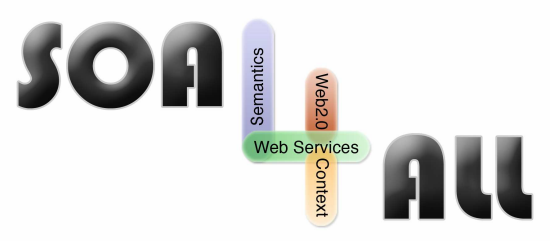

Project Number: **215219**

Project Acronym: **SOA4ALL**

Project Title: **Service Oriented Architectures for All**

Instrument: **Integrated Project**

Thematic Priority: **Information and Communication Technologies**

D1.1.1 Design Principles for a Service Web v1

Activity N:	1 – Fundamental and Integration Activities	
Work Package:	1 – Service Web Architecture	
Due Date:	M6	
Submission Date:	31/08/2008	
Start Date of Project:	01/03/2008	
Duration of Project:	36 Months	
Organisation Responsible of Deliverable:	CEFRIEL	
Revision:	2.0	
Author(s):	Elisabetta Di Nitto	CEFRIEL
	Rafael González-Cabero,	Atos Origin
	Christophe Hamerling,	EBM WebSourcing
	Jacek Kopecký,	UIBK
	Omair Shafiq,	UIBK
	Tomas Vitvar,	UIBK
	Gianluca Ripa	CEFRIEL
	Lai Xu	SAP
	Maurilio Zuccalà	CEFRIEL

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)		
Dissemination Level		
PU	Public	**X**

Version History

Version	Date	Comments, Changes, Status	Authors, contributors, reviewers
1.0	02/05/2008	First draft material	Rafael González-Cabero (Atos Origin)
1.1	05/05/2008	Definition of table of content and section on terminology	Elisabetta Di Nitto (CEFRIEL)
1.2	04/06/2008	Section on terminology extended	Gianluca Ripa (CEFRIEL)
1.3	30/06/2008	Structure extended, sections on basic principles and terminology updated	Elisabetta Di Nitto (CEFRIEL), Rafael González-Cabero (Atos Origin)
1.4	15/07/2008	Section about enterprise service bus filled in	Christophe Hamerling, Jean-Pierre Lorre (EBM WebSourcing)
1.5	25/07/2008	Contributions from various partners	Elisabetta Di Nitto (CEFRIEL), Christophe Hamerling (EBM WebSourcing), Jacek Kopecký (STI), Omair Shafiq (STI), Tomas Vitvar (STI), Lai Xu (SAP)
1.6	29/07/2008	Review comments	Stuart Campbell (TIE)
1.7	19/08/2008	Review comments	Nathalie Steinmetz (UIBK)
2.0	31/08/2008	Final version	Elisabetta Di Nitto, Gianluca Ripa, Maurilio Zuccalà (CEFRIEL)

Table of Contents

Acronyms

Acronym	Definition
B2B	Business to Business
BPEL	Business Process Execution Language
BPMN	Business Process Modeling Notation
EAI	Enterprise Application Integration
ESB	Enterprise Service Bus
IT	Information Technology
JBI	Java Business Integration
JCA	J2EE Connector Architecture
OMG	Object Management Group
OWL	Web Ontology Language
REST	Representational state transfer
SAWSDL	Semantic Annotations for WSDL
SESA	Semantically-Enabled SOA
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
UDDI	Universal Description Discovery and Integration
WSDL	Web Service Description Language
WSFL	Web Services Flow Language
WSML	Web Service Modeling Language
WSMO	Web Service Modeling Ontology
XLANG	XML-based extension of WSDL

Executive summary

SOA4All aims at providing an infrastructure that brings Web services and Service Oriented Architecture (SOA) to a Web scale, based on the design principles that made the Web such a successful platform.

One of the main objectives of the Work Package 1 of the SOA4All project is to define an architecture for SOA4All getting the best out of the *Web principles*, the *SOA principles*, and the *Semantic Web principles*. In order to achieve this result, a multidisciplinary research team has been created involving end-users and people with different research interests. Thus, there was the need to agree on the driving principles, to define a common terminology and to set the research challenges to address during the project. Some of these issues have been already addressed in the Description of Work but they are refined and stated more precisely in this document.

This deliverable analyzes the main principles, definitions, technologies, and challenges that will drive the development of the first version of the SOA4All architecture. In the first part of this deliverable several reference models and glossaries are considered:

- the **OASIS Reference Model for Service Oriented Architectures 1.0** [MacKenzie 2006],
- the **OASIS Reference Ontology for Semantic Service Oriented Architectures release candidate 2** [Kerrigan 2008],
- the **Conceptual Model** built by the **SeCSE project** [Colombo 2005 and SeCSE 2007],
- the **NEXOF-RA glossary** that at the current time is under preparation (see http://www.nexof-ra.eu/).

The second part presents the principles and rationale behind a service Web architecture along with outlining how such principles will provide the means and methods for an Internet-scale deployment and adoption of SOA infrastructures. These principles are:

- **Service-Orientation Principles**. The SOA design paradigm captures a distinctive approach to the analysis, design, and implementation to all types of service-oriented IT environments, introducing a set of principles which govern aspects of communication, architecture, and processing logic. According to [Erl 2007] these principles are: Standardized Service Contract Principle; Loose Coupling Principle; Abstraction Principle; Reusability Principle; Autonomy Principle; Statelessness Principle; Discoverability Principle and Composability Principle;
- **Web principles.** The Web is based also on a collection of principles that lead to a highly scalable means for electronic publication. The provision of Web-based lightweight integration infrastructures will facilitate openness and easy adoption for both the service provider and the consumer. SOA4All should analyze and apply these principles to service-orientation, which will lead to a global, dynamically changing environment of services accessible for third-party usage, beyond the boundaries of single organizations.
- **Autonomic Computing principles.** The concept of Autonomic Computing was first introduced in [Kephart 2003] where autonomous systems were characterized as self-manageable systems. In the context of services, the idea of self-management is very important, especially when we think at systems that are created by composing services offered by third parties and that should be able to react to the cases in which these services provide incorrect results or are unresponsive.
- **Formal Semantic Descriptions.** Current standards for describing Web services use syntactic notations such as WSDL. Since these descriptions are machine readable but not machine understandable, only IT personnel can carry out most of the tasks associated with creating and maintaining Web service-based applications such as Web service discovery, composition, and invocation. These tasks can be automated to a great extent by applying

semantic technologies (such as OWL-S [Martin 2004], WSMO [Roman 2006], WSDL-S [Akkiraiju 2005]).

The principles presented above are very high level and can be addressed from various points of view, using various technologies. Besides, some specific challenges are identified and are addressed by the SOA4All project. The SOA4All Service Web Architecture focuses on these challenges and discusses how they relate to the general principles. These challenges have been preliminarily derived from the main SOA4All objectives. They are:

- **Heterogeneity**. It should be assumed that worldwide distributed systems contain many different kinds of hardware / software systems and environments. Thus, a Service Web infrastructure should be able to handle such heterogeneity.
- **Worldwide access mechanisms.** Services should be accessible worldwide. This means that they should be identifiable in a unique way and should be invoked despite potential heterogeneity.
- **Semantic provisioning of services.** Formal semantic descriptions of services allow powerful reasoning and precise matching of requests with services. However, formal semantics introduces a relevant computational overhead that has to be taken into account for usage at runtime. Thus, as long as computational overhead represents a problem, some lightweight approaches should be studied.
- **Decentralized dynamicity and adaptability.** A central control on the life cycle of all services would hamper access and therefore scalability. Thus, service provisioning and modification should be as much as possible decentralized and unconstrained, without hampering the possibility of building solid service compositions out of them.
- **Matching requests and services.** Even if services are accessible worldwide, without proper support for matching requests and services it may be difficult for a service consumer to find the right service to use. Thus, proper matching mechanisms need to be provided. While so far the literature has focused on centralized matchmakers, the real challenge is to distribute the execution of matching algorithms on multiple nodes.
- **Enabling n:m asynchronous interactions.** The classic client-server model of interaction no longer reflects the nature of the Web. Thus, we should introduce richer models of interaction to address situations where multiple entities collaborate by playing different roles (even multiple roles), each of them sending and receiving complex messages.
- **Enabling service prosumers.** Active consumers (often referred as prosumers) become part of the content providing process and often even form democratic communities of content creators. Applying this idea of content creation (that is part of the phenomenon known as Web 2.0) to service creation is not as simple since to date the development of services has been an activity for specialists. Therefore, the challenge is to understand the kinds of tools that should be offered to users in order to let them become service prosumers.
- **Supporting both machine and human-based computation.** The last challenge is to see humans as being part of SOA4All service Web infrastructure. This requires proper user interfaces and mechanisms supporting the communication between services based on machine computation and human-based services.

The work presented here will be incrementally extended and completed during the project. In particular, the next step of the Work Package 1 will be to review the challenges driving the development of the SOA4All architecture in the light of the requirements that are being defined by other Work Packages.

1. Introduction

Service-orientation is a broad design paradigm that permits the separation of concerns and uses services as the basic building blocks of functionality. Services can be called by their users in order to obtain some results. They are usually owned by third parties that manage their provisioning usually by exploiting some software systems. One of the main characteristics of SOA is that service interfaces are published, discovered and invoked typically over the Internet. SOA is currently enjoying massive adoption in large corporations since it promises to close the gap between what companies require and what IT is able to deliver. Moreover, SOA promises a more flexible IT infrastructure that is able to react to business changes more quickly than the classic monolithic IT systems. Finally, SOA is built on top of internet standards, making interaction among companies easier and cheaper. This, on the long term, will enable the formation of business ecosystems able to traverse enterprise boundaries.

Unfortunately, these promises are still not realized. Most stakeholders currently use SOA primarily for internal integration and far less for external integration. In particular, companies seem still reluctant to expose their business services on the Internet. This situation is changing however, as increasingly companies such as Yahoo, eBay, Amazon and Google are exposing their services over the Web; allowing others to integrate these services in order to build new applications in so-called "mash-ups."

Parallel to the emergence of SOA as a valid infrastructure alternative for large enterprises, the Web has continued its success and has become the dominant information medium for consumers and for the entire range of companies. It has helped many small and medium sized enterprises to be globally visible in a world dominated by global players. Consumers have been dazzled by new means of participation brought forward by Web 2.0 technologies such as blogs. Technologies that further simplify user contributions such tagging have unleashed the power of communities with efforts such as Wikipedia demonstrating that it is possible to create large and shared information sources.

Today, the Web contains just around 25,000 Web services[(1)] of which many, perhaps the majority, are experimental - a minuscule amount in comparison to the 30 billion Web pages constituting its content. In fact, SOA is largely still an enterprise specific solution exploited by, and located within, large corporations as part of their in-house supply chains. Nevertheless, complex mobile devices and more efficient wireless communications facilitate ubiquitous computing and as optical and broadband communication infrastructures expand, the number of Web services is expected to grow exponentially in the next few years. This expected growth could be also supported by other phenomena:

- More companies will publish their offerings as services, which are accessible through the Web, and which has been inspired by the success of early adopters (e.g. Amazon).
- Web 2.0 has popularized concepts such as mash-ups and syndication though technologies such as RSS, Atom. They have thereby illustrated comparatively simple means for business networking and business flexibility.
- Efforts to turn the Web into a general platform for accessing and interconnecting arbitrary devices and arbitrary services are maturing.

Hence, there is a need to master very large systems and to handle the complexity of these systems confidently. While humans will certainly be involved in those activities concerning system design, their participation in repetitive activities related to configuration and maintenance should be reduced, providing systems with learning capabilities and self-organizing functions. Crucially, systems and software must be secure, robust, dependable and optimized in terms of functionalities

[1] source: http://seekda.com

to cater for multiple audiences.

Besides, the progresses of semantic technologies enable the automation of data and service management, exploiting the semantics associated to the information. The Semantic Web aims at extending the Web so creating the pre-conditions needed by software agents for carrying out complex tasks while browsing Web pages. Smart agents, exploiting the semantic technologies, can extracts information from Web resources (i.e. text mining, image processing, video processing, sound processing, …) and from the relationships the users establish between them (i.e. by means of links and annotations or implicitly putting items in their virtual shopping cart) adding value to the information and creating new knowledge.

In particular, we envisage that the combination of Semantic Web and SOA will lead to the creation of a "service Web"—a Web where billions of parties are exposing and consuming services seamlessly and transparently and where all types of stakeholders, from large enterprises to SMEs and individual end users, engage as peers consuming and providing services within a network of equals.

However, SOA will not scale nor be widely adopted without properly incorporating the same principles that made the Web scale to a worldwide communication infrastructure. A significant mechanization of service lifecycle activities[2] and a balanced integration of services provided by humans and machines are also pre-requisites to provide support to such an infrastructure. In a service-oriented world, users should be able to seamlessly discover and select services on the basis of their requirements and context. Users should be allowed to create on their own new complex services, using as building block other more simple pre-existing services, in a computer aided fashion. These new user generated services should be (semi) automatically adapted or integrated in to the whole system. Solving these problems in a scalable and manageable manner is a major pre-requisite to realize a Web interconnecting a large number of services (as the current Web does for information sources).

