantly easier with XML alone, or with more standard data models.

Clearly there is a great deal of work to be done in establishing RDF as a core technology that adds value to the widely adopted XML syntax alone. One possible avenue is the use of RDF as a foundation layer for Ontologies. RDF makes it relatively simple to express higher level ontological constructs. Implementing ontologies in XML alone is tricky because the validation of XML requires an XML Schema which is not always possible. For example in describing a procedure for translating an ontology into an XML Schema, (Klein et al., 2003) note several important problems. First, superclass/subclass inheritance is problematic and has to be overcome with artificial workarounds in the XML specification, and defining multiple inheritance is not possible at all in XML/S. Second, the possibility of fully automating the translation process is questionable, limiting its use for large ontologies.

In order to use RDF as a means of representing knowledge it is necessary to enrich the language in ways that fixes the interpretation of parts of the language. As described thus far, RDF does not impose any interpretation on the kinds of resources involved in a statement beyond the roles of subject, predicate and object. It has no way of imposing some sort of agreed meaning on the roles, or the relationships between them. The RDF schema is a way of imposing a simple ontology on the RDF framework by introducing a system of simple types.

**RDF Schema**

We have seen that RDF provides a means to relate resources to one another in a graph based formalism connecting subjects to objects via predicates. The RDF schema, (RDF/S) provides modeling primitives that can be used to capture basic semantics in a domain neutral way. That is,
RDF/S specifies metadata that is applicable to the entities and their properties in all domains. The metadata then serves as a standard model by which RDF tools can operate on specific domain models, since the RDF/S meta-model elements will have a fixed semantics in all domain models. The RDF/S elements are shown below:

### RDF/S classes

<table>
<thead>
<tr>
<th>Class name</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdfs:Resource</td>
<td>The class resource, everything.</td>
</tr>
<tr>
<td>rdfs:Literal</td>
<td>The class of literal values, e.g. textual strings and integers.</td>
</tr>
<tr>
<td>rdfs:Class</td>
<td>The class of classes.</td>
</tr>
<tr>
<td>rdfs:Datatype</td>
<td>The class of RDF datatypes.</td>
</tr>
<tr>
<td>rdfs:Container</td>
<td>The class of RDF containers.</td>
</tr>
<tr>
<td>rdfs:ContainerMembershipProperty</td>
<td>The class of container membership properties, rdf:_1, rdf:_2, ..., all of which are sub-properties of 'member'.</td>
</tr>
</tbody>
</table>

### II.1.3.1 RDF/S properties

<table>
<thead>
<tr>
<th>Property name</th>
<th>comment</th>
<th>domain</th>
<th>range</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdfs:subClassOf</td>
<td>The subject is a subclass of a class.</td>
<td>rdfs:Class</td>
<td>rdfs:Class</td>
</tr>
<tr>
<td>rdfs:subPropertyOf</td>
<td>The subject is a subproperty of a property.</td>
<td>rdf:Property</td>
<td>rdf:Property</td>
</tr>
<tr>
<td>rdfs:domain</td>
<td>A domain of the subject property.</td>
<td>rdf:Property</td>
<td>rdfs:Class</td>
</tr>
<tr>
<td>rdfs:range</td>
<td>A range of the subject property.</td>
<td>rdf:Property</td>
<td>rdfs:Class</td>
</tr>
<tr>
<td>rdfs:label</td>
<td>A human-readable name for the subject.</td>
<td>rdfs:Resource</td>
<td>rdfs:Literal</td>
</tr>
<tr>
<td>rdfs:comment</td>
<td>A description of the subject resource.</td>
<td>rdfs:Resource</td>
<td>rdfs:Literal</td>
</tr>
<tr>
<td>rdfs:member</td>
<td>A member of the subject container.</td>
<td>rdfs:Resource</td>
<td>rdfs:Resource</td>
</tr>
<tr>
<td>rdfs:seeAlso</td>
<td>Further information about the subject resource.</td>
<td>rdfs:Resource</td>
<td>rdfs:Resource</td>
</tr>
<tr>
<td>rdfs:isDefinedBy</td>
<td>The definition of the subject resource.</td>
<td>rdfs:Resource</td>
<td>rdfs:Resource</td>
</tr>
</tbody>
</table>

RDF/S provides simple, but powerful modeling primitives for structuring domain knowledge into classes and sub-classes, properties and sub-properties, and can impose restrictions on the domain and range of properties, and defines the semantics of containers.

