Enterprise Interoperability with SOA: a Survey of Service Composition Approaches

Rodrigo Mantovaneli Pessoa[1], Eduardo Silva[1], Marten van Sinderen[1], Dick A. C. Quartel[2], Luís Ferreira Pires[1]
[1] University of Twente, Enschede, The Netherlands
[2] Telematica Instituut, Enschede, The Netherlands
{mantovanelir, e.m.g.silva, m.j.vansinderen, l.ferreirapires}@ewi.utwente.nl
{Dick.Quartel}@telin.nl

Abstract

Service-Oriented Architecture (SOA) claims to facilitate the construction of flexible and loosely coupled business applications, and therefore is seen as an enabling factor for enterprise interoperability. The concept of service, which is central to SOA, is very convenient to address the matching of needs and capabilities in enterprise collaborations. In order to satisfy more demanding needs or to rapidly adapt to changing needs it is possible to perform service composition in order to combine the capabilities provided through several available services. This paper presents a survey on recent approaches for service composition. To perform this study a conceptual framework for service composition is proposed. This framework allows studying how different approaches deal with the service composition life-cycle and provides basic guidelines for their analysis, evaluation and comparison. The proposed framework is used to analyse five representative service composition approaches.

1. Introduction

One promising benefit of Service-Oriented Architecture (SOA) is to facilitate the construction of flexible and loosely coupled business applications. SOA-based businesses applications can span several networked enterprises, with services that encapsulate and externalize various corporate applications and data collections. Service composition is an essential ingredient of SOA, as it is concerned with aggregating interoperable services such that the goals of (enterprises in) a collaboration endeavour can be satisfied. Many individual service composition approaches and solutions have been proposed and developed in recent years. However, more effort has to be spent on their evaluation and comparison. In this paper we investigate recent approaches and technologies to support service composition.

Traditionally, research surveys on service composition tend to either focus on one particular emerging technology (such as workflow-based, AI planning-based, ontology-based, etc) [1, 2, 3, 4] or be domain-specific [5, 6]. In this paper, we adopt a different approach to this classification and organize our study around the concept of *service composition* life-cycle. We define the different phases of the service composition life-cycle, and based on this we create a comparison framework. The proposed framework establishes a set of evaluation criteria that provides basic guidelines for analysis, evaluation and comparison. We argue that our framework enables a more comprehensive understanding of existing service composition approaches, and allows us to recognize opportunities for combining approaches and identifying open issues and research challenges.

The remaining of this paper is structured as follows: Section 2 discusses enterprise interoperability issues related to service composition. Section 3 describes the service composition life-cycle. Section 4 introduces our comparison framework. Section 5 describes five different service composition approaches. Section 6 compares the studied approaches. Finally, Section 7 presents our conclusions and defines some future research directions.

2. Enterprise Interoperability and Service Composition

Enterprise interoperability denotes the ability of organizational entities (businesses, government, companies and parts thereof), hereafter called *enterprises*, to interoperate in order to achieve certain business goals [7]. Since enterprises are subject to constant internal changes and must continuously react to ongoing or imminent changes in markets and trading partners, interoperability solutions cannot be static. In addition, since enterprises increasingly use ICT to support their business activities, interoperability solutions cannot be restricted to the organizational level alone. Therefore, in order to develop practical solutions for modern enterprises, interoperability should be addressed both from organizational and technical points of view, and flexibility (next to other "ilities") should be a major concern.

The following aspects of interoperability have been distinguished [8]: (i) businesses; (ii) processes; (iii) services; and (iv) data interoperability. Businesses and processes interoperability are considered mainly at the organizational level, whereas services and data interoperability require focus on (information) technology issues. The term service can be used to denote a business function as well as a function of a computer-based application. In the context of this paper, we limit ourselves to the latter denotation, however, being aware that a comprehensive treatment of enterprise interoperability would require consideration of both denotations within a single framework.

It has now been widely recognized that SOA can bring significant benefits for enterprise interoperability [9]. A perceived value of SOA is that the concept of service, which is central to SOA, allows one to address the matching of needs and capabilities in enterprise collaborations at the proper level of abstraction. SOA is based on the assumption that enterprise systems may be under the control of different ownership domains. Therefore, the focus is on services that these systems can provide (capabilities) or that these systems want to use (needs), and ownership issues are not visible except for restrictions imposed on the use of these services. Services are self-contained units of functionality, which are described, published and discovered [10]. These properties form the basis for fulfilling the desirable flexibility mentioned above: service providers can make themselves and their services instantly known. Moreover, user needs can also be matched to a combination of multiple provider capabilities, corresponding to multiple services, through a mechanism where discovered services are composed with the aim of fully satisfying the user needs.

This brings us to the focus of this paper, namely *service composition*. We perform a survey of different service composition approaches, analysing them according to the phases of the service composition life-cycle.

3. Composite Service Life-cycle

A composite service consists of a composition of existing services to achieve some functionality that typically is not provided by a single available service. The composite service life-cycle provides an integrated view of the phases and artefacts produced for service development and execution. The phases cover design-time and runtime aspects of the composite service life-cycle and allow services to be created, operated and maintained. The life-cycle phases may vary according to the granularity chosen for the description of the different activities involved in the service composition process. Figure 1 shows the life-cycle we propose for the development and execution of composite services.

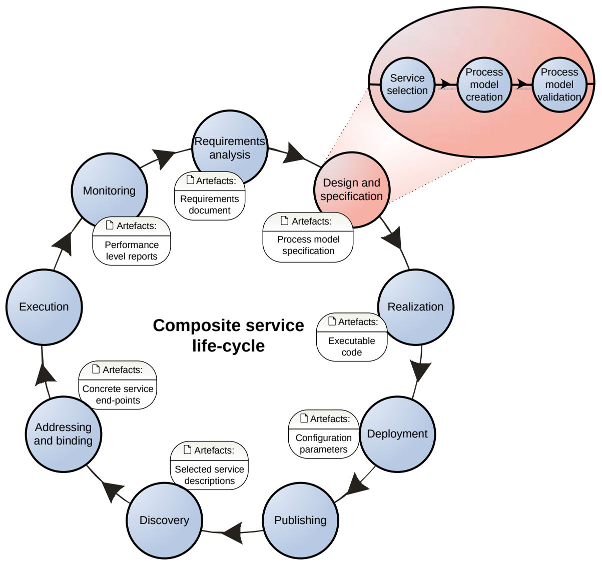

Figure 1. Composite service life-cycle

We have aimed at specifying the common phases and artefacts of a service composition process in a single and comprehensive model. The circles in Figure 1 represent phases of the composite service life-cycle, and the round rectangles represent the artefacts produced in each phase.