1.1 Purpose and Scope of this deliverable

SOA4All aims at providing an infrastructure that brings Web services and Service Oriented Architecture (SOA) to a Web scale, based on the design principles that made the Web such a successful platform. To enable this infrastructure there are a number of key research questions that must be addressed, namely:

- How can interoperability issues be resolved to allow provided services to be invoked?
- What mechanisms can be used to decouple service providers and requests to enable n:m relationships between services and maximise interactions on the Service Web?
- Which core services are required within the infrastructure to provide users with access to the Web of Services and what are the right interfaces for interacting with them?
- How can we ensure that the infrastructure scales to the Web as expected?

One of the main objectives of the work package 1 is to define an architecture for SOA4All getting the best out of the Web principles, the SOA principles, and the Semantic Web principles. In order to achieve this result a multidisciplinary research team has been created involving people with different research interests and end-users. Thus, there was the need to agree on the driving principles, to define a common terminology and to set the research challenges to address during the project. Some of these issues have been already addressed in the Description of Work but they are refined and stated more precisely in this document.

This deliverable analyzes the main principles, definitions, technologies, and challenges that will

[2] location, negotiation, adaptation, composition, invocation and monitoring as well as service interaction requiring data, protocol and process mediation

drive the development of the first version of the SOA4All architecture.. Well-established Web principles such as openness, need for interoperability, decentralization, dynamicity, etc., are considered as starting points in this work. The experience and results of previous efforts within W3C's Web services standardization, in OASIS Semantic Execution Environment architecture group, and in the NEXOF-RA project have been incorporated and used as important inputs for this deliverable.

Consistently, the document is structured as follows

- Section 3 establish a proper terminology for the SOA4All project,
- Section 4 survey the literature to identify those principles that are considered to be the pillars of the service Web,
- Section 5 briefly describes the basic technologies upon which the service Web will be built
- Section 6 identifies those challenges to address within the SOA4All project.

2. Basic definitions

Services represent an effective solution to let software systems distributed across the world and developed by different organizations to interoperate. Examples of this come from the world of business to business interaction, but also in other contexts, such as ambient intelligence and pervasive computing we are assisting to an increasing interest in this area. The interest is even stronger in the area of grid computing where services permit to parallelize computationally-intensive tasks.

Given this large variety of usage contexts, and the relative youth of the discipline itself, a fully agreed set of definitions is still missing. This is why the first part of this deliverable focuses on establishing those definitions that will be used through the following of the SOA4All project.

Several reference models and glossaries can be taken as starting points in this activity. The ones that we consider are the OASIS Reference Model for Service Oriented Architectures 1.0 released in 2006 [MacKenzie 2006], the OASIS Reference Ontology for Semantic Service Oriented Architectures release candidate 2 [Kerrigan 2008], and the conceptual model built by the SeCSE project [Colombo 2005 and SeCSE 2007]. These will be described in detail in the following sections. Finally, Section 3.4 provides the selection of definitions that will be adopted by the project. These have been mainly derived form the NEXOF-RA glossary that at the current time is under preparation (see http://www.nexof-ra.eu/).

2.1 The OASIS model

Figure 1 shows the main concepts in the OASIS model [MacKenzie 2006].

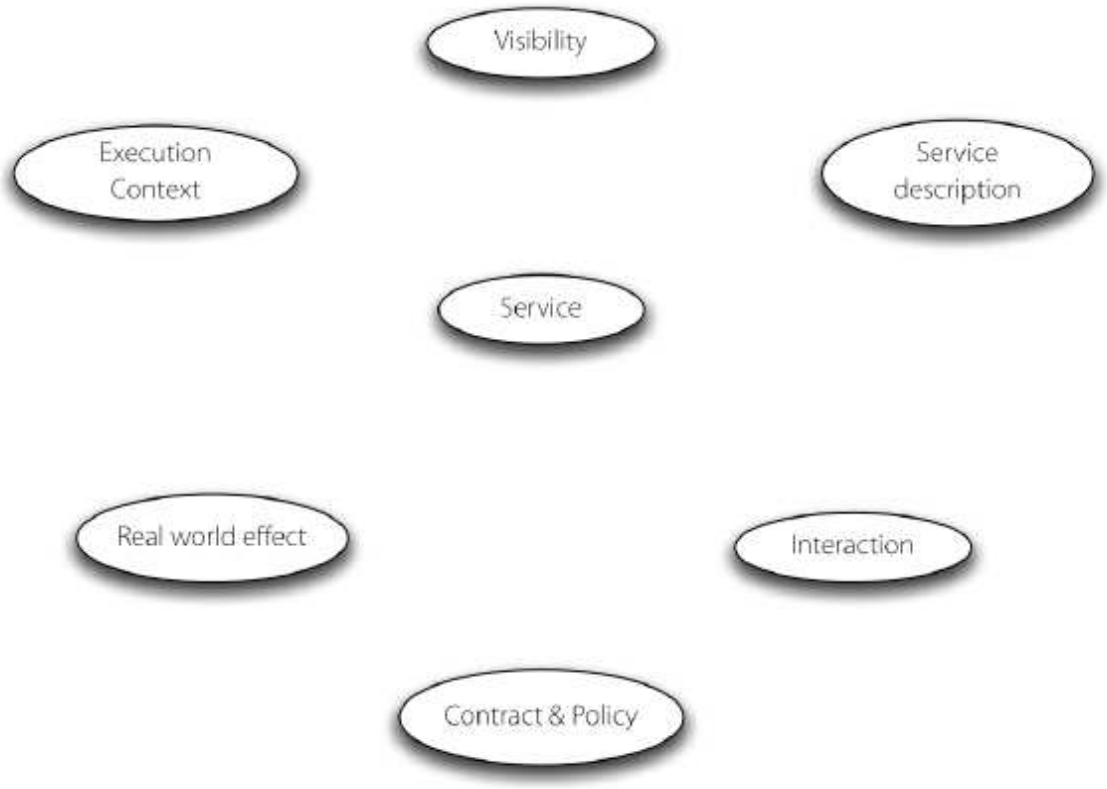

Figure 1. Main concepts from the OASIS reference architecture.

In this model a *Service* is “a mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the *Service Description*”. A Service is offered by a *Service Provider* and can

be exploited by a *Service Consumer*. Of course, this is possible if the Service Consumer has *Visibility* on the Service. The concept of Visibility is decomposed in the concepts of *Awareness*, *Willingness*[3], and *Reachablity*.

The *Interaction* with a Service usually happens through message exchange and it is based on the knowledge of the *Information model* (i.e., the format of data, their structural relationships, and the definition of the used terms) and the *Behavior model* (i.e., the temporal sequence of actions supported by the Service) of the Service. The interaction with a Service can lead to the occurrence of some *Real World Effects*.

While the aforementioned terms (Interaction, Real World Context, and Visibility) are related to the usage of the service by some other party, other important elements that are connected directly to Services are Contract and Policies, Service Descriptions, and Execution Context. A *Policy* represents some constraint or condition on the use, deployment or description of an owned entity as defined by any participant. A *Contract*, on the other hand, represents an agreement by two or more parties. The *Execution Context* includes the technical and business elements needed for an interaction with the Service as well as processes and agreements that are in place. All interactions are grounded in a particular execution context, which permits service providers and consumers to interact and provides a decision point for any policies and contracts that may be in force.

More details on the definition of the various terms can be found in [MacKenzie 2006]. Here it is highlighted the fact that such characterization of SOAs is particularly interesting because of its focus on the usage of a service more than on its internals. In fact, a special attention is paid to the context in which the service is executed and to its effects on the real world. The concepts introduced in the model are fully independent from specific technologies. This means that they do not apply only to the main instantiation of SOAs, i.e., Web Services, but are in principle applicable to other instantiations (see, for instance, REST, Jini, OSGI services). For this reason building the SOA4All architecture on top of this concepts leads to an architecture capable of supporting different and new technologies in the future.

2.2 The OASIS SEE Reference Ontology

The OASIS SEE (Semantic Execution Environment) Technical Committee (TC)[4] continues the work initiated by the Web Service Execution Environment (WSMX) project and working group[5]. It also receives and provides input to several other projects in Europe such as DIP[6], ASG[7], TripCom[8], SUPER[9], SOA4ALL[10], COIN[11], and other projects in the area of Semantic Web Services under FP6 and FP7 programme . The aim of the OASIS SEE TC is to provide guidelines, justifications and implementation directions for an execution environment for Semantic Web Services. The resulting infrastructure will incorporate the application of semantics to service-oriented systems and will

[3] The willingness to establish an interaction between two parties could be automatically determined using descriptions. For example, a mediator for descriptions may provide 3rd party annotations for reputation. Another source for reputation may be a participant's own history of interactions with another participant

[4] http://www.oasis-open.org/committees/semantic-ex/

[5] http://www.wsmx.org

[6] http://dip.semanticweb.org

[7] http://asg-platform.org

[8] http://www.tripcom.org

[9] http://www.ip-super.org

[10] http://www.soa4all.org

[11] http://www.coin-ip.eu

provide intelligent mechanisms for consuming Semantic Web Services.

The specifications developed by OASIS SEE TC is a Reference Ontology for Semantic Service Oriented Architectures (SSOAs) [Kerrigan 2008] which is based on extensions of the OASIS SOA Reference Model described in section 2.1. It extends the Reference Model with the key concepts of semantics that are relevant for Semantically-enabed Service Oriented Architectures. It helps use semantic technologies to further solve problems that SOAs (Service Oriented Architectures) are limited by. The model from OASIS SEE TC specifications has been defined formally using an ontology. The aim of this ontology is to provide a point of reference formally specified so that it can support the definition and development of SSOAs.

The OASIS Reference Ontology for SSOA extends the Oasis Reference Model, focusing on the aspects concerned with Visibility, Service description, Real world effect, and Interaction. In particular, Figure 2 shows the main extensions proposed by the Reference Ontology on the Reference Model.

The main principle of the Reference Ontology is to describe all service-oriented concepts through an ontology-based formalism that are connected via mediators. Thus, Mediators represent the way the services are made visible and therefore replace the Visibility concept in the Reference Model. Indeed, the Ontology on the side of the Service Description concept introduces the concept of Goal Description. This is a representation of the requirements for a service from the point of view of a consumer. Goal Descriptions and Service Descriptions describe the Functionality expected or offered by a service and thus enable Reachability of the service. The Behavioural and Information models, describe the Interaction that occurs between the service and its consumers.

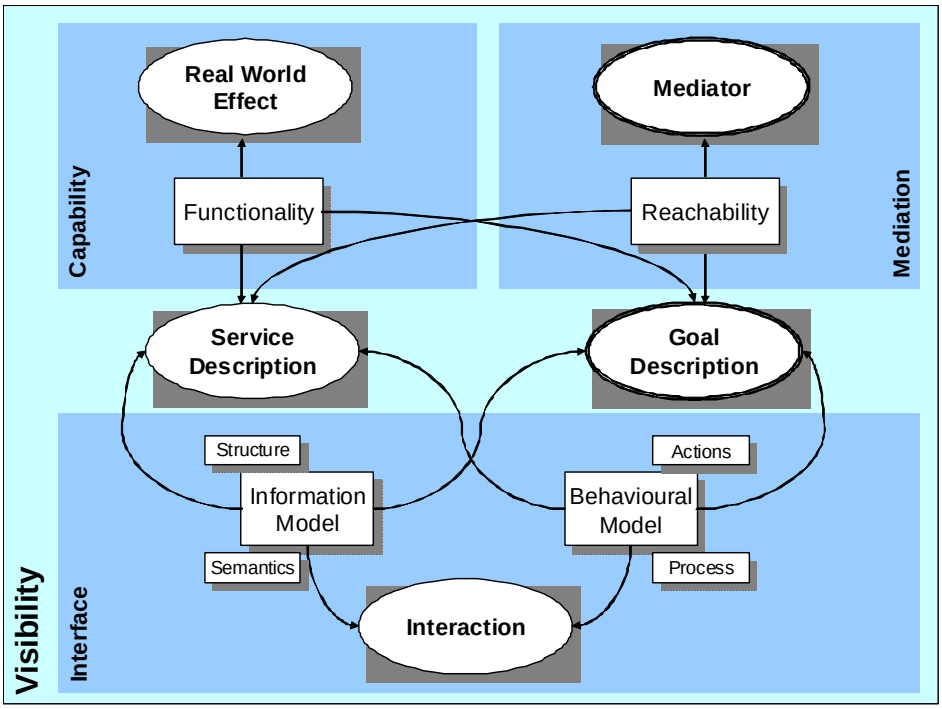

Figure 2 – Relationships between the OASIS Reference Model and the OASIS Reference Ontology.

Figure 3 details the main concepts in the OASIS Reference Ontology that are the following:

- Ontology: Service Descriptions, Goal Descriptions. Mediators (see below) can import Ontologies in order to utilize the terminology that they provide
- Service Description: A formal description of all the information needed in order to use (find, select, invoke, …) a service, made by the provider
- Goal Description: Formalization of the requesters needs in accordance with their specific context

- Capability: Service Descriptions and Goal Descriptions can be linked to a Capability. Capability is described as the pre and post conditions on the state of the information space and on the state of the real-world; the former are named *preconditions* and *postconditions* the latter as *assumptions* and *effects*
- Mediators: Described in terms of the entities it is able to connect to. States how it will resolve mismatches;
- Interface: It is a part of the service description. It specifies in detail how the communication with the service should take place.