The simple meta-modeling elements can limit the expressiveness of RDF/S. Some of the main limiting deficiencies are identified in (Antoniou & van Harmelen, 2004):

- **Local scope of properties**: in RDF/S it is possible to define a range on properties, but not so they apply to some classes only. For instance the property `eats` can have a range restriction of food that applies to all classes in the domain of the property, but it is not possible to restrict the range to plants for some classes and meat for others.
- **Disjointness of classes** cannot be defined in RDF/S.
- **Boolean combinations of classes** are not possible. For example `Person` cannot be defined as the union of the classes `Male` and `Female`.
- **Cardinality restrictions** cannot be expressed.
- **Special characteristics of properties** like transitivity cannot be expressed.
II.1.4 Ontologies

A good starting point for understanding what ontology entails, is to consider the next figure, adopted from (Daconta, Obrst, & Smith, 2003), which places a number of knowledge models on a continuum. As you go from the lower left corner to the upper right, the richness of the expressible semantics increases. This is shown on the right side of the arrow with some typical expressions that have some sort of defined semantics for the particular model. The names for the knowledge models are given on the left of the arrow. It is important to note that all of the terms on the left hand side have been called “ontology” by at least some authors, which is part of the source for confusion about the word.

Models based on the various points along the ontology spectrum have different uses (McGuinness, 2003). In the simplest case, a group of users can agree to use a controlled vocabulary for their domain. This of course does not guarantee that they will use the terms in the same way all the time, but if all the users including database designers chose their terms from an accepted set, then the chances of mutual understanding are greatly enhanced.

![Ontology Spectrum Diagram]

**Figure 1: The ontology spectrum**

Perhaps the most publicly visible use for simple ontologies is the taxonomies used for site organization on the World Wide Web. This allows designers to structure information and users to browse and search. Taxonomies can also help with sense disambiguation since the context of a term is given by the more general terms in the taxonomy.

Structured ontologies provide more sophisticated usage scenarios. For instance, they can provide simple consistency and completeness checks. If all *products* must have a *price* then web sites can automatically be checked for missing or conflicting information. Such ontologies can also provide completion where partially specified information can be expanded automatically by reference to the terms in the ontology. This expanded information could also be used for refining search, for instance. Ontologies can also facilitate interoperability, by aligning different terms that might be used in different applications (McGuinness, 2003).

Now we are in a position to see why the ontologies on the most formal end of the spectrum are often taken as the default interpretation in the context of the semantic web, providing the conceptual underpinning for “… making the semantics of metadata machine interpretable” (Staab & Stuber, 2004). But for the semantics of a domain model to be machine interpretable in any interesting way, it must be in a format that allows automated reasoning in a flexible manner. Obviously, taxonomies can specify little in this sense. Database schemas are more powerful, but limit the interpretation to a single model in terms of reasoning over the knowledge base. The only automated reasoning that can be performed is what is allowed by the relational model, i.e., retrieval of tuples actually represented...
in the database. Formal logic based reasoning about ontologies can consider multiple possible models (Bordiga & Brachman, 2003). They are at the same time more formally constrained and more semantically flexible than database schemas. Ontologies based on different logical models can support different kinds of inference, but a minimal set of services should include reasoning about class membership, class equivalence, consistency, and classification (Antoniou & van Harmelen, 2004). The ontology representation language adopted by the Web Ontology Working Group of the W3C is the Web Ontology Language (OWL). OWL is a response to a number of requirements (Smith, Welty, & McGuinness, 2004) including the need for a language with formal semantics that enables automated reasoning, and to address the inherent limitations of RDF/S as described above.

II.1.5 Web Ontology Language (OWL)

The OWL language extends earlier W3C recommendations, such as RDF and DAML+OIL, with richer modeling primitives, providing a set of constructs to define web ontologies. It sets the basic infrastructure to build some simple inferences between web resources and enables a knowledge-management architecture.

According to the original design goal, OWL was to be a straightforward extension of RDF/S, guaranteeing downward compatibility such that an OWL aware processor could also understand RDF/S documents without modification. Unfortunately this did not succeed because the generality of some RDF/S elements (e.g. the semantics of class as “the class of all classes”) does not make RDF/S expressions tractable in the general case. In order to maintain computational tractability, OWL processors include restrictions that prevent the interpretation of some RDF/S expressions. The OWL specification defines three sublanguages: OWL Full, OWL DL, and OWL Lite: OWL Full is upward and downward compatible with RDF, but OWL DL and OWL Lite are not. The names of the three sub languages of OWL describe their expressiveness, keeping in mind a fundamental tradeoff between expressiveness, efficiency of reasoning, and support for human understanding. OWL Full has constructs that make the language undecidable. Developers should therefore only use OWL Full if the other two sub languages are inadequate for modeling the relevant domain, or if they wish to maintain full compatibility with RDF. Similarly, OWL DL should be used if OWL Lite is not sufficient. Details of the syntax and semantics can easily be obtained from the technical documentation web site of the W3C.

In the 2000 was born, from a subproject of OWL, an extension of the language for describing Web Services, named previously DAML-S and now OWL-S. This extension is the actual W3C recommendation for the semantic description of Web Services.

II.1.6 1.1 Overview of the OWL Metamodel

The Web Ontology Language (OWL) is a semantic markup language for publishing and sharing ontologies on the World Wide Web. Where earlier knowledge representation languages have been used to develop tools and ontologies for specific user communities, they were not defined to be compatible with the architecture of the World Wide Web in general, and the Semantic Web in particular.