Requirements analysis. The first phase of the life-cycle identifies and prioritizes business and customers requirements. In this phase the scope is set, resources are planned, and the business context in which the new services will operate is determined. Possible needs for new services are identified based on an analysis of business and customers requirements. The artefact produced in the requirements analysis phase is a software requirements document, which typically details the system's functional and non-functional requirements in a structured form.

Design and specification. In this phase, the composite service is designed to fulfil the business and customers requirements identified in the previous phase. In this phase the services needed to realize the requested service composition are selected and a process model specifying how the services are coordinated is built and validated. This involves the following sub-phases:

- ***Service selection*:** performs the selection and ranking of suitable services to fulfil the composition requirements. The output of the selection and ranking algorithm is the list of services that fulfil all functional and quality requirements of the user, ordered by some criteria.
- ***Process model creation*:** relates to the creation of a service composition. We should be able to determine what component services are executed and by whom, at what moment in time, what are the components dependencies and what are the expected results. The process model can be specified in terms of a single party view or a multi-party view. The first is usually defined as a service orchestration, while the second is defined as a service choreography. The resulting artefact is a specification of how to coordinate the discovered and selected services to meet the user needs. This specification can be produced at design-time or runtime, using manual, semi-automated or automated composition methods.
- ***Process model validation*:** services may have the same or similar functionality. Therefore, it is possible that more than one composite service is generated to fulfil the service requirements. In this case, the composite services should be evaluated and ranked on their overall utility. This evaluation is usually performed based on the composite service functional and non-functional properties. The most commonly used evaluation method is by using utility functions that rank the created composite services according to weights specified by the service requester for each non-functional property. The validation of the composite service correctness is another aspect of the service composition. It consists of verifying if the requested requirements are achieved by the composite service, if there are no deadlocks, etc.

Realization. This phase focuses on the service technical implementation details. In this phase the service identified and designed in the previous phase is built and tested. The service implementation can be coded in different computer languages and the source code can be obtained through actual programming, wrapping existing legacy applications or by model-to-code transformations. The materialization of a service is completed when some executable (code) specification is produced as artefact.

Deployment. The service deployment phase addresses the problem of installing, configuring and managing services and service instances in the service execution environment.

Publishing. In this phase the service description information is published. Usually this information is specified in a service description document. The published information allows one to advertise the service, and so that the service can be discovered by potential consumers. The service description document describes what the service does, where it can be found, and how it can be invoked.

Discovery. The discovery phase concerns the service consumers' ability to find (either at design-time or at runtime) the service descriptions published in service registries in the Publishing phase. Once the services are published in the registry, users can search and find the services that meet their requirements.

Service binding. A service may have different implementations, and each implementation may have multiple deployments in different service end-points. The service binding phase focuses on how service endpoints are discovered and instantiated. The binding phase may be performed either at design-time or at runtime, and can be static or dynamic. Static service bindings are defined at design-time and define a tightly coupled interaction between service user and service provider. Dynamic service bindings allow a dynamic binding of service user and service provider's service at runtime, given a selection and discovery mechanism, usually defined at design-time.

Execution. The execution phase involves the invocation of all participating services, possibly hosted in different provider domains. The execution of a composite service must be consistent with the specified process model and for this, a coordination mechanism is required. During composite service execution, this coordination mechanism is responsible for invoking the participating services, receiving notifications of completion from each participating service, transferring and transforming (when required) the input/output parameters among the participating services, and evaluating pre-conditions that must be satisfied prior to the invocation of a participating service and post-processing actions that must be performed after the execution of a participating service.

Monitoring. In this phase, the service provider constantly monitors the composite service execution, evaluating its performance and verifying if the agreed performance levels are met. The goal of the service monitoring phase is to rapidly respond to indications of service degradation or failure, and to ensure that the service level agreements are fulfilled. To achieve these objectives, in general, service monitoring requires a set of QoS metrics to be gathered and interpreted.

4. Comparison Framework

In this section, we present our framework for analysis and comparison of service composition approaches. The proposed framework is derived from our service life-cycle. We specially focus on the service discovery and composition phases, or *Process model creation*, but also include other phases, such as process model verification and service execution. For each considered life-cycle phase we developed a set of evaluation criteria. Based on the framework, one may evaluate different service composition approaches, having a common ground to compare them. The characterization of service composition approaches is generally based on literature review. Table 1 presents our framework.

Table 1. Comparison framework

Life-cycle Phase	Evaluation Criterion
Service Discovery	Service description
	Service matching and selection
Process model creation	Behaviour specification
	Information specification
	Level of automation
	Composition time
	Coordination distribution
Process model verification	Composition correctness
Execution	Service binding

In the following, we present in more detail each of the considered evaluation criterion of our framework:

Service description: aims at evaluating how a service is described. Different aspects can be considered, but we may reduce them to two groups: functional and non-functional aspects. The service functional description focus on the functionality supported by the service, which is usually expressed in terms of inputs, outputs, preconditions and effects (IOPEs). The non-functional service description aspects describe other properties the service such as: performance, availability, cost, etc.

Service matching and selection: refers to the task of discovering and selecting services that can be used in the composition. Services are discovered based on a set of requirements specified in a service request. The discovery and matching may be based on different information, such as service goals, or IOPEs. At the end of the service discovery process a list of services are retrieved. This set can be further filtered based on selection criteria. Different approaches may be taken to perform service selection. For example, non-functional properties, such as service cost, may be used to rank candidate services.

Behaviour specification: deals with the design of composite service behaviour. Here we focus on describing the languages and formalisms used by approaches for modelling the behaviour of a service composition (combined behaviour of the component services) and for defining constraints between the service operations that determine allowed invocation orders.