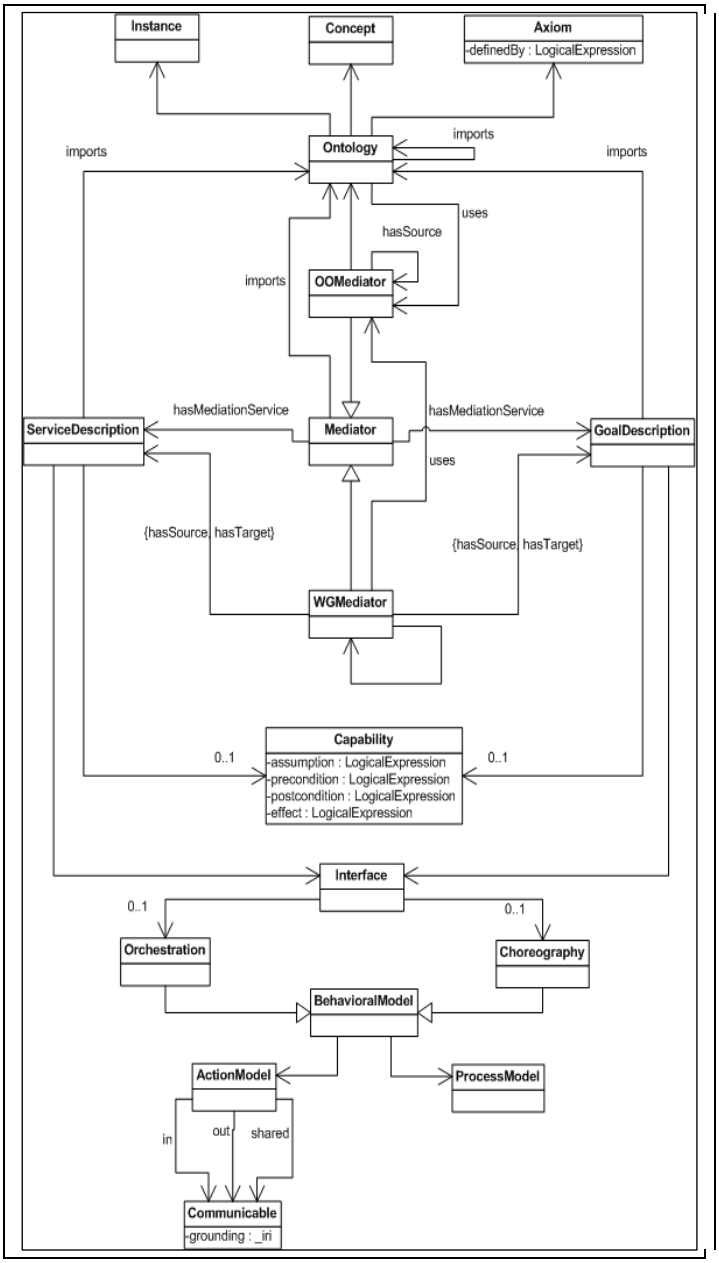

Figure 3 – OASIS Reference Ontology.

The SEE TC provides a testbed for the Web Services Modeling Ontology (WSMO)[12], which is anticipated as a contribution for use by the TC and will seek to demonstrate the viability of using

[12] http://www.wsmo.org

WSMO concepts, relationships and definitions as a means to achieve successful dynamic interoperation of multiple ambient services, whether or not they share a common design or source. The SEE TC will not implement actual software products or solutions based on the specifications developed along the course of work of this group.

2.3 The SeCSE model

SeCSE is a FP6 project focusing on engineering service-centric systems. It proposes a conceptual model aiming at providing a common terminology for services. This model has been designed to be extensible in order to be easilyaccommodated and extended. It is worth to note that the SeCSE Conceptual Model has been investigated, by the NEXOF-RA team (www.nexof-ra.eu) as an important piece of work for the Reference Model of NESSI Open Service Framework (NEXOF).

The SeCSE model, when compared to the OASIS Reference Architecture and Reference Ontology, provides more details on some aspects, in particular, it defines the structure of Service Descriptions and their publication and discovery processes, the structure of a Service Composition, what is need to negotiate Service Level Agreements (SLAs) and what is need to monitor a service. In addition, the model highlights the various stakeholders that have a role in a service-centric system life cycle, as well as the various classes of services that may participate in a service-centric system. The model is described using UML and is explained in [Colombo 2005 and SeCSE 2007]. Within this document we focus on two of the main diagrams; the one describing the stakeholders interested in the service-oriented activities and the one that classifies services in various categories.

The diagram of Figure 4 shows that three main kinds of stakeholders can play some role in a service-centric system. These stakeholders are Systems (also composed of some hardware part), Persons, and Organizations. All these can act as Services, Service Consumers, Providers, Developers, etc. The classification is incomplete, that means that other roles could be discovered in the future. The fact that Service is available as role (Actor) in the diagram allows us to represent the fact that various stakeholders, not necessarily a Software System, can implement a Service.

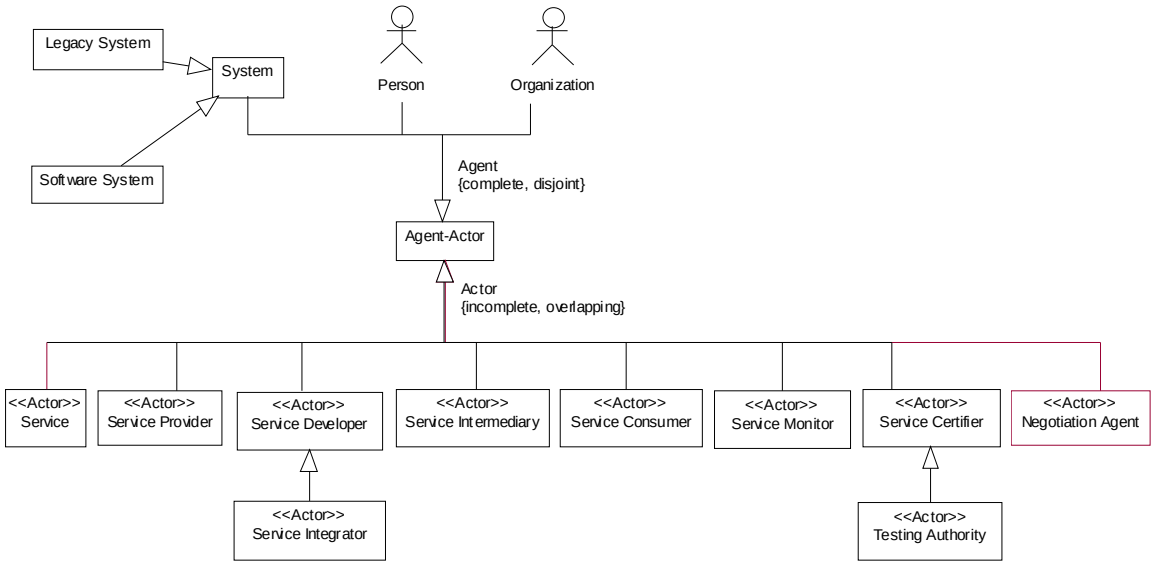

Figure 4 – Stakeholders and roles in the SeCSE model.

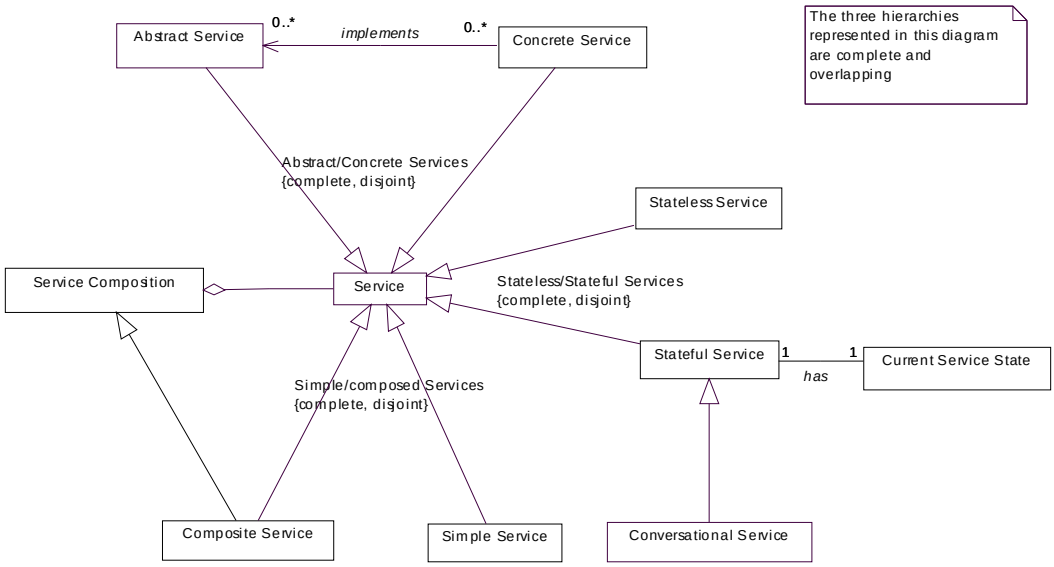

Figure 5 – Classes of services in the SeCSE model.

As shown in Figure 5 a Service can be Abstract when it has a description but it misses an implementation or it can be Concrete if the opposite case applies. It can be Stateless or Stateful. So called Conversational Services are seen as a specific case of Stateful services where the state that is maintained by the service is related with the specific conversations it is engaging with its Consumers. Finally, a Service can be Simple or Composite. A Composite Service is obtained by aggregating other Services in a Service Composition.

The SeCSE conceptual model also highlights the fact that a Service is characterized by one or more Service Descriptions that are composed of various Facets. Each Facet focuses on a specific aspect of a Service (its syntactic interface, its behaviours, its QoS characteristics, …). The discovery of Services is based on a matching between a Service Request and a Service Description and can happen at various stages in the service life cycle (at requirement time, design time, runtime). A Service Composition can be bound to Abstract or Concrete Services. In the first case, the binding has to be concretized before the execution of the corresponding invocation. The Monitoring of services is an activity that is performed while services are running, but it requires planning and design activities prior execution in order to properly define what and how to monitor.

2.4 The SOA4All definitions adapted from the NEXOF-RA glossary

The NEXOF-RA glossary[13] is a selection of terms that is being prepared by the NEXOF-RA project, in cooperation with NESSI related projects, and is intended to form a common glossary across the projects and beyond. It takes many of its definitions from established standards or pseudo standards. These terms concern various aspects of the service lifecycle ranging from requirement analysis to operation. One of the inconveniences of such glossary with respect to the models that we have previously described is that the various terms are defined in isolation and the relationships between terms are not highlighted. This can easily lead to ambiguities and missing definitions. Its main advantage stands in the fact that it is much broader than the other models.

SOA4ALL defines its terminology starting from the NEXOF-RA glossary in order to enable interaction with the other projects of the NESSI platform. In particular, in this section those terms of the glossary are selected that are of interest of the SOA4All project. SOA4ALL will propose modifications to some definitions and introduce new definitions to cover aspects that are relevant to

[13] See http://www.nexof-ra.eu/.

the SOA4All project. The new definitions are identified by looking at the main objectives of the SOA4All project and at its case studies.

In the following subsection we group the definitions in various categories. Section 2.4.1 presents the main terms in the service context, Section 2.4.2 lists the terms defined by the Semantic Web community, Section 2.4.3 lists the possible actors that can be involved in the service life cycle, Section 2.4.4 focuses on the main activities belonging to the service life cycle, Section 2.4.6 presents the architecture-related terms, Section 2.4.7 focuses on terms related to technology, and, finally, Section 2.4.8 presents terms that come from the Business Management domain.

2.4.1 Main entities under study

Atomic Service: An Atomic Service is a service that does not invoke other services.

Binding: An association between an interface and a concrete implementation. A binding specifies the protocol and data format to be used in transmitting messages, defined by the associated interface, to a specific endpoint.

Composite Service: A Composite Service is a software service implemented through the composition of software services.

Conformance: Fulfilment of a product, process or service of all requirements specified; adherence of an implementation to the requirements of one or more specific standards or technical specifications. [EBXML 2001]

Execution context: The execution context of a service interaction is the set of infrastructure elements, process entities, policy assertions and agreements that are identified as part of an instantiated service interaction, and thus forms a path between those with needs and those with capabilities. *Note: this definition is not part of the NEXOF-RA glossary. It has been derived from [MacKenzie 2006]*

Non-functional Property or Quality of Service: a set of quantifiable quality properties of a service.

Non-functional Requirement: A non-functional requirement specifies a particular quality of the system rather than a function of the system. Two kinds of non-functional requirements are typically distinguished: 1. Execution qualities observable at run-time such as security, availability, reliability, etc.; 2. Evolution qualities observable at design time such as scalability and extensibility.

Requirement: A condition or capability needed by a user to solve a problem or achieve an objective. [Pohl 2005]

Service: A service is an abstract entity consisting of a set of capabilities offered by one or more providers to consumers. The service is provided by means of consumer service requests. The capabilities of the service and information how to use these capabilities are described in a service description. It can be realized by living beings, information systems, machines, etc.