OWL builds on RDF and RDF Schema and augments the RDFS vocabulary for describing properties and classes: among others, relations between classes, cardinality, equality, richer typing of properties, characteristics of properties, and enumerated classes.

1http://www.w3.org/2001/sw/WebOnt/
1.1.1 Organization of the OWL Metamodel

The primary OWLBase package contains the metamodel constructs common to both OWL DL and OWL Full - corresponding to the abstract syntax elements of the Web Ontology Language. Two additional sub packages contain constraints required to distinguish the two dialects (OWL DL and OWL Full) from one another. The package structure for the OWL metamodel and its dependencies on the RDF metamodel are shown in Figure 1.

1.1.2 Design Considerations

**Naming** As in the RDF metamodels, prefixes are used in naming MOF classes and MOF properties that directly represent OWL classes and OWL properties, respectively. For example, OWLClass represents owl:Class and OWLimports represents owl:imports. Individual, which does not have a prefix, represents something which is not explicitly defined in the RDF/XML serialization of OWL. Exceptions to this convention include OWLUniverse, OWLGraph, and OWLStatement, included for vendor use in mapping RDF graphs and/or statements to OWL, for mapping to other metamodels.
1.2 OWL Ontology

As shown in Figure 2, OWL ontology consists of a collection of facts, axioms, and annotations, defined in terms of RDF graphs and statements. The ontologyID (in the form of the URI reference it has by virtue of being a resource) allows us to make statements about a particular ontology - including annotations such as the relationship between a particular ontology and other ontologies, version information, and so forth.

![Figure 2 Ontology Diagram](image)

An OWL ontology contains a sequence of annotations, axioms, and facts. Annotations on OWL ontologies can be used to record authorship and other information associated with an ontology, including imports references to other ontologies. The main content of OWLOntology is carried in its axioms and facts, which provide information about classes, properties, and individuals in the ontology. Names of ontologies are used in the abstract syntax to carry the meaning associated with publishing an ontology on the Web. The intent is that the name of an ontology in the abstract syntax is the URI where it can be found, although this is not part of the formal meaning of OWL. Imports annotations, in effect, are directives to retrieve a Web document and treat it as an OWL ontology.

II.1.7 Web Ontology Language for Web Services (OWL-S)

OWL-S is a W3C recommendation that provides a complete framework, based on the RDF syntax and the OWL ontology model, for describing Web Services in terms of what they can do, how they can work and how they can be caught up.

OWL-S was born for solving challenges related to Web Services such as semantic Web Services discovery, dynamic Web Services composition and Web Services execution monitoring.
Figure 5  OWL-S model of service

OWL-S describes a Web Service as a resource presented by a ServiceProfile, described by a ServiceModel and supported by a ServiceGrounding:

- The ServiceProfile has the similar functions of a registry, such as UDDI. It describes organizations that present the service, provides a list of the I/O parameters, preconditions and effects used by the service and describes additional information such as the quality and the accuracy.
- The ServiceModel provides the fundamental information needed for the composition and interoperation of services. OWL-S allows three different kinds of processes: Atomic, Simple and Composite.
- The ServiceGrounding specifies explicitly the input and output links between the atomic processes of the service, showing how the communications between these processes are to be realized as messages. However, the real implementation of the Service Grounding is a set of elements which links each atomic process to the corresponding WSDL file.

II.1.8 OWL-S and WSMO

As described above web services must be discovered, described, and appropriately connected in an implementation independent way. (Berardi et al., 2005) outline 3 different approaches for web service discovery, on a trade-off between ease of provision and accuracy: 1) natural language keyword matching, 2) ontology based keyword matching (increasing precision through a controlled vocabulary), and 3) semantic matchmaking, based on precise semantic descriptions of services and service needs. Currently, service descriptions in UDDI, for example, are primarily text descriptions with no semantic markup, requiring a lot of manual input, and not facilitating the more advanced approaches to discovery. But there are two emerging approaches to facilitate machine processable semantic markup for web service descriptions: Web Service Modeling Ontology (WSMO Working Group, 2005) and OWL-S (OWL-S Coalition, 2005).

WSMO consists of three main components: a modeling framework of core elements for semantic web services, a formal description language (Web Service Modeling Language - WSML) and an execution environment (WSMX). The WSMO core elements are

1. Ontologies – provide the formally specified terminology of the information used by all other components
2. Goals – objectives that a client wants to achieve by using Web Services
3. Web Services – semantic description of web services including functional capability and usage interface
4. Mediators – connectors between components with mediation facilities for handling heterogeneities

Each of these elements is further described by non-functional properties including the Dublin Core Metadata Set, versioning information, quality of service information, and other relevant annotations.

Together these components are able to define the terminology of the domain and how it relates to applications, and to describe the service in terms of its pre-conditions, post-conditions, effects, and mediators required during the discovery and execution of the service. WSMO is described in more detail below.