Information specification: since services handle data in the form of input and output parameters, it is necessary to model the data (types) that a service can handle, the data flow of the composite service and possible data transformation between the component services of a composite service.

Level of automation: given a set of available services and a user service request description, the problem of service composition synthesis is concerned with the creation of a new composite service, thus producing a specification of how to coordinate the available services to realize the client request. Such a specification can be obtained either automatically, i.e., using a tool that implements a composition algorithm, semi-automatically, i.e., in case the user makes choices during the composition phase aided by an interactive tool, or manually by the user. Depending on the intended purpose of a given approach, different (and specific) requirements may arise and be imposed.

Composition time: refers to the moment at which the approaches perform the service composition synthesis. Two distinct moments may exist, namely design-time and runtime. However, an initial composition plan can be defined at design-time, which can be adapted dynamically at runtime.

Coordination distribution: the coordination of a composite service requires that the service is completely specified, in terms of both the specification of how various services are linked, and the internal process model of the composite Two main kinds of coordination have been identified in [11]:

- ***Centralised:*** centralised coordination is based on a hub-and-spoke topology, in which one service is given the role of process mediator/delegator, and all the interactions pass through such a service. This mechanism is usually defined as orchestration.
- ***Peer-to-peer:*** in decentralized coordination, there are multiple coordination entities, placed at distributed locations, each executing a composite service specification (which is a portion of the original composite service specification). The coordination entities communicate directly with each other rather than through a central coordinator, in order to transfer data and control when necessary in an asynchronous way. This mechanism is usually defined as choreography.

Composition correctness: refers to the capability of checking the correctness and reliability of the composite service with respect to the service requirements. Composition correctness requires verification of the composed service's properties, such as reachability, liveness or safety.

Service binding: in order to enhance the flexibility of a composition, services are usually not hard-coded into the composition model but bound into it at different times (i.e., runtime and design time). During execution, a composition engine has to target messages to specific services, which are defined in the composition schema. The service selection model deals with static and dynamic binding, i.e., how a service is selected and bound statically at design-time or dynamically at runtime. Alonso et al. [12] describe four different service binding models:

- Static binding: service endpoint URL is hard-coded;
- Dynamic binding by reference: service URL is computed and stored into a variable;
- Dynamic binding by lookup: before each service invocation a query is sent to a registry to locate a suitable implementation;
- Dynamic operation selection: no assumptions are made about the signature of the arbitrary service to be invoked.

5. Service Composition Approaches

Recently, several techniques and methodologies for modelling and specifying different aspects of service composition have been proposed. In this section, we discuss some representative approaches with respect to the evaluation criteria defined in our framework, in order to compare them.

5.1. METEOR-S

The METEOR (Managing End-To-End OpeRations) project at the Large Scale Distributed Information Systems (LSDIS) Lab at the University of Georgia focused on workflow management techniques for large-scale transactional workflows. Its follow-up project, which incorporates workflow management for semantic web services, is called METEOR-S (METEOR for Semantic web services) [13]. A key feature in this project is the usage of semantics for the complete life-cycle of semantic web services. Its annotation framework is an approach to add semantics to current industry standards such as WSDL. Finding an appropriate service for the composition is realized by a discovery engine that queries an enhanced UDDI registry.

Service description. The service descriptions are semantically augmented, which resulted on the SAWSDL (Semantic Annotations for WSDL) language. SAWSDL is based on WSDL-S, which was a joint specification developed by IBM and LSDIS Lab for adding semantic annotation to WSDL. SAWSDL is a simple extension of WSDL using the extensibility elements. It provides a mechanism to annotate the capabilities and requirements of web services (described using WSDL) with semantic concepts defined in an external domain model (e.g., ontology). Externalizing the domain models allows SAWSDL to take an agnostic view towards semantic representation languages. This allows developers to build domain models in their preferred language or reuse existing domain models. It has two basic types of annotations: the model reference and the schema mapping. Additionally, the METEOR-S framework extends the SAWSDL annotations with preconditions and effects, used to describe the conditions that must be met before an operation can be invoked and the result that the invocation of the operation will have.

The approach uses QoS ontologies to represent the semantics of service non-functional properties and has described generic QoS metrics based on four dimensions: time, cost, reliability, and fidelity. Each metric specification consists of a quadruple. *QoSq(s,o) = <name, comparisonOp, val, unit>*, where *‘name’* is the parameter name, *‘comparisonOp’* is a comparison operator, *‘val’* is a numerical value, and *‘unit’* is the metric unit. The approach also presents a mathematical model that formally describes the formulae to compute QoS metrics among workflow tasks and an algorithm to automatically compute the overall QoS of a workflow.

Service matching and selection. METEOR-S has developed a three-phase algorithm for service selection that requires the users to enter service requirements as templates constructed using ontological concepts. In the first phase, the algorithm matches services (operations in different WSDL files) based on the functionality they provide. In the second phase, the result set from the first phase is ranked based on semantic similarity between the input and output concepts of the selected operations and the input and output concepts of the template, respectively. The optional third phase involves ranking based on the semantic similarity between the precondition and effect concepts of the selected operations and preconditions and effect concepts of the template. The semantic matching on the semantic template of the activity is done against the operations, inputs, outputs, preconditions and effects of the services available. The ranking on semantic matching is based on the weights assigned by the process creator to the individual semantic parts of the activity, namely operations, inputs, outputs, preconditions and effects. The assigned weights are normalized before calculation

Information specification. The information specification is based on manual specification of the data semantics. The model reference annotation is used to specify the association between a WSDL element and a concept in some semantic model (ontology). The schema mapping annotations are used by the METEOR-S framework to deal with further mismatches in the structure of the inputs and outputs of the web services, particularly transforming one data representation into another, such that it can be used in another web service. Mappings are created between the web service message element and the ontology concept with which the message element is semantically associated. In addition to a mapping from the web service message element to the ontology concept, also called the *liftingSchemaMapping*, an additional mapping from the ontology concept to the message element, called the *loweringSchemaMapping*, is specified. Once the mappings are defined, two web services can interoperate by reusing these mappings and the ontologies now become a vehicle through which web services resolve their message level heterogeneity.