Service Behaviour: The observable effects of an operation or event, including its results. [EBXML 2001]

Service Capability: A Service Capability is the functionality offered by a service, including quality of the service.

Service Composition: Service Composition is a combination of service invocations that allows the consumer to achieve a given goal. *Note: the NEXOF-RA definition is the following: Service Composition is the act of executing a service coordination.*

Service Contract: The service contract is a formal, agreed, binding contract between a service consumer and a service provider.

Service Choreography: A Service Choreography is the specification of interactions among a set of services, described from a global perspective.

Service Description: A service description is a set of documents that describe the interface, the accessibility and the capability of a service.

Service Interface: The service interface is the specification of how to perform service requests.

Service life cycle: Comprises the design-time, runtime, and retirement of a service.

Service Implementation: The core business logic written in a specific language. [Margolis 2007]

Service Orchestration: A Service Orchestration is the description of how a specific service can be realised by interacting with other services. The orchestration is under control of a single endpoint. An Orchestration may be executable.

SLA: The SLA is a formal, agreed, binding contract between a service consumer and a service provider constraining the quality of service.

Software Service: A Software Service is a special service which can be accessed by the service consumer only via a piece of software. This software constitutes the interface between the service consumer and the software service. It does not determine whether the service is realized by a human or a piece of software. The interfaces to human services are proxies to enable an interaction with the human.

Web Service: A Web Service is a software service designed to support interoperable XML based machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). [W3C 2004]

2.4.2 Semantic aspects

Concept: A concept is an element of a semantic model. This specification makes no assumptions about the nature of concepts, except that they must be identifiable by URIs. A concept can for example be a classifier in some language, a predicate logic relation, the value of the property of an ontology instance, some object instance or set of related instances, an axiom, etc. [W3C 2007]

Semantics: Semantics in the scope of this specification refers to sets of concepts identified by annotations. [W3C 2007]

Ontology: An ontology is an explicit specification of a conceptualization[14]. [Gruber 1993]

Semantic Annotation: A semantic annotation in a document is additional information that identifies or defines a concept in a semantic model in order to describe part of that document[15]. [W3C 2007]
Note: the NEXOF-RA definition is the same, plus the following sentence: In SAWSDL, semantic annotations are XML attributes added to a WSDL or associated XML Schema document, at the XML element they describe. Semantic annotations are of two kinds: explicit identifiers of concepts, or identifiers of mappings from WSDL to concepts or vice versa. [W3C 2007]

Semantic Model: A semantic model is a set of machine-interpretable representations used to model an area of knowledge or some part of the world, including software. Examples of such models are ontologies that embody some community agreement, logic-based representations, etc. Depending upon the framework or language used for modelling, different terminologies exist for denoting the building blocks of semantic models. [W3C 2007]

[14] "A conceptualization is an abstract, simplified view of the world that we wish to represent for some purpose. Every knowledge base, knowledge-based system, or knowledge-level agent is committed to some conceptualization, explicitly or implicitly." [Gruber 1993]

[15] In SAWSDL, semantic annotations are XML attributes added to a WSDL or associated XML Schema document, at the XML element they describe. Semantic annotations are of two kinds: explicit identifiers of concepts, or identifiers of mappings from WSDL to concepts or vice versa. [W3C 2007]

2.4.3 Actors

Actor: This may be either: 1. A person or organization that may be the owner of agents that either seek to use services or provide services. 2. A physical or conceptual entity that can perform actions. Examples: people; companies; machines; running software. An actor can take on (or implement) one or more roles. An actor at one level of abstraction may be viewed as a role at a lower level of abstraction.

Agent: An agent is a software program acting on behalf of a person or organization. [W3C 2004]

Application Service Provider: An Application Service Provider (ASP) is a organisation that offers individuals or organisations access over the Internet to applications and related Software Services that would otherwise have to be located in their own personal or enterprise computers.

Business Entity: Something that is accessed, inspected, manipulated, produced, and worked on in the business. [EBXML 2001]

Business Partners: An entity that engages in business transactions with another business partner(s). [EBXML 2001]

Service Requester: Is the actor asking for a service. A service requester might agree SLAs with providers, ensures the services address the correct business requirements, and provides funds for using services. *Note: the NEXOF-RA definition is the following:* Agrees SLAs with providers, ensures the services address the correct business requirements, and provides funds for using services.

Service Aggregator: A client application that automatically pipelines the services needed to retrieve a requested result.

Service Intermediary: A service intermediary is the actor whose main role is to transform messages in a value-added way.. *Note: the NEXOF-RA definition is the following: A service intermediary is a Web service whose main role is to transform messages in a value-added way. (From a messaging point of view, an intermediary processes messages en route from one agent to another.) Specifically, we say that a service intermediary is a service whose outgoing messages are equivalent to its incoming messages in some application-defined sense. [W3C 2004]*

Service Provider: A Service Provider makes services available to consumers. It may define with consumers some Service Level Agreements that regulate the terms of the service offer. *Note the NEXOF-RA definition is the following: Agrees SLAs with requesters and makes services available to them in compliance with the SLA.*

Service Registry or Registry: A mechanism whereby relevant repository items and metadata about them can be registered such that a pointer to their location, and all their metadata, can be retrieved as a result of a query. [EBXML 2001]

Service Repository or Repository: A location or set of distributed locations where Repository Items, pointed at by the registry, reside and from which they can be retrieved. [EBXML 2001]

Service Role: An abstract set of tasks that is identified to be relevant by a person or organization offering a service. Service roles are also associated with particular aspects of messages exchanged with a service. [W3C 2004]

Service Goal: it is the representation of an objective for which fulfilment is sought through the execution of a service. *Note: this definition has been derived from [Roman 2006]*

2.4.4 Types of services

The definitions in this section are not part of the NEXOF-RA glossary and have been derived from [SeCSE 2007].

Stateless service: A service that does not maintain an internal state between different invocations.

Stateful service: A service that maintain an internal state between different invocations. It

includes: a) services for which the execution of the same operation at different times by different consumers can produce different results, even if the input data are the same; b) services that require their operations to be called in a specific order. *Note: the NEXOF-RA definition is the following:* A Service with a state. It includes: a) services for which the execution of the same operation at different times by different consumers can produce different results, even if the input data are the same; b) services that require their operations to be called in a specific order.

Conversational (or interactive) service: Service that requires its operations to be called by the same Service Consumer in a specific order. In many cases, the result of the execution of one operation depends also on the operations previously executed by the same Consumer.

Simple service: Service not formed by other services.

Composite service: Aggregate service resulting from the composition of different services.

Abstract service: It represents the abstract notion of Service. It has a double aim; 1.From the Service Provider perspective it is intended to capture the idea of "business service", that is, an offered service which does not necessarily have a concrete implementation. 2.From the Service Consumer perspective it represents a desired service, thus related to a Service Request. An Abstract Service may be published and discovered, just as concrete services. Before being able to serve a Service Request it has to be "concretized" in a concrete service.

Concrete service: A service that has a concrete implementation and can, therefore, be executable.

2.4.5 Activities

Business Activity: A business activity is used to represent the state of the business process of one of the partners. [EBXML 2001]

Business Process: A Business Process is a collaborative service that is closely linked to a business purpose.

Business Process Modelling: The activity of analysing and designing the structure of business processes and the resources needed to implement them.

Design Time: Comprises all activities that a service provider has to perform prior to the provision of a service. The design time ends with the agreement on an SLA.

Runtime: Comprises all activities that service providers and service consumers have to perform during the provision of a service. *Note: the NEXOF-RA definition is the following: Comprises the provision of a service starting with the agreement on an SLA.*

Semantic Web Service Creation. Service Creation combines activities related to construction of semantic descriptions of the Web services, their annotation, and the creation of the ontologies used to define the formal descriptions. *Note: this definition is not part of the NEXOF-RA glossary. It has been derived from the Infraweb project.*

Service Discovery: An activity of finding and identifying a service that is expected to fulfil user requirements.

Service Deployment: All of the activities that make a service available for use.

Service Execution: The process for delivering operational services to the service consumer.

Service Invocation. The invocation of services is usually hard-coded within client applications. In contrast, automatic Service Invocation is the automatic invocation of a Web service by a computer program or agent, given only a declarative description of the targeted service. *Note: this definition is not part of the NEXOF-RA glossary. It has been derived from the Infraweb project.*

Service Location. Location provides the proper means to find and situate services. Service Location involves the integration of a set of subtasks such as *Service Crawling*, which collects information about Web Services from the Web; *Semantic Indexing* that allows data collected by the Service Crawling subtask to be analyzed and organized in a way that facilitates intelligent queries;

Service Discovery, which uses reasoning techniques for providing intelligent matching of requests and services; and last but not least *Service Ranking* and *Service Selection* for ordering services and then identifying which is the most suitable service according users needs

Service Monitoring: An activity that provides an awareness of the state of a service.

Service Provisioning or Service Supply: Service provisioning is the execution of a functionality offered by a service in order to satisfy a specific request from a consumer. It can be regulated by an agreement.

Service Publication: Any action to expose the service description.

Transaction: An agreement, communication, or movement carried out between separate entities or objects, often involving the exchange of items of value, such as information, goods, services and money.

2.4.6 Architecture-related terms

Autonomic System: An autonomic computing system is a system able to configure itself in the face of a changing environment.

Reference Architecture: A reference architecture is an architectural design pattern that indicates how an abstract set of mechanisms and relationships realizes a predetermined set of requirements.

Service-Oriented Architecture: Service-Oriented Architecture is an architectural style, based on services.

Service-Oriented Infrastructure: Service-oriented infrastructure results from applying the principles of service orientation to IT infrastructure. [TheOpenGroup 2007]

2.4.7 Terms related to technology

End Point: An association between a binding and a network address, specified by a URI, that may be used to communicate with an instance of a service. An end point indicates a specific location for accessing a service using a specific protocol and data format. [W3C 2004]

Enterprise Service Bus: An ESB is an integration platform that combines messaging, Web services, data transformation, and intelligent routing to reliably connect and coordinate the interaction of significant numbers of diverse applications across extended enterprises with transactional integrity. [Chappell 2004]. *Note: the NEXOF-RA definition is the following: An ESB is an standards-based iintegration platform that combines messaging, Web services, data transformation, and intelligent routing to reliably connect and coordinate the interaction of significant numbers of diverse applications across extended enterprises with transactional integrity. [Chappell 2004]*

Mashup: Mashup is a (Web) application that combines data from more than one source into a single integrated tool; an example is the use of cartographic data from “Google Maps” to add location information to real-estate data, thereby creating a new and distinct Web service that was not originally provided by either source.

2.4.8 BPM concepts

Business Process Description: Specifies an activity graph that includes a set of activities as well as their relationships.

Business Process Context: Defines a context in which a business has chosen to employ an information entity. [EBXML 2001]

Business Process Library: A repository of business process specifications and business information objects within an industry, and of common business process specifications and common business information objects that are shared by multiple industries. [EBXML 2001]

3. Principles of the Service Web Architecture

In this section the principles and rationale behind a service Web architecture are presented along with outlining how these principles will provide the means and methods for an internet-scale deployment and adoption of SOA infrastructures. First it begins by describing the SOA paradigm. Then the SOA principles are contrasted with the principles underlying Web, Autonomic computing, and the Semantic Web.

3.1 Service-Orientation Principles

Service-orientation provides a broad design paradigm that permits the separation of concerns and uses services as the basic building blocks of functionality. Service-oriented computing represents a new generation of distributed system that encompasses its own design paradigm and design principles, design pattern catalogs, pattern languages, a distinct architectural model, and a set of associated technologies and frameworks [Erl 2007]. Service-orientation provides a way of thinking about the design of a solution in terms of services, service-based development and the outcomes of those services.

As it has already been stated, the architectural model aims at enhancing efficiency, agility, flexibility and productivity by positioning services as the primary atomic functional elements. In the context of SOA4ALLs work, these services are classified according to:

- The functionality they provide within the architecture. These can be distinguished between business and middleware services. Business services (such as booking a hotel room) are services which various service providers supply through their back-end systems. Additionally, they are the subject of integration and interoperation within the architecture and can provide a certain value for users. On the other hand, middleware services (e.g. those that provide discovery and interoperability support) are lower level services that are used to facilitate the integration and interoperation of business services.
- The abstraction level within the architecture. Namely, these can be distinguished between Web services and services. A service is a general service that might take several forms when instantiated (such as purchasing a flight), whereas a Web service is an actual implementation of the service that is designed to support interoperable XML based machine-to-machine interaction over a network and that is consumed by and provides a concrete value for a user (such as the purchase of a particular flight from Innsbruck to Vienna).

All these classes of services should be designed and developed, according to a common approach. The SOA design paradigm captures a distinctive approach to the analysis, design, and implementation to all types of service-oriented IT environments, introducing a set of principles which govern aspects of communication, architecture, and processing logic. According to [Erl 2007] these design principles are: the Standardized Service Contract Principle; Loose Coupling Principle; Abstraction Principle; Reusability Principle; Autonomy Principle; Statelessness Principle; Discoverability Principle and Composability Principle. Each of these design principles is briefly explained in one of the following sections.