OWL-S is a W3C initiative to provide an ontology and language to describe web services. It is less revolutionary than WSMO, as is evidenced by its closer ties to current standards like WSDL and UDDI. Its primary role is to assist discovery, which it fulfills by specifying three key components of a service:

- **What does the service provide for prospective clients?** The answer to this question is given in the "profile" which is used to advertise the service.
- **How is it used?** The answer to this question is given in the "process model." This perspective is captured by the ServiceModel class.
- **How does one interact with it?** The answer to this question is given in the "grounding." A grounding provides the needed details about transport protocols.

Thus each service presents a ServiceProfile (what it does), is described by a Service Model (how it works), and supports a ServiceGrounding (how to access it).

While OWL-S is a less comprehensive approach, there are certain similarities between the two approaches.

- **OWL-S Service Profile ≈ WSMO capability + goal + non-functional properties.** WSMO separates provider (capabilities) and requester points of view (goals) while OWL-S Profiles combine existing capabilities (advertisements) and desired capabilities (requests)
- **OWL-S process model ≈ WSMO Service Interfaces.** The process model in the OWL-S ServiceModel roughly corresponds to the interfaces in the WSMO Web Services descriptions of WSMO.
- **OWL-S Grounding ≈ WSMO Grounding.** Both provide a mapping to WSDL.

Nevertheless, clear differences exist in the overall architecture as well as the reliance of WSMO on explicitly defined mediators. A key objective of the WSMO is to define a taxonomy of mediators to translate between messages produced by one Web service and those expected by another. In the OWL-S vision this is a step which can detract from the primary purpose of discovery. To be sure the translation problems still need to be solved, but OWL-S assumes this will be possible through some form of composition (Ankolekar et al., 2004). But this has some implications for the use of each system in a specific context.

**II.1.9 Web Service Modeling Framework (WSMF and WSMO)**

The *Web Service Modeling Framework (WSMF)* [124] defines a framework for supporting the development, discovery and description of Web Services and Web Service compositions [54]. The project was dedicated to the following complementary principles:

1. **Maximal Decoupling,** meaning the services or components in a composite application should be as independent of each other as possible [54].

35
2. Scalable Mediation, a scalable mediation service should enable a service to speak with any other service in the system [54].

The ultimate vision of WSMF is to enable an application to automatically discover and invoke the service or services needed to fulfil a request. The services should be described in such a way that the application may evaluate both the functional and non-functional properties of the service, such as cost, quality of service, etc. Such a system could enable automatic electronic cooperation between enterprises [54]. In order to achieve the goals set in the vision, the following functional requirements were set for the WSMF:

1. Automated Discovery and Comparison of Service Vendors, today the service selection is done manually, thus putting a limit to the scalability of electronic commerce applications [54].
2. Automated Handling of Heterogeneous Data Formats, ontology technology must be utilized to help defining proper data formats and to automate the mapping between different formats [54].
3. Automated Handling of Heterogeneous Business Logics, a mediation service is needed to compensate for the differences in the business logics of cooperating trading partners [54].

The WSMF consists of four main elements.

Ontologies Defines the terminology that is used in the WSMF specification, enabling reuse of terminology and interoperability between components referring to the same or linked terminologies [54].

Goal Repositories Stores the description of client objectives, or goals. Goal descriptions consists of pre-conditions that specify the required input of a service fulfilling the given goal, and post-conditions specifies the output of a service fulfilling the goal. The goals are kept separate from the Web Service specifications since a goal may be relevant for several services [54].

Web Services Descriptions Are an important aspect for enabling automatic handling of Web Services. The WSMF regards the external properties of a Web Service to be the most important description, a so called “black box” description. However, if more details are needed, such as knowing the execution state of a composite service, a more detailed “grey box” description may be provided [54].

The WSMF defines thirteen aspects of services that should be described. The basic “black box” description elements are the unique identification of a service, the description of purpose through goal references, the definition of input and output ports, data structures for input and output data and error data to be passed to the requester. For the “grey box” description further details are added, such as data flow specifications, proxies for enabling dynamic service invocation at runtime, control flow specifications for defining the execution sequence of a complex service, handlers for exceptions and compensation schemes for errors. The message exchange and understanding protocols related to the service must also be described. Finally WSMF notes that if there are non functional properties that are important for the service these also must be covered in the description [54].

Mediators The Mediators of the WSMF focuses on mediating between the different interaction styles of web services. This can include [54]:

1. Mediation of data structures, this typically consists of integrating various XML formats using transformation languages like XSLT [71]. WSMF hints towards utilizing ontology technologies for more intelligent and automated information integration [54].
2. Mediation of business logics, for linking Web Services that provide complementary services but have interaction patterns that does not fit out of the box. For mediating between different business logics it is necessary to compensate for data and process sequencing mismatches [54].

3. Mediation of message exchange protocols, for ensuring reliable messaging across of messaging protocols. Problems to be solved here are message acknowledgement and duplication prevention [54].