Behaviour specification. METEOR-S specifies the process model describing the behaviour of services by capturing semantics of the activities in the process template during the design phase. The activities are not bound to web service implementations, but defined using semantic descriptions. Such templates are independent of the service description and process definition standards. The process template is a collection of activities, which can be linked using control flow constructs. The process templates in METEOR-S have a BPEL-like syntax. For representing control flow, the template uses the BPEL constructs. The template has also some additional constructs, like invoke activity, criteria, semantic-spec, discovery-spec, etc. which are prescribed in the BPEL specification, i.e., they are METEOR-S specific constructs independent of any process specification standard that can be used to generate executable processes.

Level of automation. The development module provides a GUI-based tool for creating semantic web services using SAWSDL. The tool provides support for semi-automatic and manual annotation of existing web services or source code with concepts from domain ontologies.

Composition time. METEOR-S offers support for two types of service composition, namely Static Composition (services to be composed are decided at design-time, static binding) and Dynamic Composition (services to be composed are decided at runtime, dynamic binding).

Coordination distribution. The coordination of the composite service is based on a BPEL-like centralised process engine, or an orchestration.

Composition correctness. The constraint analysis and optimization sub-modules deal with correctness and optimization of the process based on quality of service constraints. There is also support for state machine based verification of BPEL process.

Service binding. The current prototype supports three kinds of service binding: static binding, deployment-time binding and dynamic binding. In static binding, a set of services is permanently bound to the composition. Deployment-time and runtime binding are achieved by using a proxy-based approach to bind a set of services that realise the service composition. In deployment-time binding, configuration is performed before the process starts executing. In runtime binding, configuration is performed after the process starts executing. Both deployment-time and runtime binding support reconfiguration. The configuration module has the ability to change the service bound to the proxies by simply changing a field in a shared data structure. This data structure is synchronised and accessed by each proxy before each service invocation. During reconfiguration, the process manager locks the data structure, thus making all proxies wait while the process is being reconfigured..

5.2. SODIUM

SODIUM (Service-Oriented Development In a Unified fraMework) was an international project, involving research, technological and industrial partners, dedicated to tackling interoperability challenges that companies face at the data, services and business levels [14]. The project has developed a Generic Service Model, containing the common concepts of heterogeneous services from multiple points of view. The special characteristics of individual service technologies (such as Web services, Grid services or P2P services) are then dealt with as extensions to the core.

The SODIUM methodology adopts a model-driven and iterative approach for service composition and evolves in four phases: the user starts by defining the details of the complex task at a high abstraction level (phase 1); the user uses this abstract description to generate queries used for service discovery (phase 2); the discovered services are used to populate each of the abstract process tasks, hence transforming it to a concrete process description, and both the abstract and concrete descriptions are stored in the service composition (phase 3); and the concrete process description is transformed into executable descriptions and publishable documents about the composite service which has been built (phase 4).

Service description. The service is represented in a UML model, according to an UML profile that can be used to model semantic aspects of web services. Activities are stereotyped in order to represent web service operations. Parameters of this activity element represent its inputs and outputs. A web service activity has a set of tagged values. The web service provider is defined by the tagged value *provider*. The URL to the WSDL file is registered in the tagged value *wsdl*. The exact service operation to invoke is given by the three tagged values *service*, *portType* and *operation*. To represent a p2p service operation, activities are stereotyped as *P2PServiceOperation*. The five tagged values type defined for a peer-to-peer service operation are *PSDL*, *Operation*, *Service* and *Pipe*. To represent a grid service operation, activities are stereotyped as a *GridServiceOperation*. The six tagged values type defined for a grid service operation are *gwsdl*, *ServiceLocation*, *Service*, *Operation*, *PortType* and *ResourceInstance*. The service may optionally have semantic references for the data types used and QoS offered, which are described depending on what kind of information we can retrieve about the service.

The approach makes use of OMG's QoS profile to represent collections of QoS properties with precise semantic meaning in the UML service model. Each QoS property contains a set of QoS dimensions with a name, its allowed value domain, an ordering function (whether higher or lower values are considered better) and its relationship to other QoS properties. The ordering direction of a property is defined as either increasing or decreasing, where increasing means that higher values are preferred. All the QoS properties to be used elsewhere shall be defined as a QoS property either within the model itself or as in imported model. Since SODIUM adopts the OMG profile, it already has generic capabilities to define QoS ontologies. In this sense, SODIUM does not suggest any specific QoS dimensions as part of the language, but is capable of defining any needed QoS dimension.

Service matching and selection. The Behavioural Service Discovery Framework (BSDF) presented a novel approach in service discovery, which enables modelling queries with behavioural constraints in a visual manner. It comprises three main components: (i) a visual query modeller that models behavioural service queries by means of UML behavioural diagrams; (ii) a translator that transforms the raw XMI output of the modeller to a generic XML-based query language, namely USQL; and (iii) a query engine capable of processing and executing USQL queries in various types of target registries, repositories, networks, etc.

The framework is able to match the query against various types of service choreography advertisements, independently of their format and protocols. The basic idea of the USQL engine lies in the logical grouping of heterogeneous registries, depending on the domain their advertised services belong to. Having the registries organized in this way, the engine sets the service requestor one step closer to his/her specific requirements and narrows the range of the returning results, making them more relevant to the initial request.

Information specification. Data objects are used to represent the data content that is created in the composition and that may be passed along to different activities. A data object has a specific *ObjectType* (optional) and may also represent the input and output parameters of the whole composition.

If the outputs of one service do not perfectly match the required input of the next service, there is a need to introduce intermediate data transformation steps between the services. This requires manual adjustments by the developer when specifying transformation nodes for defining data transformations as expressions in QVT (Queries/Views/Transformations). The transformation node is used for one-to-many, many-to-one and many-to-many data transformations. In a many-to-one transformation, the information from many source data objects is used to produce the content of a single target data object.