3.1.1 Standardized Service Contract Principle

In order to make the description of service capabilities understandable to any interested party, the properties of a service should be compliant with some design standard, namely the service contract. The service contract may include any information regarding the identification of the services (e.g. URL, name, textual description); functional properties, such as the type of the input/output parameters, interaction model; and non-functional properties, such as QoS, the location of the service, security constraints, etc.

Standardization supports the interpretability of services, resulting in an increase in the predictability of the service behaviour. The ability to predict the future behaviour of a service is a key mechanism to achieve scalability, since it allows the evaluation of the necessary computational resources

required to enact a specific service. This mechanism enables the intelligent provisioning of resources to prevent software resources running out.

3.1.2 Loose Coupling Principle

The Loose Coupling Principle states that the interface of a service should be decoupled or from (or loose coupled to) its consumers and the surrounding. Loose coupling, as presented in [Kayne 2003], intentionally sacrifices precision in the description of the interfaces of services for a greater good: the achievement of flexible interoperability among systems which are heterogeneous with respect to technology, location, performance, and availability. Loosely coupled applications aim to be more reusable and adaptable to new requirements.

Loosely coupled systems, such as event-driven systems [Eugster 2002, Luckham 2002] or space-based systems [Krummenacher 2007] have proven to be highly scalable when compared with tightly coupled systems.

3.1.3 Abstraction Principle

The Abstraction Principle dictates that the details of a given software artefact should be hidden where those details are not indispensable for others to effectively use it. Therefore, all the information necessary to invoke the service is contained in the service contract; and all the knowledge of the underlying logic, technology, etc. should be completely buried. This principle can be considered as a synonym to the old software engineering concept of black boxing.

The Abstraction Principle enables replaceability, which as outlined in [Armstrong 2003], combined with fault isolation and fault recovery, enhances scalability.

3.1.4 Reusability Principle

The Reusability Principle states that the functionality provided by services is as domain and context independent as feasible, in such way that facilitates their reuse [Erl 2007]. As a direct consequence of the application of this principle the logic of a service should be highly generic, independent from its original usage scenario. The Reusability Principle is a key enabler for SOA infrastructures, since it makes possible the creation of huge libraries of domain-independent services that leverage the construction of new complex context-dependent services.

3.1.5 Autonomy Principle

The Autonomy Principle states that services should be able to carry out their processes independently from outside influences. The only way to affect the results of a service should be through the modification of the input parameters as specified in the service contract.

Service autonomy increases reliability and more importantly predictability and fault isolation, which as presented in [Armstrong 2003] leads to an increase of the overall system scalability.

3.1.6 Statelessness Principle

The Statelessness Principle dictates that services should minimize resource consumption by deferring the management of state information to when strictly necessary [Erl 2007]. This notion of statelessness has been taken to the extreme in the REST architectural style [Fielding 2000], which has also been successfully applied to SOA in recent years.

Note that this principle affects more the service implementation aspects rather than the service design, at least at a conceptual level. Therefore it will not be further discussed in this document.

3.1.7 Discoverability Principle

The Discoverability Principle states that we should annotate services with metadata to enable services to be discovered by interested parties. This principle is closely related with the Standardized Service Contract Principle, since the discovery process could be performed using the information contained in the service contract.

3.1.8 Composability Principle

The Composability Principle identifies services as effective composition participants, regardless of the size and complexity of the composition [Erl 2007]. From a bottom-up perspective, we consider combining simpler services into larger services; from a top-down view, service composition is an effective way to tackle with the complexity of certain types of processes.

The Composability Principle is a core element within the definition of a service Web, since the ability to create new services easily, using existing ones, is a key pre-requisite to the widespread take-up of SOA.

3.2 The Web principles

The Web is based also on a collection of principles, identified below, that lead to a highly scalable means for electronic publication. The provision of Web-based lightweight integration infrastructures will facilitate openness and easy adoption for both the service provider and consumer. SOA4ALL should analyze and apply these principles to service-orientation, which will lead to a global, dynamically changing environment of services accessible for third-party usage, beyond the boundaries of single organizations. Within this environment, services will undergo many changes; and there will be a very high churn rate. For instance, users and resources will appear, disappear, and change location; resources can be initially free, and then transform to pay-per-use; and occasionally be blocked, out of service, etc.

The major principles we will incorporate to SOA4All from the Web are described in the following subsections.

3.2.1 Distributed Principle

The Distributed Principle is the process of aggregating several computing entities' power to collaboratively run a single task, transparently and coherently. Those entities appear as a single centralized system. Applying this principle to the middleware architecture will allow the transparent distribution of components over the network so that executing processes running in middleware can be scaled across numerous physical servers over a network. The distributed principle would also apply to business services, enabling running processes to span across enterprises distributed over a network.

3.2.2 Openness Principle

The Openness Principle states that a system should be easy to extend; in principle everybody should be able to contribute effortlessly to the system either as a provider or consumer of information. The usage of this infrastructure as a service provider or user must be as simple, smooth, unrestricted and even as possible. Openness is a major and essential necessity to ensure global adoption of a software environment.

3.2.3 Interoperability Principle

Interoperability should be provided through the integration of heterogeneous proprietary and legacy solutions through common interfaces based on standards where they exist. Interoperability on the Web is, at least in theory, platform and vendor neutral allowing all providers and requesters of information to participate on level playing field.

3.2.4 Human-centric Principle

The Human-centric Principle puts humans in the centre of the architecture. This principle is associated with concepts such as personalizing business services, facilitating service usability, promoting multichannel access and service delivery, building trust, and achieving efficiency, accountability, and responsiveness according to user requirements. It will also facilitate the seamless implementation of business processes across organizational boundaries.

Clearly this principle has been defined by considering the current Web reality that is constituted of a number of information services all accessible through a human oriented, Web-based interface. It

does not apply to those Web Services that are accessible through some programmatic interface.

Within SOA4All we envisage an environment in which services can be used and composed by everyone. In this context, it is of paramount importance to understand how to reinterpret this principle in a novel way.

3.3 Autonomic Computing principles

The concept of Autonomic Computing was first introduced in [Kephart 2003] where autonomous systems were characterized as self-manageable systems.

In the context of services, the idea of being able to self-manage is very important, especially when we think at systems that are created by composing services offered by third parties. Such system do not have any control on the way the component services are actually offered, Therefore, they should be equipped with some self-management capability that would allow them to react to the cases in which these services provide incorrect results or are unresponsive

As is identified in [Parashar 2006] the basic requisites for a self-manageable system are:

- **Knowledge aware**. The system should possess knowledge not only of its components, status, capacity, etc., but also of the context of its activity and those of other resources within the infrastructure.
- **Able to sense and analyze environmental conditions**. This includes both the ability to proactively take the pulse of individual components and services, looking for ways to improve its functions, and the ability to notice change and understand the implications of that change. The definition of environment here usually includes all that is important and has an effect on the execution of the system, but it is not explicitly defined as an input to the system.
- **Able to actuate on its environment**. The self-manageable system should be able to plan for and affect changes by altering its own state and effecting changes in other components of the environment.

As presented in [Parashar 2006], Autonomic Computing leverages the Web Services model to facilitate communication among heterogeneous components.

The characteristics of autonomous systems are being applied today in four fundamental areas of self-management to drive significant operational improvements where traditional manual-based processes are neither efficient nor effective. These four areas (depicted in Figure 6) are related to different attributes of autonomous systems, and they are self-configuring capabilities, self-healing capabilities, self-optimizing capabilities, and self-protecting capabilities.

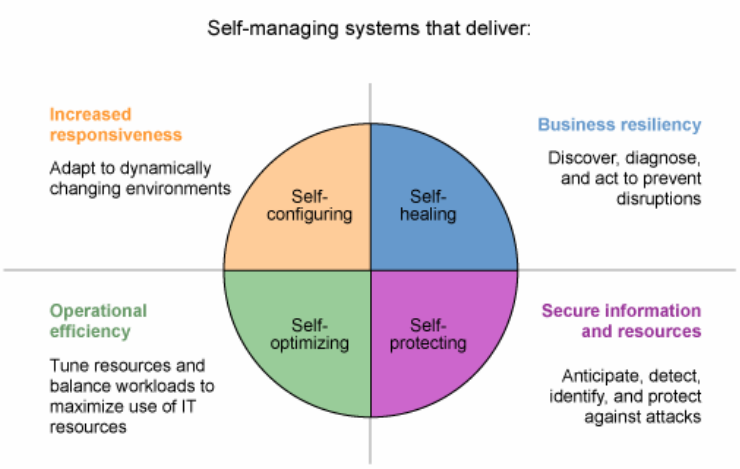

Figure 6 – Autonomic Computing Attributes

For each of these main areas of applicability, a design principle can be extracted where it is believed it should be incorporated in SOA4All in order to leverage the construction, configuration and deployment of infrastructures that enable Web scale service-oriented environments.

3.3.1 Self-healing Principle

According to the Self-healing Principle, computing systems should be able to detect, diagnose, and repair localized problems resulting from failures both in software and hardware. With this purpose, systems should analyze monitoring information (ranging from simple log files to more structured and complex provenance information). The system would then match the diagnosis against known software patches (or alert a human programmer if there are none), install the appropriate patch, and retest.

3.3.2 Self-configuration Principle

The Self-configuration Principle states that systems should configure themselves automatically in accordance with high-level declarative policies. These policies specify what is desired, not how it is to be accomplished [Kephart 2003]. The deployment, configuration, and integration of large, complex or highly changing systems is a challenging, time-consuming, and error-prone task even for experts. When a component is introduced in the environment, it should announce its capacity and should be incorporated seamlessly. The rest of the system should then adapt to its presence and be aware of its functionality. Within the context of service compositions, this would mean that any time a new service appears, compositions could reconfigure themselves to take advantage from the presence of the new service.

3.3.3 Self-optimization Principle

The system should continually seek ways to improve their operation, identifying and seizing opportunities to make themselves more efficient in performance or cost [Kephart 2003]. Thus this principle implies that the system should be able to monitor itself and should be able to carry out actions to tune its resources. These tuning actions could mean reallocating resources, such as in response to dynamically changing workloads, to improve overall utilization, or ensuring that particular business transactions can be completed in a timely fashion [Miller 2005]. This principle in the SOA4All context applies to service compositions. They should be able to show some degree of self-optimization is order to address the evolution of component services.

3.3.4 Self-protection Principle

The Self-protection Principle can be seen as two different but correlated facets, since it declares that the environment should exhibit:

- **Proactiveness**. The environment should anticipate problems based on early reports from sensors and take steps to avoid or mitigate them
- **Coordinated responsiveness**. The overall environment should react as a whole, They will defend the system as a whole against large-scale, correlated problems arising from malicious attacks or cascading failures that remain uncorrected by self-healing measures.

  These two facets in the SOA context refer to execution environment of service compositions.

3.4 Formal Semantic Descriptions

Current standards for describing Web services use syntactic (XML-based) notations such as WSDL. Because these descriptions are machine readable but not machine understandable, the semantics of Web services can only be interpreted by humans; thus IT personnel must carry out most of the tasks associated with creating and maintaining Web service-based applications. The requirement for specialist workers to be involved in all points in the Web service lifecycle causes numerous problems, the most significant of which are the impossibility to scale and the lack of responsiveness. Maintaining millions of services, let alone billions, to cope with environmental and context changes solely through human effort is simply not feasible.

Tasks such as Web service discovery, composition, and invocation can be automated to a great extent by applying semantic technologies (such as OWL-S [Martin 2004], WSMO [Roman 2006], WSDL-S [Akkiraiju 2005]). Semantics allow programs to access services through a machine-processable description of offered capability rather than as an endpoint. The use of semantics thus forms a scalable access layer over Web service data and processes. In the following we summarize some principles mostly derived from [Fensel 2007].

3.4.1 Ontology-based Principle

The mark-up of a Web service with formal descriptions makes them computer-interpretable, use-apparent and agent-ready [McIlraith 2001]. The combination of semantics with service-orientation allows us to define scalable, semantically rich, formal service models founded on ontologies. Ontologies are used as the data model meaning that all resource descriptions as well as all data interchanged during service usage are based on Ontologies.

The extensive usage of Ontologies to the modelling of service-based applications will facilitate the intelligent management and operation of SOA environments. Semantics will enable the management of categories of services as a whole; aiding the user in the visualization and update of services; facilitating the (semi) automation of service lifecycle activities, such as service discovery, contracting, negotiation, mediation, composition, and invocation; and enabling the advanced monitoring of execution and provenance analysis associated with the enactment of millions of services.

3.4.2 Centrality of Mediation

As a complementary design principle to loose coupling, mediation addresses the handling of heterogeneities that naturally arise in open environments. Heterogeneity can occur in terms of data, underlying Ontology, protocol or process. Mediation mechanisms should handle all these aspects by making sure that heterogeneous information and protocols/processes can still exist but it does not prevent service compositions from working properly.

3.4.3 Ontological Role Separation

Users, or more generally clients, exist in specific contexts, which will not be the same than those of available Web services. For example, a user may wish to book a holiday according to preferences for weather, culture and childcare, whereas Web services will typically cover airline travel and hotel availability. Thus, it is necessary to differentiate between the desires of users or clients and available services.