4. Mediation of dynamic service invocation, some Web Services may be required to invoke other services in order to provide their functionality. This can be done dynamically by referring to goals that must be fulfilled, the mediator will then find the appropriate service to fulfil these sub goals at runtime [54].

The WSMF proposes an extension of the now deprecated WSFL [81] language as a syntax for WSMF. WSFL is similar in spirit to the WSMF but lacks the modeling features requires, such as the use of ontologies and the separation of public and private business logics [54]. The Semantic Web Service based DAML-S [5] language is also mentioned as a starting point for a web based markup language for WSMF, but also here some modeling features are missing, such as the difference between public and private processes or between business logics and message exchange protocols. DAML-S also does not provide any mediation features [54]. The last technology to be mentioned as interesting is the Process Specification Language (PSL) which is proposed as a language for defining the formal semantics for the WSMF [54].
III Agent-Oriented Computing

III.1 Introduction

The primary goal of multi-agent systems is to obtain a coherent, adaptive and robust behavior thanks to the interaction of a large amount of software components, eventually heterogeneous and individually non-reliable [8].

III.1.1 Historical context

In the beginning, software systems were considered as entities that receive data as input, and transform this data in order to deliver results as output. But as problems became larger and larger, it was necessary to decompose such systems into several less complex sub-systems. Their capacity to solve local problems and to interact provided a mechanism for solving more complex global problems. This is the approach taken by the DAI (Distributed Artificial Intelligence) community.

The multi-agent community, stemming from the DAI community, considers agent interaction as the key for systems that want to be capable of dealing with complex tasks. Activities of a system are then considered as the result of interactions between various concurrent and autonomous agents. These agents work within groups, or societies, using cooperation, concurrency, conflict solving mechanisms, in order to fulfill the tasks of the system [18].

Nevertheless, DAI is not the only origin of multi-agent systems, Artificial Life also contributed to their emergence. The combination of these two fields provided both a cognitive and a reactive dimension. Indeed, DAI is attached to the concept of intelligence as symbolic reasoning, and Artificial Life is attached to the concepts of autonomy, behavior, viability,...

III.1.2 What is an agent?

III.1.2.1 Definition

According to M. Wooldridge, an agent is a software system able to act autonomously and flexibly in a changing environment [72].

According to G. Weiss [62], an agent is a computational entity, like a program or a robot, that perceives and acts autonomously on its environment.

III.1.2.2 Characteristics

According to M. Wooldridge [72], agents’ main characteristics are autonomy, reactivity, pro-activity and social ability.

- **Autonomy** An agent is able to take decisions without the intervention of an human or another agent.

- **Reactivity** An agent is able to perceive its environment and keep a constant link with it, in order to deal with the change within this environment.

- **Pro-activity** An agent should not only be directed by events generated by its environment, it must also take initiatives, following its own objectives.

- **Social ability** An agent must be seen as a social being that is integrated in a society, in which tasks, resources and roles are distributed among the agents.
III.1.3 Types of agents

There are two main "schools" inside the multi-agent community. The first one, called the "cognitive school", considers agents as entities that are basically intelligent. The second one, called the "reactive school", considers agents as very simple entities that react directly upon environment’s changes [70].

Cognitive agents, or intelligent agents, are able to work independently on certain tasks, due to their reasoning capabilities. They also need sometimes to coordinate their tasks, and negotiate on certain goals. This approach is directly influenced by the DAI.

Reactive agents have no individual intelligence, and are based on the Artificial Life models. They only react to the stimuli’s created by the environment and by the other agents. In reactive systems, it’s the whole system that is considered to have an intelligent behavior, not the agent itself (like in an ant’s colony for example).

The different schools of thought as well as the different needs of applications have lead to different agent architectures. The architecture of a single agent can be one of the following:

- **Reactive** – where the agent responds to some stimuli from its environment. The behavior of the agent is stimulus-response. The agent does not contain any internal model of its environment and does not deliberate on its actions.

- **Deliberative** – where the agent contains an internal model of its environment and reasons about its world to decide upon its actions

- **Hybrid** – a combination of reactive and deliberative agent architectures where the agent contains a reactive component as well as a deliberative component. Hybrid agents are capable of behaving reactively as well as reason about its environment.

III.1.4 Agents vs Objects

Are agents merely objects endowed with more (smarter) functionality? Many researchers question the properties an object must possess in order to be defined an agent. As agents, objects are characterized by their behavior, the state in which they are, and by the fact that they communicate thanks to simple message passing.

The first difference concerns the control of their behavior. Indeed, objects don’t have any control over their behavior, whereas agents control it in order to fulfill their goals.

The second but most significant difference resides at the level of autonomy and flexibility. The use of private and public methods do not permit objects to control the application of these methods. Whereas agents are able to decide whether they want to apply a request, according to the specific agents goals.

The third difference concerns the activity of agents and objects. Agents are active, in the sense that their execution is an infinite loop in which they observe their environment, update their state, and select actions to perform. Objects only become active when another object invoke a method on it.