Behaviour specification. The Visual Service Composition Language (VSCL) is a graphical composition language for defining service compositions containing heterogeneous (web, grid, p2p) services. The main concepts of the languages are the tasks and the flow of data and control between tasks. The task-graph node consists of a task part and an entire sub-graph. Thus, the language has a construct that can be repeated at arbitrary levels to create a recursive decomposition structure. A task may be executed by different kinds of services, namely P2P, Web or Grid services. A service composition consists of Nodes and Flows/Edges. The Nodes are *TaskNodes* , which represent the invocation of a remote service, *ControlNodes*, which represent specific crossings for control flow, *ObjectNodes*, which are used for data transfer between the tasks, *EventNodes*, which represent an expression node that passes control through its outgoing arcs upon the occurrence of a predetermined event or *TransformationNodes*, which are used for defining data transformations. Two different kinds of Flow are used to specify flow of control and data between nodes. A Flow indicates a directed flow of either flow of control (*ControlFlow* or *EventFlow*) or flow of data objects (*ObjectFlow*). A Flow has one source node and one target node.

Level of automation. The aim of the SODIUM approach is semi-automated tool support for service composition. In order to do so, the approach has automated large parts of the steps needed in the process of developing composite services. Many of the proposed steps for automation are model-driven transformations that transform between models and lexical descriptions about the services, both forward and reverse engineering.

Composition time. Though runtime service selection is discussed, the primary focus has been design-time service discovery and composition.

Coordination distribution. The coordination distribution runs on a central execution engine. This component deals with the execution of compositions by invoking services and orchestrating the control and data flow across the different steps of the composition.

Composition correctness. The analyser is a component used for validating a composition against a set of rules. This allows one to check the model for syntactical errors. A dialog box allows the user to select which rules to validate, to execute and present the analysis result. Rules can be defined for each of the main classes in the model. Some constraints should be checked during editing, while others should be checked before the generation of lexical executable representation of the composition. However, no formal proof of correctness is given.

Service binding. SODIUM refines the concept of binding beyond the basic distinction of static and dynamic binding. Service binding can be defined at the design, the compilation, the deployment, the beginning of the execution of a composition, or just before the actual service invocation takes place.

5.3. MoSCoE

MoSCoE (Modeling Web Service Composition and Execution) [15] is a project coordinated by Iowa State University. MoSCoE aims at the creation of a framework for modelling service composition and execution. The composition process in MoSCoE is divided in three-steps: abstraction, composition and refinement. Abstraction is provided to the framework users, allowing them to request a service using a high-level specification. The service providers advertise their services using common standard service description languages, namely OWL-S and WSDL descriptions. Given a service request, the framework's composition engine creates a suitable composition, from the existing services, if possible; otherwise it starts the refinement phase, guiding the user through a service request refinement procedure to create a service composition. The refinement process is iterative, stopping when a suitable service composition is found, or when the user decides to end the composition process. If a service composition is obtained, it is translated into a concrete BPEL workflow, which can be executed. The service composition defines rules for non-functional properties. At the execution time these rules are monitored, and in case of some specified event takes place, the appropriate actions are taken. Furthermore, while executing the service composition, various data and control flow transformations are carried out by referring to the pre-defined ontologies used in the service descriptions, and to specified inter-ontology mappings.

Service description. Existing services are represented in OWL-S and WSDL specifications. OWL-S is used to semantically describe existing services, and specifying functional and non-functional aspects of the services, mainly for discovery. The MoSCoE project claims that other languages for service description can be supported. In fact, the framework translates all the service request and service descriptions to State Machine representations. This means that the service composition process is independent of the service description languages.

Service matching and selection. MoSCoE performs service discovery based on semantic descriptions. It assumes that existing services are semantically described in OWL-S. Services are discovered based on the specified user request functional aspects, the so called IOPEs (Inputs, Outputs, Preconditions and Effects). The service request is specified in a visual form, using a UML State Machines representation of the desired service. This information is interpreted to perform service discovery. The discovered services are organised in terms of degree of *semantic match*, which can be *Exact*, *Plug-in*, *Subsumption*, *Intersection* or *Disjoint*. The degree of match is computed through semantic reasoning on the requested properties and the existing services' semantic descriptions. Services that have intersection and disjoint matches are not considered as valid matches.

From the set of valid matches, a selection based on non-functional properties is performed. Non-functional properties are also described in an ontology. MoSCoE defines a quality vector which allows the user to specify which non-functional properties are of interest. Based on the quality vector, a quality matrix can be constructed, where lines represent the quality vector and columns the candidate services values for the considered non-functional properties. Furthermore, it is possible to define weight values for the different considered non-functional properties, which allows computing an additive value function, and consequently selecting the best suited service.

Information specification. MoSCoE allows one to semantically describe services, in terms of both functional and non-functional properties. However, the necessary supporting ontologies may be defined by different parties. This may cause problems of semantic interoperability of the services used in a composition.

Behaviour specification. In MoSCoE, service requests are specified using a UML State Machine representation. The discovered services are also translated to a State Machine representation. However, the MoSCoE service composition is described using a Transition System representation, more specifically, a Symbolic Transition System (STS). Therefore, transformations have to be performed on the service request and candidate services, to obtain Symbolic Transition Systems from state machine representations.

Once the candidate services are translated to STS, they are combined to reach the service goal specified by the user. This composition is obtained automatically, consisting on sequential and/or parallel compositions of STS service representations to meet the service goal specified by the user. If a composition is possible, the framework proceeds with the translation of the resulting STS representation to executable code, namely BPEL executable code. In case no composition is possible, the *Refinement* phase is triggered.

The Refinement phase consists of performing a new iteration with the user, asking for a service request refinement (using the UML State Machine representation). At the moment a refinement request is issued, concrete information about the problems/issues encountered during the service composition creation is provided. Given this, the user is guided on the refinement process, being asked to give more detailed information on concrete problems found on the service composition. After delivering the more detailed information, the UML State Machine is interpreted again, a new service discovery is performed, and a new set of services is retrieved. The service request and the set of discovered services are translated to State Machines and to STSs. Based on the STSs, the framework performs a new attempt to create a service composition that matches the *refined* user service request. If this is possible, the framework stops the service composition process, generating the executable code; otherwise it asks the user for a new refinement. This cycle may happen indefinitely if no service composition is constructed, unless the user decides to stop the process.