3.4.4 Independency of description with respect to implementation

The descriptions of semantic Web services elements should be independent of the executable technologies. While the former (the description) requires a concise and sound description framework based on appropriate formalisms in order to provide concise semantic descriptions, the latter (the implementation) is concerned with the support of existing and emerging execution technologies for the Semantic Web and Web services.

3.4.5 Problem Solving Principle

A SESA is a Semantically-Enabled SOA, that is, a SOA where all principles listed above hold. Indeed, the other main principle of SESA is the problem-solving principle, that is, the ability to discover and invoke services according to a goal-based approach. Users (service requesters) describe requests as goals. Such goals are expressed according to a semantic approach and are independent from services. SESAs are able to solve those goals through logical reasoning over their descriptions [Vitvar 2007].

4. Current technologies

This section will focus on technologies that the project considers most relevant to the service Web architecture. For the sake of brevity, each technology is only presented briefly.

4.1 Web Services

Web Services are software components that comply to two main standards, WSDL [WSDL 2007] and SOAP [SOAP 2007]. WSDL (Web Service Description Language) defines the syntactical structure of the programmatic interface offered by a Web service. In particular, it defines the operations offered by the Web service in terms of their parameters and return value. SOAP (Simple Object Access Protocol) is the communication protocol being used by Web services. It is an XML-based language, it prescribes the structure that all messages exchanged between a Web service and its consumers should comply to. Both WSDL and SOAP are independent of a specific programming language and of the transport protocol being used. In most cases, however, Web Services rely on HTTP as the transport protocol. This allows the communication to pass through firewalls. Another standard that is related to Web Services is UDDI (Universal Description Discovery & Integration) [UDDI 2004]. It defines how to communicate with a registry for publishing information about Web Services interfaces and for discovering services.

4.2 REST

Representational State Transfer (REST) is the name of an architectural style developed by R. Fielding as a formalization of the architectural principles underlying the World-Wide Web [Fielding 2000]. REST is composed of a number of constraints that ensure certain beneficial properties of the resulting architecture. These properties have made the Web scalable and evolvable, and it could have grown to the size and popularity it has today, without showing any signs of inherent barriers to future growth.

The Web is (was intended to be) an internet-scale distributed hypermedia system, i.e., a non linear way of presenting multimedia information (text, graphics, audio, video, etc.) associated by means of hyperlinks. This goal implies certain requirements, which affect REST as well. In particular, the Web needs to be:

- Simple, with a low barrier of entry, to attract users and developers
- Extensible, to be able to grow past the initial simplicity
- Distributed hypermedia, to be able to use the power of many internet hosts
- Anarchically scalable, to isolate performance issues of independent parts
- Independently deployable, both in terms of hosts and in terms of protocols and data formats, to allow gradual evolution and coexistence of old and new components
- Human-oriented, both optimized for better user experience, and tolerant of humans' erratic interactions with the system

The REST architectural style contains the following ingredients (which are themselves simpler architectural styles):

- **Client-server**. This style separates the concerns of the server (serving data, processing user inputs) from those of the client (user interface, presentation and interaction). This simplifies portability of the user interface even to platforms that would not support servers, and it also allows the components to evolve independently.
- **Layering**. While an actual system may consist of hierarchical layers that build one on another, the components are constrained only to see the immediate layers with which they interact. This restriction puts a bound on the overall system complexity and promotes component independence, while adding overhead and latency to the interactions, which are

mitigated by the increasing performance of computers and networks.

- **Stateless communication**. “Each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server.” [Fielding 2000] To understand this constraint, the state of an application must be separated into the state of resources on the server, and the state of the interaction (also known as the session) between the client and the server. This constraint adds communication overhead (again, mitigated by increased performance of modern systems), and it makes servers relinquish some of the control over how the application is behaving. On the positive side, stateless communication improves the scalability of servers (by freeing their resources between requests) and the reliability of applications (by simplifying the task of recovering from partial failures), as discussed in [Richardson 2007].
- **Uniform interface**. All the components in a RESTful system must support a single uniform interface. In particular, HTTP's uniform interface consists of basic methods (GET to retrieve Web pages, POST for submitting data to a resource, etc.). With a single interface, a Web browser can access any Web resource, and there is no need for specialized browsers for different resources; implementations are decoupled from the applications. REST uniform interface is optimized for large-grain hypermedia data transfer, which is not necessarily efficient for all applications.
- **Caching**. To improve network efficiency and server scalability, components on the Web are allowed to cache responses marked as cacheable. ISPs (Internet Service Providers) and organizations may deploy large caches to lower the bandwidth used by the users of the Web; but also the client browser incorporates a cache to improve the perceived performance. Caching introduces the potential problem of data inconsistency, but the human users of the Web handle this problem easily.
- **Code on demand**. Finally, REST allows client functionality to be extended by downloading and executing code from the server. This is typically scripts inside Web pages (most commonly in Javascript), or as embedded programs such as applets and Flash programs. By allowing code-on-demand, the client software only needs to implement a reduced set of required features. A common example of user interface extensions through code-on-demand is the Web 2.0 wave of AJAX Web sites.

These constraints are all applied to the architecture of the Web, as embodied mainly in the HyperText Transfer Protocol HTTP. Nevertheless, some of these constraints cannot be easily enforced, and it is common for Web sites to break some of them (most notably, the stateless communication constraint is often broken by using *cookies* for session maintenance),.

In addition, the hypermedia aspect of the Web leads to a further pair of requirements, which affect Web architecture, especially in the area of document formats:

- **Links and connectedness**. Resources on the Web must be able to refer (link) to other resources, the user must be allowed to navigate the resulting graph of links freely.
- **Addressability**. Stemming from the requirement for links, it is necessary that all resources are addressable. For this, REST uses URIs (Uniform Resource Identifiers).

REST was designed with the human-oriented Web in mind; however, the constraints can also be applied to machine-oriented Web services. An automated, machine-oriented Web application or service is said to be *RESTful* when it uses the uniform interface (using all the methods as appropriate), when its communication is stateless, and when it enables cacheability.

In contrast to RESTful Web services, traditional SOAP-based Web services commonly only use the POST method, use transient messages that are not cacheable, and keep conversation sessions between the server and the client. These violations lead towards tighter coupling between the client and the service, and limit interoperability and scalability of the resulting systems.

4.3 Enterprise Service Bus

The key item for integration of services within a SOA is the Enterprise Service Bus (ESB). The goal of an ESB is to provide virtualization of the enterprise resources, allowing the business logic of the enterprise to be developed and managed independently of the infrastructure, network, and provision of those business services.

Enterprise Service Bus (ESB) is a new class of integration tools. It supports integration and is also characterized by transformation and routing functions. More in-depth descriptions of the ESB concept are available in [Chappell 2004, Craggs 2005].

Originally, first commercial ESB products were described both as a way to integrate existing middleware services (J2EE application servers, message-oriented brokers, etc.) and products (e.g., B2B solutions) and to connect applications with the required protocols. More recently, since the advent of the SOA approach, ESBs have also been presented as a way to create a SOA.

ESBs clearly face two major challenges:

- How to integrate heterogeneous technology and products, possibly produced by separated vendors, in a way that is appropriate with respect to the size of each particular integration problem?
- How to provide the required features to fully address the specifics of SOA needs?

The Java business Integration (JBI) standard seeks to address the first challenge by adopting the SOA principles. An ESB is built from JBI containers that can be an integration framework, a host for Java connectors, an XSLT engine, a mediation engine, a service orchestrator, etc. JBI maximizes the decoupling between the JBI containers that all provide and consume services. The ESB links the containers together, allowing them to interact.

Currently, it turns out that JBI-compliant ESBs are mostly open-source ESBs that aim at promoting highly configurable and made-to-measure ESBs in order to better fit business needs.

The JBI specification has been standardized by the Java Community Process (JCP) expert group. The JBI specification promotes a plug-in based architecture which enables the creation of tailored integration solutions by putting together the best-of-breed integration components. One of the main interest for using JBI compliant software in an ESB is that it is based on Web Services best practices (WSDL usage, loose coupling, XML messaging).

Companies are currently struggling with the second challenge, as they realize that an ESB vendor's solution does not fit their needs. The reasons are manifold: for example, the ESB does not provide management models to control and enforce QoS at different levels and track consumer usage; it does not fit into existing management and security frameworks; it is unable to connect to or evolve toward a B2B architecture. This problem will still be open, as long as SOA technology editors do not address the immediate and long-term business needs, and concrete functional SOA.

4.4 WS Policy

The Web Service Policy Framework[16] (WS-Policy) defines a syntax and semantics for service providers and service requestors to describe their requirements, preferences, and capabilities.

A policy is defined as a collection of policy alternatives where each alternative is a collection of policy assertions. An assertion is a basic unit of policy. It is used to represent a requirement, capability, or a behaviour of a Web Service. A policy assertion specifies characteristics which are critical for selecting and using the Web Services, for instance contextual properties. An assertion can include an arbitrary number of child assertions and attributes. To facilitate interoperability, WS-Policy defines a normal form for policy expressions which is a straightforward XML Infoset

[16] http://www.w3.org/Submission/WS-Policy/

representation of a policy, enumerating each of its alternatives that in turn enumerate each of their assertions. An example of a WS Policy assertion is the need for credentials expressed as username and password for accessing a resource.

A policy is built up using assertions and nested combinations of operators and attributes. Policy syntax is used to describe acceptable combinations of assertions to form a complete set of instructions to the policy processing infrastructure, for a given Web Service invocation. Each set of assertions is termed an alternative.

When applied in the Web Services model, policy is used to convey conditions on an interaction between two Web Service endpoints. Satisfying assertions in the policy usually results in behaviour that reflects these conditions. Typically, the provider of a Web Service exposes a policy to convey conditions under which it provides the service. A requester might use this policy to decide whether or not to use the service. A requester may choose any alternative since each is a valid configuration for interaction with the service, but a requester must choose only a single alternative for an interaction with a service since each represents an alternative configuration.

4.5 SAWSDL

Semantic Annotations in WSDL and XML Schema [Farrel 2007] is a W3C specification that defines how to add semantic annotations to Web Service Description Language and to XML Schema [XMLSchema 2004]. It defines extension attributes that can be applied to elements in both WSDL and XML Schema in order to annotate WSDL interfaces, operations and their input and output messages. SAWSDL is the first step towards standardization in the area of Semantic Web Services.

Semantic annotations in WSDL and XML Schema are used for these purposes:

- Associating WSDL interfaces with some taxonomical categories to help semantic Web service discovery,
- Describing the purpose or applicability of WSDL operations to help discovery or composition,
- Linking and mapping inputs, outputs and faults of WSDL operations to semantic concepts to help facilitate mediation and service discovery and composition.

While the semantic annotations are used to point to taxonomies, ontologies or mappings, SAWSDL is independent of any particular ontology language or mapping language. The mechanism only requires that the concepts in the semantic models can be identified with URIs.

SAWSDL can be split in two parts: semantic model references from elements in WSDL or XML Schema to concepts in a semantic model (usually an ontology or taxonomy), and data mappings between XML and semantic models. A more detailed presentation of these two aspects can be found in Deliverable D1.2.1 “WSMO grounding in SAWSDL” [Kopecký 2008].

4.6 WSMO

The Web Service Modeling Ontology [Roman 2006] is a conceptual model for describing the semantics of Web services and related entities, for the purpose of automating some aspects of service discovery and usage. WSMO is complemented by the Web Service Modeling anguage [Bruijn 2005], a concrete language that implements the conceptual model and fleshes out the details. In the following, all statements about WSMO also apply to WSML.

WSMO has four main building blocks:

- *Ontologies* for knowledge representation
- *Web services* represent the semantics of services
- *Goals* represent user requests that can be satisfied with services
- *Mediators* represent components that bridge any heterogeneities

Ontologies and ontological instances (data) are used in all the other parts of WSMO. For

representing ontologies with varying levels of expressivity and reasoning complexity, WSML provides several *fragments*: WSML-Core allows basic modeling supported by most knowledge representation technologies. Figure 7 illustrates the relationships between the various fragments of WSML.

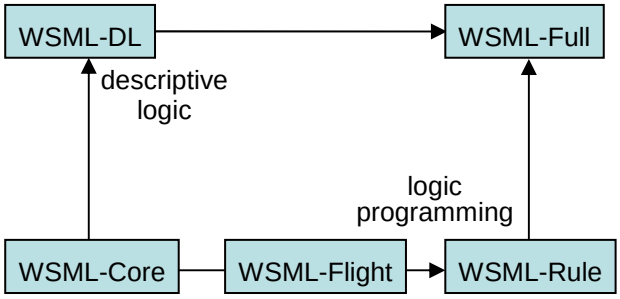

Figure 7. WSML Ontology Language Fragments.

WSML-Flight and WSML-Rule extend WSML-Core with techniques of Logic Programming for advanced reasoning with axioms and rules. WSML-DL extends WSML-Core with modeling constructs coming from Descriptive Logic. And finally, WSML-Full unifies both branches (Logic Programming and Descriptive Logic) and thus provides the most expressive language.