The last difference is that agents are "able" to make incorrect decisions, and learn from these errors. Whereas objects can not make such erroneous decisions: errors committed are only programming and conception errors, from which they can’t learn anything.
III.1.5 What is a multi-agent system?

III.1.5.1 Definition
A multi-agent system is a set of intelligent agents interacting in a common environment, in order to fulfill a set of goals, or to accomplish a set of actions [70].

III.1.5.2 Characteristics

- A multi-agent system is not dedicated to the resolution of one particular problem, they don’t have predefined organization. Agents reason on their organization depending on the problem to solve.

- Agents inside a multi-agent system do not need to know the global goal to act, they are autonomous.

- Data in a multi-agent system is decentralized, i.e. distributed among the agents.

- Multi agent systems are characterized by coordination, communication and relations that exist between their agents. Coordination may be either cooperation in order to achieve a common goal, or negotiation in order to satisfy optimally each agent’s interest. Communication is achieved thanks to several protocols, like FIPA and KQML.

III.2 Multi Agent Systems

Multi-agent systems (MAS) extend single-agent architectures with an infrastructure for interaction and communication. Ideally, MAS exhibit the following characteristics:

- they are typically open and have no centralized designer;

- they contain autonomous, heterogeneous and distributed agents;

- they provide an infrastructure to specify communication and interaction protocols.

Agents in MAS can be viewed as an implementation of modular design. The complex structure and functionality of the MAS as a whole are realized by an arrangement of components (agents) that each has a certain specific functionality and autonomy. As such, MAS are very interesting as an integration paradigm. There are two main differences with traditional paradigms, such as object-orientation: (a) it radicalizes the notion of component autonomy, and (b) it provides a coordination and communication infrastructure that traditionally has been present in a rudimentary form only (cf. CORBA), so that these aspects had to be part of the component code.

Several architectures and models for MAS have been proposed that handle coordination in different ways. One of the first mechanisms in Distributed AI (which can be regarded as the forerunner of MAS), was Contract Net Protocol (CNP) which had a market-like coordination structure (for more details, see below). Another early architecture is based on mediators. The concept of mediator, first introduced by Gio Wiederhold, was a way to deal with the integration of knowledge from heterogeneous sources. An example of MAS architecture based on the concept of mediators, and typically not employing any centralized control, is RETSINA [33].

III.2.1 Agent Societies

The term "society" is used in a similar way in agent society research as in human or ecological societies. The role of the society is to allow its members to coexist in a shared environment and
pursue their respective roles in the presence and perhaps cooperation with others. Main aspects in the definition of society are purpose, structure, rules and norms. Structure is determined by roles, interaction rules and communication language. Rules and norms describe the desirable behavior of members and are established and enforced by institutions that often have a legal standing and thus lend legitimacy and security to society members.

When multi-agent systems are considered from an organizational point of view, the concept of desirable social behavior becomes of utmost importance. That is, from the organizational point of view, the behavior of individual agents in a society should be understood and described in relation to the social structure and overall objectives of the society. Not any kind of social, or asocial, behavior is acceptable. On the other hand, as agents are autonomous by definition, the individual behavior of the agents cannot be controlled directly. The solution for this dilemma is to be found in the society that should provide social mechanisms by means of which the behavior of the agents can be streamlined into a desirable direction without compromising at any time the autonomy of the agent. Davidson [11] has proposed a classification for artificial societies based on the following characteristics:

- openness - describing the possibilities for any agent to join the society;
- flexibility - indicating the degree agent behavior is restricted by society rules and norms;
- stability - defining the predictability of the consequences of actions;
- trustfulness - specifying the extent to which agent owners may trust the society.

Open societies impose no restrictions on agents joining the society. They assume that participating agents are designed and developed outside the scope and design of the society itself and therefore the society cannot rely on the embedding of organizational and normative elements in the intentions, desires and beliefs of participating agents but must represent these elements explicitly. In closed societies, on the other extreme, it is not possible for external agents to join the society. Agents in closed societies are explicitly designed to cooperate towards a common goal and are often implemented together with the society [16]. Closed societies provide strong support for stability and trustfulness properties, but only allow for very little flexibility and openness. The large majority of existing MAS are closed. Besides open and closed societies, Davidson distinguishes semi-open and semi-closed

### III.2.2 Coordination in MAS

Multi-agent systems that are developed to model and support organizations need coordination frameworks that mimic the coordination structure of the particular organization. The organizational structure determines important autonomous activities that must be explicitly organized into autonomous entities and relationships in the conceptual model of the agent society [13]. Furthermore, the multi-agent system must be able to adapt to changes in organization structure, aims and interactions.