Level of automation. The MoSCoE approach to service composition can be classified as semi-automatic. The process of composing the existing services is automatic, but the user is asked for refinements in case a service composition matching the user initial service request cannot be found. Furthermore, the user is expected to specify a UML State Machine representing the service request. This process may be extremely complex for non-technical users.

Composition time. MoSCoE is mainly targeted to design-time service composition. The runtime service composition is not emphasized in the framework documentation.

Coordination topology. MoSCoE creates a service composition strategy, or orchestration, which defines the behaviour of a mediator, consisting of a plan that allows the management of interactions with the different service composition components.

Composition correctness. MoSCoE uses Symbolic Transitions System (STS) to represent services and service compositions. By using this formal representation, the framework has mechanisms to formally verify the created service compositions. MoSCoE checks soundness and completeness of the composition against the provided set of restrictions on the service composition. The service composition is also checked at the moment a new refinement is issued. This checking is used to provide specific information concerning the problems found at the composition time to meet the specified service user goals.

Addressing and binding. MoSCoE defines static service bindings at design-time. However, constraints and rules are also defined for non-functional properties of the service composition. These properties are monitored at runtime. If some specified event takes place, the engine stops the service composition execution and an alternative service composition is selected, if available.

5.4. SeCSE

SeCSE (Service-Centric Systems Engineering) [16] was a European project from the 6th Framework Programme for Research and Development. The SeCSE project focused on the creation of new methods, tools and techniques for requirements analysis, system integrators and service providers to support cost-effective development and use of dependable services and service-centric applications.

The four main research areas of the project were Service engineering; Service discovery; Service-centric systems engineering; and Service delivery. To address these areas, SeCSE has defined a methodology consisting of the following phases: i) Business requirements definition and service discovery; ii) Composition creation; iii) Instrumentation and monitoring rule definition; iv) Service regression testing; v) Deployment of service composition; vi) Service-centric system description; vii) Service-centric system publication.

Service description. Faceted Service Specification [17] is used to perform service description. The proposed Facet Specification structure includes a stable element and variable elements. The principal stable element is that the XML file structure used to represent a Service Specification and associated Facets will not change, since these are independent of Facet types. Instances of any new Facet type can be listed in a Service Specification provided a type name has been defined and made public. Any new Facet Specification language can be used simply by embedding specifications written in the language within a Facet Specification file. The variable elements that the SeCSE runtime architecture needs to accommodate are: i) new Facet types, as service consumers' requirements evolve; ii) service specification languages, since new service specification languages are emerging and existing ones are still evolving. However, service consumers wishing to evaluate a service specification need to be able to establish whether their tool (if any) is capable of interpreting the specification. Hence, an indicator of the language used is needed, but the mechanism used to interpret it needs to accommodate potentially arbitrary choices of language and their versions.

Service matching and selection. The service discovery phase is performed based on a user (or Service Integrator as used in SeCSE) service request. A service request is defined using UML Use Case specifications. Additionally the service request specifies also functional requirements, using VOLERE. Use cases and requirements are expressed in structured natural language, using a SeCSE tool called UCaRE.

Once the use cases and requirements are specified, they can be used to construct a service request in UCaRE, allowing the user to select the information to be used in the service discovery query. The service request query is then passed to EDDiE, the SeSCE service discovery engine. After some manipulation on the natural language service request query, a two steps matching is performed: i) XQuery text-searching functions to discover an initial set of services descriptions that satisfy global search constraints; ii) traditional vector-space model information retrieval, enhanced with WordNet to further refine and assess the quality of candidate services set. The resulting matches are then presented to the user.

Information specification. SeCSE uses Faceted Service Specification to deal with information specification, and service description.

Behaviour specification. The process model specification is based on the activities identified in the requirements analysis phase. The user identifies the different required activities, and based on the set of discovered services he realises how the service composition can be made. This is done through a workflow definition. Given this workflow, the user proceeds with the actual service composition, using the Composition Designer (CD) tool provided by the SeSCE project. CD allows manipulating the WSDL descriptions of the discovered services and creating the necessary BPEL service description. From the resulting data, CD can generate a UML model that is recognized by the SeSCE Architecture-time Service Discovery (ASD) tool, which allows the publication and discovery of the composed service.

The CD tool also allows the user to further enrich the service composition description with binding and monitoring rules. These rules enable runtime dynamic adaptation of the service composition in case specific events take place. The definitions of the rules consist of Event-Condition-Action (ECA) expressions.

Level of automation. The service composition process can be considered as semi-automatic. The definition of the service composition is obtained based on a workflow specified by the user of the system. However, the user is supported in this task by the high level specification of the requirements provided in the requirements analysis phase. Furthermore, the actual service composition, or instantiation of the defined workflow, is performed based on the list of services discovered automatically from the user's service request. The user also specifies binding rules, which allow a dynamic adaptation of the service composition according the defined constraints.

Composition time. The composition process proposed by SeSCE can be considered as hybrid. The service composition is specified at design-time, as the dynamic binding rules. However, dynamic adaptations of the service composition take place at runtime, in case some pre-defined event takes place, such as the unavailability of a given service.

Coordination topology. The coordination topology consists on a centralised topology or orchestration, where a single party manages the service composition execution, monitoring and reconfiguration.

Composition correctness. SeCSE focuses mainly on QoS requirements verification, to perform runtime adaptation of the service composition. A technique based on a genetic algorithm is used for this purpose.

Addressing and binding. Binding is defined at design-time. However, the SeCSE approach also defines binding rules at design-time, which allow one to specify ECA rules to be interpreted at runtime and guide possible dynamic binding changes at runtime, as some events are observed, leading to reconfiguration actions on the service composition.

5.5. WSMF

The Web Service Modelling Framework (WSMF) [18] is a framework for describing the various aspects related to web services composition. The framework provides a Web Services Execution Environment (WSMX), which is a reference implementation of the Web Services Modelling Ontology (WSMO) and operates using the Web Services Modelling Language (WSML).

The WSMF conceptual model incorporates four core elements that are essential to represent semantic web services and related issues, namely ontologies, which provide the common terminology used by other WSMO elements, services, which are requested, provided, and agreed upon by requesters and providers, goals, which provide means to characterize user requests in terms of functional and non-functional requirements, and mediators, which deal with interoperability problems between different WSMO elements. In addition to these core elements, WSMO introduces a set of core non-functional properties that are globally defined and may be used by all its modelling elements.