Mediators are used where different descriptions express similar meaning differently. For instance, ontology mediators can be used to import OWL ontologies into WSML, or to map between different, yet semantically overlapping terminologies.

Finally, goals and Web services describe what users want and what Web services provide. In WSMO, descriptions of goals and Web services have the same structure, therefore in the following, we will only talk about Web service descriptions, in terms of what a service *provides*, and the reader may also read it in terms of what a client *requests*.

A Web service description captures the *functional semantics* (the *capability*) of a Web service, i.e. what the service does, and the *behavioral semantics* (the *interface*), i.e. how the service communicates with other parties.

Functional semantics are expressed with a *capability* to construct that specifies the preconditions and assumptions that must be valid before the service can be used, and the post-conditions and effects which are expected to be valid after the service is successfully invoked.

The behavioral semantics part of the description of a Web service has two aspects: external behavior called *choreography*, i.e. how the clients talk to the service, and internal behavior called *orchestration*, i.e. how the service uses other services to implement its functionality.

Figure 8 illustrates the structure of WSMO Web service (and goal) descriptions:

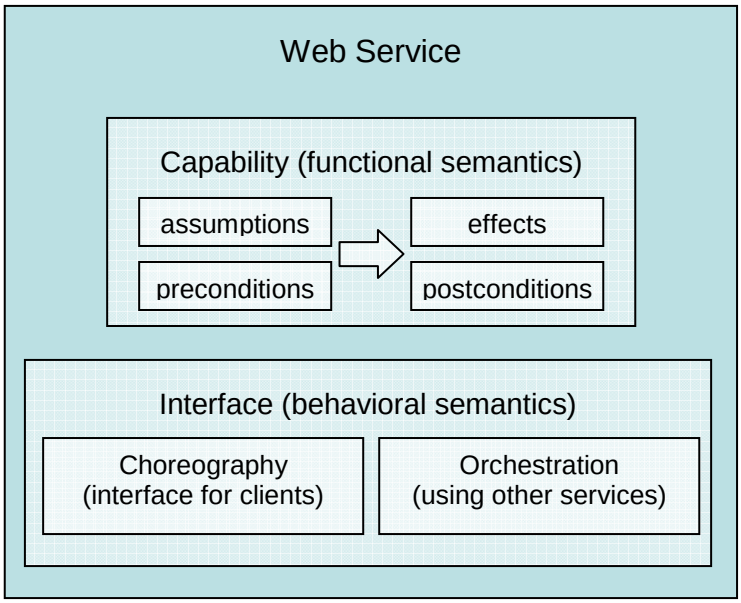

Figure 8. Structure of WSMO Web Service Description.

A WSMO choreography is a state machine, with its states described using ontologies, in terms of concepts, their instances and the relations between them. Inputs and outputs of the Web service are represented as instances of certain concepts that can be read or written by the client communicating with the service. Each concept in the choreography state ontology can be assigned to a role which determines whether the clients may read or update (or both) the values of instances of that concept. In WSMO these concepts and instances are called *accessible*.

The accessible concepts must be available to the client using some underlying messaging protocol. Therefore, a choreography definition includes *grounding*, which defines the protocol for reading and writing of the accessible concepts by the clients; in other words, the grounding specifies how the client may communicate with the service.

4.7 BPM techniques: BPML, BPEL

Business Process Management (BPM) is an IT-enabled management discipline that treats business processes as assets to be valued, designed and enhanced in their own right. BPM technologies support both human-centric processes (e.g., claims processing, accounts payable or customer servicing) and system-intensive processes (e.g., straight-through processing or trade settlement), as well as a mixture of both (e.g., loan granting) [Di Nitto 2008].

SOAs represent one of the most relevant approaches for building IT systems supporting BPM. Thus, various initiatives have been started to make business processes easily translated in some executable languages able to compose and coordinate various services. In this context, two of the most well-known initiatives are BPML and BPEL.

The Business Process Modelling Language (BPML) is a standard developed and promoted by BPMI.org (the Business Process Management Initiative) [Curbera 2002]. BPML can be seen as a language competing with other workflow description standards such as IBM's WSFL (Web Services Flow Language) and Microsoft's XLANG (Web Service for Business Process Design) which recently merged in to BPEL4WS (Business Process Execution Language for Web Service). After BPMI.org merged with the OMG, BPMI.org finally decided to drop BPML in favour of BPEL4WS.

BPEL4WS (in short BPEL) was proposed by BEA, IBM and Microsoft. In July 2002, the first version of BPEL was published [Curbera 2002]. Subsequently, SAP and Siebel joined the effort and the

second version of BPEL [Andrews 2003] was published in May 2003. There are also other versions of BPEL: Websphere Integration Developer version 6.0 (informally WebSphere BPEL), Oracle BPEL and so on. The latest version of BPEL has been described in [Arkin 2005]. Many major vendors of business solutions have joined the Web Services Business Process Execution Language Technical Committee (WSBPEL TC), including Adobe, Hewlett-Packard, NEC, Oracle and Sun.

BPEL is also known as a Web services flow language, Web service execution language, Web service composition language, Web service orchestration language and Web-enabled workflow language. Web services composition languages, such as BPEL, build directly on top of Web Service Description Language (WSDL). BPEL can provide and/or use one or more services described by means of WSDL. A Web service composition language can glue composed services together into a process model. BPEL provides the means to specify such a process model in an executable manner. An important difference between WSDL and a Web service composition language is revealed when considering the states. WSDL is in essence stateless while Web service composition languages such as BPEL record states for processes, but don't describe the interfaces for the individual webservices.

BPEL combines the features of a block structure language inherited from XLANG with those for directed graphs originating from WSFL [van der Aalst 2003]. The language is intended to support the modelling from two types of processes executable and abstract processes. An *abstract*, (not executable) *process* is a business protocol, specifying the message exchange behaviour between different parties without revealing the internal behaviour for any one of them. An *executable process* specifies the execution order between a number of activities constituting the process, the *partners* involved in the process, the *messages* exchanged between these partners, and the *fault* and *exception handling* specifying the behaviour in case of errors and exceptions.

5. Challenges for the SOA4All Architecture

The principles presented in Section 3 are very high level and can be addressed from various points of view, using various technologies, including those that we have briefly summarized in Section 4. In the current section we focus on the specific challenges that are concerned with the SOA4All service Web architecture and discuss how these challenges relate to the general principles of Section 3.

The challenges discussed below have been preliminarily derived from the main SOA4All objectives. They will be integrated in the next deliverables with those challenges that will emerge from the analysis of the requirements that are being defined for the various case studies that will be developed during the project. Following each of the challenge, a summary table,Table 1, summarizes the challenges in this section and their mapping to the general principles.

5.1 Heterogeneity

It should be assumed that worldwide distributed systems contain many different kinds of hardware / software systems and environments. In particular, distributed service-based systems can contain different kinds of services possibly built using different standards (for instance, in Section 4 Web and REST services have been presented).

Thus, a service Web infrastructure should be able to handle such heterogeneity. To do so, it has to implement some of the principles that have discussed in Section 3, and, in particular, *the standardized service contract*, *loose coupling*, and *abstraction* principles of SOAs as well as the *openness* and *interoperability* principles of Web and the *independency of descriptions with respect to implementations* principle of semantic descriptions. Fully addressing these principles, in fact, allow consumers to interact with services regardless the standard they adopt.

When the service Web infrastructure is also able to handle heterogeneity at the semantic level it becomes much more powerful. In this case it is implementing the centrality of mediator principle and it is highlighting the importance of the role of mediator as the one that keeps the mapping between different semantic descriptions.

5.2 Worldwide access mechanisms

Services should be accessible worldwide. This means that they should be identifiable in a unique way and should be invoked despite potential heterogeneity.

Thus, if a wide-spread service Web infrastructure is to be realized, a worldwide communication infrastructure able to deliver the following is required:

- **A global addressing schema**, that allows each service to be addressed in a unique way, regardless the device that is hosting it. In its simplest form, this may be a unique name and, more elaborately, a description of the capability of a service (that is, the degree to which it can be used to achieve a certain goal).
- **A transport layer** to transmit requests for service invocations and the results from them.
- **A platform-independent interface**. This commonly accepted interface should process service requests and access to service implementations. Of course issues such as efficiency should be properly addressed since, as pointed out in [Fielding 2000], a uniform interface may degrade efficiency while gaining in standardization.

The main principles that are concerned with this challenges are, once again, *standardized service contract*, *abstraction*, and *openness*. Indeed, the implementation of the *ontology-based* principle allows for sophisticated ways of describing services and identifying them within the Web.

5.3 Semantic provisioning of services

As suggested by the ontology-based principle, formal semantic descriptions of services allows

powerful reasoning and precise matching of requests with services. However, formal semantic introduces a relevant computational overhead that has to be taken into account for usage at runtime. In the future new processor architectures with higher computational capabilities will solve this problem.

Thus, until the computational overhead will represent a problem, some lightweight approaches should be studied. They represent a compromise in the trade of between expressive power and computational speed. The challenge is to understand to what extent we can simplify semantic languages without losing too much of their capabilities. An example of a lightweight approach is SAWSDL that has been discussed in Section 4.5.

Another issue that requires special care from the semantic description perspective is the coexistence in the service Web of software services and human services. Thus, we need to understand how human services are described and how they are accessed and provided.

Of course, semantic provisioning of services addresses all principles that are concerned with semantics, but it also implements the *human-centric* principle. As is discussed above, people can be those who execute services. Even in their usual role of service users, they should be put at the centre of the world and supported in all their activities concerned with the identification, analysis, and selection of services. In all these activities, of course (lightweight) semantic approaches can introduce significant simplifications for people.

5.4 Decentralized dynamicity and adaptability

Services can appear, be modified, or disappear in an ad hoc fashion. Thus, it should be possible to control the life cycle of services and to handle their dynamicity by offering proper mechanisms that enable the adaptation of those systems that exploit these dynamic services (see for instance [Colombo 2006]). Adaptation usually is concerned with the possibility of replacing on the fly a service with another similar one that can be identified and selected during the execution of the system.

Of course, a central control on the life cycle of all services would hamper access and therefore scalability. Thus, their provisioning and modification should be completely decentralized and unconstrained, without hampering the possibility of building solid service compositions out of them.

Addressing this challenge means to implement most of the principles that have been outlined in Section 3. Note though the important role of the *autonomic* principles that are those that enable and drive the possibility of self-adapting a system based on the status of the services it is using and of all those that could replace them.

5.5 Matching requests and services

Even if services are accessible worldwide, without proper support for matching requests and services it may be difficult for a service consumer to find the right service to use. Thus, proper matching mechanisms need to be provided. While so far the literature has focused on centralized matchmakers, the real challenge is to distribute the execution of matching algorithms on multiple nodes. In [Cugola 2008] an approach to enable such matching by using a distributed publish/subscribe infrastructure is presented. The approach exploits content-based routing to route requests towards those services that can fulfil them. This routing approach is based solely on syntax-based properties of requests and services. The challenge here is to explore the use of the additional information provided by the semantic-based description of services in order to create techniques that provide smarter routing, which will result in a highly scalable and reactive smart semantic middleware.

Many principles are related to this challenge. Among the others, the *discoverability* principle and the *problem solving* principle appear to be the most relevant. Of course, matching addresses the discoverability principle providing proper discovery techniques. Indeed, it also addresses the problem solving principles in all cases in which the request is expressed in terms of high-level goals that then need to be associated with specific service descriptions.

5.6 Enabling n:m asynchronous interactions

The classic client-server model of interaction no longer reflects the nature of the Web. Thus, we should introduce richer models of interaction to address situations where multiple entities collaborate by playing different roles, each of them sending and receiving complex messages. More importantly, the role (requestor or responder) that each entity plays in those interactions might be interchanged.

It is believed that a publish/subscribe approach could suit the requirements since it offers multicast communication. Also, it allows the interacting party to remain anonymous thus guaranteeing a high level of loose coupling.

Besides the *loose coupling* principle, publish/subscribe also addresses the *distribution* principle and, indirectly, the *composability* principle. It, in fact, enables a new kind of composition approach where the composition logic can be decentralized in the various interacting peers instead of being centralized in a single component as it would happen when using a normal BPEL engine (see Section 4.7).

5.7 Enabling service prosumers

From a purely technological viewpoint, the mechanisms used in Web 2.0 are similar to the "standard" Web. However, Web 2.0 brings a number of Web-related concerns to the fore which, when incorporated into SOA, will be an important ingredient of a Service Web. In particular, within Web 2.0 the provision of content has been democratized – in contrast to the standard Web where users are considered to be passive spectators of read-only content. Active consumers (often referred as prosumers) become part of the content providing process and often even form democratic communities of creators.

Applying this idea to services is not as simple since to date the development of services has been an activity for specialists. Therefore, the challenge is to understand the kinds of tools that should be offered to users in order to transform them in service prosumers.

Clearly, this challenge is related to the *human-centric* and to the *problem solving* principles, but it also has an impact on the *discoverability* and *composability* principles since the ultimate aim of prosumers is to build new services by discovering and composing those that already exist. In many cases, the point of view of a prosumer will be different than the point of view of who has developed a certain service. Thus, the *ontological role separation* principle will have to be addressed in order to cover this viewpoint mismatch.