Coordination can be defined as the process of managing dependencies between activities [55]. Coordination is an important problem inherent to the design and implementation of MAS [2]. Examples of coordination theories are joint-intentions [44], shared plans [5] and domain-independent teamwork models [59]. Behavioral approaches are gaining terrain in agent research. Concepts as organizational rules [73], norms and institutions [39] and social structures [22] all start from the idea that the effective engineering of MAS needs high-level agent-independent concepts and abstractions that explicitly define the organization in which agents live [16]. An important contribution in agent coordination is Jennings’ joint responsibility framework [83], based on human teamwork models. This model is based on the idea that being a part of a team implies some sort of
responsibility towards the other members of the team; joint responsibility. Jennings built on the work of Cohen et al. by distinguishing between the commitment that underpins an intention and the associated convention, where a commitment is a pledge or a promise to do something and conventions are means of monitoring commitments, e.g. specifying when a commitment can be abandoned.

From a programming point of view, coordination models can be divided into two classes: control-driven and data-driven [19]. Control-driven models are systems made up of a well-defined number of entities and functions, in which the flow of control and the dependencies between entities need to be regulated. The data-driven model is more suited for open societies where the number of entities and functions is not known a priori and cooperation is an important issue.

In DAI, coordination approaches were often based on contracting. The most famous example of these is the Contract Net Protocol (CNP) [53] for decentralized task allocation. In short, CNP acts as follows:

- all agents must register with the matchmaker;
- when an agent needs to locate other agents, it must send a request message to the matchmaker describing the requested service;
- other agents can then make bids;
- once bids have been received, the request will select one (according to some criteria) and allocate the task to that bidder;
- the bidder can then accept the task.

The CNP protocol assumes that all agents are eager to contribute, and the most appropriate bid is the bid of the agent with the best capability and availability. A more sophisticated version is the TRACONET model [57]. In this model, agents are supposed to be self-interested. This means that contractors have to pay a price for the service performed. Contractors try to minimize the costs by selecting the bidder with the lowest price (all things being equal). Potential subcontractors try to maximize their benefit, and may sometimes discard offers or respond by a counter offer.

Contractual Agent Societies (CAS) apply the concept of contracting to the coordination of MAS, and are inspired by work in the areas of organizational theory, economy and interaction sociology, which model organizations and social systems after contracts [12]. A market place is a set of mutually trusted agents, and when an untrusted agent wants to join the market place, it applies at a socialization service that not only plugs in the agent technically, but also makes it agree on a social contract. Social contracts govern the interaction of a member with the society. A social contract is a commitment of an agent to participate in a society and includes beliefs, values, objectives, protocols and policies that agents agree to obey in the context of the social relationship. A mechanism of social control may be negotiated as part of the social contract, defining deviations from agreed "normal" behavior and corresponding sanctions (e.g. banning). The notion of contract is not necessarily related to market places; it was also used in a collaborative Information System context in the thesis of Verharen [15].

Economics and organizational theory consider that relationships between and within organizations are developed for the exchange of goods, resources, information and so on. Williamson argues that transaction costs are determinant for the choice of organizational model [67]. Transaction costs are not just the costs of delivering a message. They will rise when the unpredictability and uncertainty of events increase, or when transactions require very specific investments, or when the risk of opportunistic behavior of partners is high. Roughly speaking, when
transaction costs are high, economic agents tend to choose a hierarchical model in order to control
the transaction process. If transaction costs are low, than the market is usually a better choice [56].
Although this economic theory cannot be applied directly to agent societies, it strongly suggests that
agent societies should support both
the hierarchical and the market model, and not just one of them.

Powell introduced networks as another possible coordination model [45]. Networks stress the
interdependencies between different organizational actors and pay a lot of attention to the
development and maintenance of communicative relationships, including the definition of rules and
norms of conduct within the network. Coordination in markets is achieved typically through a price
mechanism in which independent actors are searching for the best bargain. Hierarchies are typically
coordinated by supervision, that is, actors that are involved in authorized relationships acting
according to certain routines. Networks achieve coordination by mutual interest and
interdependency [61].

A good overview of Agent Coordination is available from [84].

III.2.3 Negotiation

Another important area of work in multi-agent systems is in agent negotiations. Agents within a
multi-agent system resolve conflicts through negotiation. Several negotiation techniques and
models have been studied and applied in agent systems. An overview is available in [85].

III.2.4 Communication

The main challenge of coordination and collaboration among heterogeneous and autonomous
intelligent systems in an open, information-rich environment is that of mutual understanding. A
mechanism for communication must include both a knowledge representation language (based on
ontologies) and a communication protocol.

III.3 Platform independent Model for Agents (PIM4Agents)

Initial work around a model-driven approach to agent technologies was worked on during the
INTEROP mobility stay of Christian Hahn from DFKI at SINTEF in 2006. Further work from this
resulted in the PIM4agents proposal that was presented at I-ESA’2007, which is briefly described
below.

Various agent-oriented methodologies and metamodels exist to describe multiagent systems (MAS)
in an abstract manner. Frequently, these frameworks specialise on particular parts of the MAS and
only few work has been invested to derive a common standardisation which limits the impact of
agent-related systems in commercial applications. In this paper, we present a metamodel for agent
systems that abstracts from existing agent-oriented platforms. Furthermore, we illustrate how an
agent-oriented software development process in accordance to the model-driven development
(MDD) approach could be formulated around the abstract view on agent systems and thus (i) further
the development process of agent systems to increase the interoperability among agent platforms
and (ii) facilitates the interoperability of agent platforms and potential areas of application.