Service description. In WSMO, requestors of a service express their objectives as goals, which are high level descriptions of concrete tasks. A WSMO goal description consists of a requested capability and requested interfaces. The former specifies the objective to be achieved in terms of a capability from the user perspective, while the latter specifies the communication behaviour for automated web service usage supported and required by the client. Analogously, a WSMO service description consists of a capability, which is a functional description of a web service describing constraints on the input and output of a service through the notions of preconditions, assumptions, post-conditions, and effects, and choreography interfaces that specify how the service behaves in order to achieve its functionality, i.e., the interaction behaviour supported by a service.

Service matching and selection. In the context of WSMF, service discovery is based on matching abstract request goal descriptions with semantic annotations of services. Hence, in order to precisely express user goals with respect to discovery, WSMO goals carry an additional non-functional property *typeOfMatch*, which denotes the matchmaking notion to be applied. The simplest approach uses an algorithm that matches keywords from the goal description with keywords from the service descriptions. For the lightweight semantic discovery, the capabilities of goals and candidate services are transformed into a variant of WSML and expressed as *taxon* concepts. Once this transformation has taken place, an appropriate reasoning engine is used to determine if there is an overlap in the set of concepts resulting from the transformation of the goal and service capabilities, respectively. The QoS-based discovery mechanism can be used to filter services from a large set of candidates or to order services that match a goal according to some specific QoS characteristic.

Information specification. In WSMF, ontologies are used to define a common agreed terminology, by providing concepts and relationships between these concepts. Only WSML is used as WSMX internal data representation. By reusing standard terminologies, different elements can be either linked directly or indirectly via predefined mapping and alignments, in order to achieve interoperability between services. The concept of mediation in WSMO has been introduced to handle heterogeneities that may exist between elements that should interoperate, by resolving data and behavioural mismatches. A WSMO mediator connects the WSMO elements in a loosely coupled manner, and provides mediation facilities for resolving mismatches that might arise in the process of connecting different elements defined by WSMO. More specifically, WSMO defines four types of mediators: OOMediators, which mediate heterogeneous ontologies, GGMediators, which connect Goals, WGMediators, which link web services to Goals, and WWMediators, which connect interoperating web services, resolving their mismatches.

Behaviour specification. The behaviour specification is based on Abstract State Machines (ASM), consisting of states and guarded transitions. A state is described by the WSMO ontology and the guarded transitions are used to express changes of states by means of transition rules. The domain ontology constitutes the underlying knowledge representation for the ASM, and transition rules specified in terms of logic formulas describe how state changes when a transition is executed. WSML defines a syntax and semantics for ontology descriptions and comprises different formalisms, most notably Description Logics and Logic Programming. The underlying formalisms are used to give a formal meaning to ontology descriptions in WSML, resulting in variants of the language that differ in logical expressiveness and in the underlying language paradigms.

Level of automation. The framework aims at an automated, goal-driven service composition that builds on pre- and post-conditions associated to service descriptions.

Composition time. During design-time, the design and implementation of adapters, creation of ontologies and service descriptions, rules for lifting/lowering, and mapping rules between ontologies are carried out. The runtime phase involves discovery, selection and execution of the appropriate services to accomplish a given goal.

Coordination distribution. Information interchange for consumption and cooperation of services happens in a peer-to-peer manner, without the need of a central coordination entity.

Composition correctness. Apparently there is no explicit support for correctness in this framework.

Service binding. WSMO defines a proxy infrastructure for dynamic service binding and invocation. For each invoked service, a proxy has to be declared. The proxy allows referencing a service without knowing at design-time which concrete service is bound to it. This reference may consist of a goal definition or of a name, pre-conditions, post-conditions, input ports, output ports as well as error ports. The binding process happens at runtime and is based on binding rules defined in the proxy. The binding can be fixed to exactly one service, be defined in a registry (e.g., UDDI) or depend on input data coming from an input port. The last case means that the requester has full control over which service to select at runtime, because he can specify the binding criteria through the input ports. The only condition is that the data required to execute the composite service can be provided by the service bound at runtime.

6. Discussion

Table 2 summarizes the profiles of the composition approaches described in the previous section, according to multiple evaluation criteria defined within our comparison framework. By contrasting their profiles, it may be concluded that each approach focus on a specific set of phases involved in a composite service lifecycle, while disregarding others.

Table 2. Comparison of the approaches

	METEOR-S	SODIUM	MoSCoE	SeCSE	WSMF
Service description	Semantically augmented with SAWSDL	UML models enriched with. constraints and OMG's QoS profile	Services are represented in OWL-S and WSDL	Services are represented in Faceted service specification	Capability described in terms of pre- and post-conditions, assumptions, and effects
Service matching and selection	three-phase matching algorithm based on semantic similarity	Based on USQL queries with behavioural constraints	Semantic reasoning optimized with non-functional properties	Two-phase matching algorithm based on text-searching functions	Keyword matching, lightweight semantic matching and QoS based
Behaviour specification	Based on process templates with BPEL-like syntax	A graphical composition language is used to define data and control flow	Based on UML state machine diagram and Symbolic Transition Systems	BPEL-like service composition creation based on abstract workflow definition	Based on abstract state machines, consisting of states and guarded transitions
Information specification	Model reference annotations and schema mapping annotations	Data objects used as internal data representation format and data transformations expressed in QVT	Inter-ontology mappings	Faceted service specification	Ontologies are used as internal data representation format and mediators are defined in case of data mismatch
Level of automation	Support for manual and semi-automatic	Semi-automatic support	Semi-automatic support	Semi-automatic support	Automatic
Composition time	Design-time and runtime	Design-time	Design-time	Design-time with binding rules for dynamic adaptation	Runtime
coordination distribution	Centralized	Centralized	Centralized	Centralized	Peer-2-peer
Composition correctness	State machine based verification of BPEL	No support for formal proof of correctness	Symbolic transitions system based	No explicit support	No explicit support
Service binding	Static binding, deployment-time binding and dynamic binding	Design-time, compilation-time, deployment-time and runtime	Static binding	Static and dynamic binding	Static and dynamic binding

When comparing the described approaches, it is possible to notice that services can be described in different ways. They are described according to different existing standards (e.g. WSDL, SAWSDL, OWL-S, WSML), and can be characterized by a set of input and output parameters, QoS parameters, keywords, pre- and post- conditions, effects, etc. Nevertheless, it has been done, at varying levels of abstraction and each of these levels implies a different description of services, ranging from simple unstructured keywords to detailed characterizations of possible state transitions.