5.8 Supporting both machine and human-based computation

Section 5.3 has highlighted the role services operated by humans could have. Indeed, incorporating human interaction and cooperation in a comprehensive fashion creates a route to solving tasks such as service ranking and mediation, which otherwise remain computationally infeasible. In several scenarios, Web 2.0 and human computing approaches, together with their underlying social consensus-building mechanisms, have proven the potential of combining human-based services with services provided via automated reasoning. As we have already discussed, the transparent provisioning of services abstracting over whether the 'engine' is a human or machine will significantly increase the overall quality of services available to the end-user. In the end, a service need not necessarily by supplied by a computer program, enabling for example, current approaches to service discovery and (human) expert finders to be combined.

From this perspective the challenge is, therefore, to see humans as being part of our service Web infrastructure. This requires proper user interfaces and mechanisms for standard services to human services communication.

Several principles are concerned with human-based computation. In particular, the *discoverability*,

composability, and *problem solving* principles are seen here from a new perspective. Not only machines but also people can discover and compose services or solve problems. The implementation of such principles in the service-based architecture has to take this consideration into account.

Table 1. Challenges and principles.

	Heterogeneity	Worldwide access mechanisms	Semantic provisioning of services	Decentralized dynamicity and adaptability	Matching requests and responses	Enabling n:m asynchronous interactions	Enabling service prosumers	Supporting both machine and human-based computation
SERVICE-ORIENTATION PRINCIPLES								
Standardized Service Contract Principle	X	X						X
Loose Coupling Principle	X			X	X	X		X
Abstraction Principle	X	X			X			
Reusability Principle				X			X	
Autonomy Principle				X				X
Statelessness Principle								
Discoverability Principle				X	X		X	X
Composability Principle				X		X	X	X
THE WEB PRINCIPLES								
Distributed Principle				X		X		
Openness Principle	X	X		X	X			
Interoperability Principle	X				X			X
Human-centric Principle			X				X	X
AUTONOMIC COMPUTING PRINCIPLES								
Self-healing Principle				X				
Self-configuration Principle				X				
Self-optimization Principle				X				
Self-protection Principle				X				
FORMAL SEMANTIC DESCRIPTION PRINCIPLES								
Ontology-based Principle		X	X	X	X		X	
Centrality of Mediation	X		X	X	X			
Ontological Role Separation		X	X		X		X	
Independency of descriptions with respect to implementations	X		X	X				
Problem Solving Principle			X	X	X		X	X

6. Conclusion

This deliverable tries to set the stage for the SOA4All service Web architecture by adopting a proper terminology, mostly inherited by the NEXOF-RA glossary, defining the main general principles for the service Web, shortly presenting some interesting technologies, and, finally, identifying some challenges for the development of the SOA4All architecture.

We have considered as principles those defined in the fields that can be considered the main pillars of the SOA4All project, in particular, Web services, Web, and semantic Web. We have also discussed the principles that are the basis of autonomic computing since we think that some of them should be addressed by the SOA4All architecture in order to enable the development of flexible service compositions that are able to self-adapt to the external situation in which they run.

While discussing the challenges that we have identified, we have tried to relate them to the principles that could help in addressing them.

The work presented here will be incrementally extended and completed during the project. In particular, the next step will be to review the challenges that will be driving the development of the SOA4All architecture in the light of the requirements that are being defined by the owner of use cases.

7. References

1. [Akkiraju 2005] Akkiraju R., Farrell J., Miller J., Nagarajan M., Schmidt M.T., Sheth A., Verma K., WSDL-S Technical NoteVersion 1.0 Web Service Semantics, available at: http://www.w3.org/Submission/WSDL-S/, 2005
2. [Andrews 2003] Andrews, T. Curbera, F. Dholakia, H. Goland, Y. Klein, J. Leymann, F. Liu, K. Roller, D. Smith, D. Thatte, S. Trickovic, I. and Weerawarana. S. (2003) Business process execution language for Web services, version 1.1, May 2003.
3. [Arkin 2005] Arkin, A. Askary, S. Bloch, B. Curbera, F. Goland, Y. Kartha, N. Liu, C.K. Mehta, V. Thatte, S. Yendluri, P. Yiu, A. and Alvesa, A. (2005) Web services business process execution language, version 2.0, December 2005
4. [Armstrong 2003] Armstrong J., *Making reliable distributed systems in the presence of software errors*. PhD thesis, Royal Institute of Technology, Swedish Institute of Computer Science (SICS), Stockholm, December 2003.
5. [Brickley 2004] Brickley D., Guha R.V., (Eds.) RDF Vocabulary Description Language 1.0: RDF Schema W3C Recommendation 10 February 2004 available at: http://www.w3.org/TR/rdf-schema/
6. [Bruijn 2005] Jos de Bruijn (editor): The Web Service Modeling Language WSML, version 0.21 available at http://www.wsmo.org/TR/d16/d16.1/v0.21/.
7. [Craggs 2005] Steve Craggs, "Best-of-Breed ESBs - Making your choice", Global Integration Summit, May 23-25, 2005, Banff, Canada.
8. [Chappell 2004] Dave Chappell, "Enterprise Service Bus: Theory in Practice", O'Reilly Media, June 2004 (1st edition).
9. [Colombo 2005] Colombo, M., Di Nitto, E., Di Penta, M., Distante, D., and Zuccalà, M.. Speaking a Common Language: A Conceptual Model for Describing Service-Oriented Systems, International Conference on Service Oriented Computing (ICSOC 2005), December 2005.
10. [Colombo 2006] Colombo, M., Di Nitto, E., Mauri, M.: SCENE: A Service Composition Execution Environment Supporting Dynamic Changes Disciplined Through Rules. ICSOC 2006: 191-202.
11. [Cugola 2008] Cugola, G. and Di Nitto, E. 2008. On adopting Content-Based Routing in service-oriented architectures. Inf. Softw. Technol. 50, 1-2 (Jan. 2008), 22-35.
12. [Curbera 2001] Curbera, F.; Nagy, W.A.; and Weerawana, S., “Web Service: Why and How?”, In Proceedings of the OOPSLA-2001 Workshop on Object-Oriented Services. Tampa, Florida, 2001.
13. [Curbera 2002] Curbera, F. Goland, Y. Klein, J. Leymann, F. Roller, D. Thatte, S. and Weerawarana. S. (2002) Business process execution language for Web services, version 1.0, July 2002.
14. [Di Nitto 2008] Di Nitto, E., Ghezzi, C., Metzger, A., Papazoglou, M., Pohl, K., A Journey to highly dynamic, self-adaptive service-based applications, to appear on the Automated Software Engineering Journal, 2008.
15. [EBXML 2001] Team, T. A. ebXML Glossary 2001.
16. [Erl 2007] Erl T., *SOA Principles of Service Design*, the Prentice Hall Service-Oriented Computing Series from Thomas Erl, 2007.
17. [Eugster 2002] Eugster P.Th., Felber P., Guerraoui R., Handurukande S.B., “Event systems. How to have your cake and eat it too”, 22nd International Conference on

Distributed Computing Systems Workshops (ICDCSW '02), 2002.

18. [Farrell 2007] Farrell J., Lausen H. (Eds.), Semantic Annotations for WSDL and XML Schema W3C Recommendation 28 August 2007, available at: http://www.w3.org/TR/sawsdl/.
19. [Fensel 2007] Fensel, D., Lausen, H., Polleres, A., De Bruijn, J., Stollberg, M., Roman, D., Domingue, J. *Enabling Semantic Web Services: Web Service Modeling Ontology*. Springer, 2007.
20. [Fielding 2000] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures.* PhD thesis, University of California, Irvine, 2000. Chair: Richard N. Taylor.
21. [Foster 2004] Foster I., Frey J., Graham S., Tuecke S., Czajkowski K., Ferguson D., Leymann F., Nally M., Sedukhin I., Snelling D., Storey T., Vambenepe W., Weerawarana S, *Modeling Stateful Resources with Web Services,* downloadable at: http://www.ibm.com/developerworks/library/ws-resource/ws-modelingresources.pdf, Version 1.1, 2004.
22. [Gruber 1993] Gruber, T. R. A translation approach to portable ontology specifications Knowl. Acquis., Academic Press Ltd., 1993, 5, 199-220.
23. [Kayne 2003] Kayne D., Loosely Coupled, The Missing Pieces of Web Services Rds Associates Inc, ISBN: 1881378241, 2003.
24. [Kephart 2003] Kephart, J.O. and Chess, D.M., "The vision of autonomic computing", IEEE Computer, January 2003, 36:1, pages: 41-50.
25. [Kerrigan 2008] M. Kerrigan, B. Norton, A. Mocan (Editors): "Reference Ontology for Semantic Service Oriented Architectures", OASIS SEE (Semantic Execution Environment) Technical Committee specifications, Release Candidate, June 2008. Available at http://www.oasis-open.org/apps/org/workgroup/semantic-ex/document.php?document_id=27923.
26. [Kopecký 2008] Jacek Kopecký, Adi Schütz, 1.2.1 WSMO grounding in SAWSDL, SOA4All Deliverable D1.2.1 2008.
27. [Kreger 2001] Kreger, H. *Web Services Conceptual Architecture*, downloadable at: http://www.ibm.com/software/solutions/webservices/pdf/WSCA.pdf 2001.
28. [Krummenacher 2007] Krummenacher R., Simperl E. and Fensel D.: Towards Scalable Information Spaces. Workshop on New forms of reasoning for the Semantic Web: scaleable, tolerant and dynamic, ISWC 2007.
29. [Luckham 2002] Luckham D., *The Power of Events: An Introduction to Complex Event Processing* in Distributed Enterprise Systems, Addison-Wesley Professional, 2002.
30. [MacKenzie 2006] C. M. MacKenzie, K. Laskey, F. McCabe, P. F. Brown, R. Metz (eds.): Reference Model for Service Oriented Architecture 1.0, OASIS SOA-RM Technical Committee Specification, 2 August, 2006, available at: http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf.
31. [Margolis 2007] Margolis, B. SOA for the Business Developer: Concepts, BPEL, and SCA (Business Developers series) Mc Press, 2007
32. [Martin 2004] Martin D., Burstein M., Hobbs J., Lassila O., McDermott D., McIlraith S., Paolucci M., Parsia B., Payne T., Sirin E., Srinivasan N., Sycara K., "OWL-S 1.1 Release: Semantic Markup for Web Services", available at: http://www.daml.org/services/owl-s/1.0/owl-s.pdf, 2004.

33. [McIlraith 2001] McIlraith S., Son T.C., and Zeng H., (2001) Semantic Web Services. IEEE Intelligent Systems, 16(2):46–53.
34. [Miller 2005] Miller B., The autonomic computing edge: Can you CHOP up autonomic computing?, available at: http://www.ibm.com/developerworks/autonomic/library/ac-edge4/, 2005.
35. [Parashar 2006] Parashar M., Hariri S., (Eds.) *Autonomic Computing: Concepts, Infrastructure, and Applications*, 1st Ed., CRC, 2006.
36. [Pohl 2005] Pohl, K.; Böckle, G. & van der Linden, F. Software Product Line Engineering. Foundations, Principles, and Techniques. Springer, Berlin, 2005.
37. [Richardson 2007] Leonard Richardson and Sam Ruby. *RESTful Web Services.* O'Reilly Media, May 2007.
38. [Roman 2006] Dumitru Roman, Holger Lausen, Uwe Keller (editors): Web Service Modeling Ontology (WSMO), version 1.3 2006 available at http://www.wsmo.org/TR/d2/v1.3/.
39. [SeCSE 2007] SeCSE consortium, Refined conceptual model, Project deliverable A5.D9.2, 2007, available at http://secse.eng.it.
40. [SOAP 2007] SOAP Version 1.2 Part 1: Messaging Framework (Second Edition), Recommendation, W3C, April 2007. Available at http://www.w3.org/TR/2007/REC-soap12-part1-20070427/.
41. [TheOpenGroup 2007] The Open Group: Service-Oriented Infrastructure Project Description, Version 1.1, July 2007.
42. [UDDI 2004] UDDI Version 3.0.2, OASIS UDDI Spec Technical Committee Draft, October 2004. Available at http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm.
43. [van der Aalst 2003] van der Aalst, W.M.P. Don't go with the flow: Web Services composition standards exposed, IEEE Intelligent, February 2003.
44. [Vitvar 2007] Tomas Vitvar, Michal Zaremba, Matthew Moran, Maciej Zaremba, Dieter Fensel: SESA: Emerging Technology for Service-Centric Environments. IEEE Software 24(6): 56-67 (2007).
45. [W3C 2004] W3C: Web Services Glossary 2004. http://www.w3.org/TR/ws-gloss/. Accessed on 2008-06-16.
46. [W3C 2007] W3C: Semantic Annotations for WSDL and XML Schema – Terminology. http://www.w3.org/2002/ws /sawsdl/spec/#Terminology. Accessed on 2008-05-30.
47. [WSDL 2007] Web Services Description Language (WSDL) Version 2.0. Recommendation, W3C, June 2007. Available at http://www.w3.org/TR/wsdl20/.
48. [XMLSchema 2004] XML Schema Part 1: Structures. Recommendation, W3C, October 2004. Available at http://www.w3.org/TR/xmlschema-1/.