The development of a platform-independent model for agent systems (in the following
PIM4Agents) is an important step towards the adoption of Model-Driven Architecture guidelines
and standards proposed by the OMG. This work presents a core metamodel that abstracts from
existing agent-oriented platforms and could thus be considered as platform-independent. This
metamodel addresses the conceptual and technological interoperability barrier as it aims to define
platform-independent modeling language constructs that can be used to design an agent system on a very abstract level and transfer the concepts to agent-based applications using a model-driven approach. Based on this transformation, automatically generated agent systems can finally be executed (see Figure 1). By developing a PIM4Agents, we pursue the following goals:

- Bridging the gap between (i) the various existing agent-oriented modeling methods and thus improving the interoperability between agent platforms and (ii) potential application areas and agent systems.
- Clearly define a platform-independent abstraction that can be used to integrate and define mappings from particular applications to agents.
- Reusing vertical mappings to agent-related frameworks that could be shared for different application-oriented metamodels. In this case, only the horizontal mappings from PIM to PIM - need to be newly formulated.

![Figure 6 PIM4Agents model](image)

One challenge in defining the platform-independent model for agents – like for every metamodel on the PIM level – is to decide which concepts to include and abstract from the target execution platforms (PSMs) that support the architectural style of agent-based systems.

In order to support an evolution of the PIM4Agents metamodel, it is structured around a small core that could be possibly enriched by adding smaller extensions, each focusing on a specific aspect of an agent system. Grouping modelling concepts in this manner allows the metamodel evolution by adding (i) new modelling concepts for particular aspects, (ii) extending existing modelling concepts, or (iii) defining new modelling concepts for describing additional aspects of agent systems (e.g. security).
The core of the PIM4Agents metamodel is shown in the figure above. The metamodel is centered on the concept of Agent, the autonomous entity capable of acting in the environment. Each Agent has access to a set of Resources from its surrounding Environment.

The Capability represents the set of Behaviours the Agent can possess. It allows to group Behaviours that, conceptually, have a correspondence with regard to what they allow the Agent to do. Each Behaviour can be simple or composed by subbehaviours, therefore a whole hierarchy of specific Behaviours can be created. Each Behaviour may also send or react to a Message according to a given Protocol. The Role is an abstraction of the social behaviour of the Agent in a given social context, usually a Cooperation. This Role specifies the responsibilities of the Agent in that social context. Correspondingly, the Cooperation represents the interaction between the Agents performing the required set of Roles. The detailed realisation of this interaction is described by a Protocol that indicates what are the Messages to be expected from each of the Roles at which point in time. The execution of the Protocol is performed by a set of Behaviours, each of which sends and/or reacts to messages in accordance to its Role. Agents can take part in an Organisation, a special kind of Cooperation that also has the same characteristics as Agent. Therefore, the Organisation can perform roles and has Capabilities which can be performed by its members, be it Agents or suborganisations. The multiple inheritance of the Organisation, from Agent and Cooperation, also allows it to have its own internal protocol that specifies how the Organisation coordinates its members.

### III.4 Conclusion on Agents

Agent technologies are emerging as an important set of technologies that can complement and also be built around and on top of other infrastructure technologies. The PIM4Agents approach shows a
possibility of unifying concepts from various agent technologies, in a way that again can be unified with more general service oriented approaches.
IV Bibliography – References

IV.1 Bibliography – Service Oriented Computing


[ATHENA WKD 5.1 2004], ATHENA project: Perspectives in Service-Oriented Architectures and their Application in Environments that Require Solutions to be Planned and Customisable, Working Document WKD 5.1., March 2004


[ebXML 2003] OASIS and UN/CEFACT, Electronic Business XML (ebXML),
http://www.ebxml.org


[Eric 2004] Mark Ericson, Interoperability: the Key to Web Service Quality, published on 27.05.2004
http://www.mywebservices.org/index.php/article/articleview/1468/1/50/


[Gnutella], Gnutella, www.gnutella.com


[Ideas 2001] Ideas IST-2001-37368, Deliverables D 3.4, D 3.5, D 3.6: A Gap Analysis Required Activities in Research, Technology and Standardisation to close the RTS Gap


[MBI 2003a] Microsoft, BEA, IBM, Web Services Coordination (WS-Coordination)


[RosettaNet], Homepage for RosettaNet, available at http://www.rosettanet.org


[SOAP] SOAP, [http://www.w3.org/TR/SOAP](http://www.w3.org/TR/SOAP)


[UDDI] [http://www.uddi.org/](http://www.uddi.org/)


IV.2 Bibliography – Semantic Services


IV.3 Bibliography – Agent-oriented Computing


[60] Paolo Giorgini University, *The tropos methodology: An overview*.