The service discovery phase is directly dependent on the way services are described. Consequently, the achievable accuracy of a result in the discovery phase may vary significantly from one approach to another, since different sorts and amounts of information are available during the discovery phase and since more or less structure and semantic are embedded in the service descriptions, which are used by the matching algorithms. There is, however, an inherent trade off between expressiveness of the service descriptions and computational performance.

The way in which behaviour is specified also varies across the compared approaches. The behaviour of a composite service can be described explicitly, using some language that directly specifies the composition flow control (allowed order of invocations). This principle is currently taken by SODIUM and SeCSE approaches and reflects its primary focus on design-time service composition. Alternatively, such behaviour can also be described indirectly, by specifying the conditions under which the involved services in a composition can be invoked, its inputs and outputs parameters and the effects of such invocations. METEOR-S and MoSCoE allow IOPEs (inputs, outputs, preconditions and effects) to be specified at the level of WSDL operations. It allows AI planning techniques to be used to fully or partially automate the service composition process. The planning algorithms can be executed at design-time or runtime to find a suitable ordering of the operations, based on the initial conditions and the goal of a particular client. However, planning algorithms usually have high computational complexity and require substantial resources. WSMO specifies the conditions and effects using abstract state machines, consisting of states and guarded transitions. A state is described within an ontology and the guarded transitions are used to express changes of states by means of transition rules. However, this implicit behaviour specification may be neither intuitive nor trivial to make sure that the expectations implied by the designed transition rules match the expected operation message exchange patterns in the context of a service composition.

Concerning information specification, there are several converging ideas focusing on schema mappings and data transformations to cope with heterogeneity issues that can exist between the formats of the data exchanged between services. Similar to various existing approaches, developers may specify mappings between each element of an input/output parameter of a service and concepts from different data schemes. Ontologies are used as the information model throughout both WSMO and MoSCoE. Because WSMO heavily emphasizes mediation, mediators are a first class component of the WSMO service model. An example of a WSMO mediator for resolving data mismatches is the *ooMediator*, which links two ontologies, resolving possible mismatch issues between them. In METEOR-S, the data mediator module uses the data semantics of the proxies and the services, to perform XSLT transformations data mediation between the on-the-wire XML data. With this approach, the grounding needs to link the inputs and outputs of the service with the appropriate XSLT transformations. Therefore, mapping and merging of schemas becomes a core question and some (*semi*)automatic support has to be developed to reduce the exhaustive work needed for manual creation and maintenance of these mappings.

Regarding the coordination distribution, more traditional approaches assume a centralized coordination, where a central entity coordinates the invocation of services involved in a composition. In the other hand, WSMF goes beyond this traditional form of central coordination, where a peer-to-peer interaction takes place among equal partners, in terms of their level of control over other entities.

7. Conclusions

In this paper, we have presented an analytical framework for analysis and comparison of service composition approaches. The framework was developed following the phases of the composite service life-cycle. Additionally a set of criteria was identified to evaluate each of the considered life-cycle phases.

We claim that service composition plays a major role in enterprise interoperability, and so here we present some state of the art on service composition approaches. According to our framework, an ideal approach would efficiently cover all of the requirements that we have identified across the key phases of the composite service life-cycle. However, for practical reasons, the compared approaches focus on specific phases of the life-cycle, while neglecting others. Not surprisingly, the described approaches widely differ in how they address the above mentioned requirements.

Based on our study we conclude that none of the service composition approaches we investigated covers all the life-cycle phases and is commonly accepted by the research community. For this reason, we believe there are opportunities to be exploited by combining the benefits of the different approaches. However, there are still some issues that have not been explicitly addressed in our study. An example is the way ontologies are used, since ontologies are currently a major technology for supporting service description and composition. Different organisations define ontologies in different ways, which may generate major problems of interoperability. Some approaches define manual mappings to deal with the ontologies interoperability problem; however these mappings or mediation techniques may be error prone and difficult to apply in realistic applications.

An issue that apparently is not being widely addressed is the support to end-users' service composition at runtime. This is a major research challenge and business opportunity, since the idea of delivering services at runtime on demand to end-users is a natural opportunity and benefit of service-oriented systems. However, as it can be observed from our study, the majority of the approaches mainly focus on design-time service composition, providing support to service developers. On this specific topic we foresee many research challenges as well as many business opportunities in the upcoming years.

References

[1] F. Lautenbacher, B. Bauer, "A Survey on Workflow Annotation & Composition Approaches", *In Proceedings of the Workshop on Semantic Business Process and Product Lifecycle Management (SemBPM)*, Innsbruck, Austria, June 2007, pp. 12-23.

[2] Seog-Chan Oh, Dongwon Lee and Soundar R. T. Kumara, "A Comparative Illustration of AI Planning-based Web Services Composition," *ACM SIGecom Exchanges*, Vol. 5, No. 5, 2005, pp. 1-10.

[3] N. F. Noy, "Semantic integration: a survey of ontology-based approaches", *ACM SIGMOD Record*, Vol. 33, No. 4, December 2004, pp. 65-70.

[4] M.H. ter Beek, A. Bucchiarone, and S. Gnesi , "Formal Methods for Service Composition", *Annals of Mathematics, Computing & Teleinformatics*, Vol 1, No 5, 2007, pp. 1-10.

[5] D. Griffin, D. Pesch, "A Survey on Web Services in Telecommunications". *IEEE Communications Magazine*, July 2007, Vol. 45, No. 7, pp. 28-35.

[6] J. Brønsted, K. M. Hansen, M. Ingstrup, "A Survey of Service Composition Mechanisms in Ubiquitous Computing", In *Proceedings of UbiComp 2007 Workshop*, June 2007, Vol. 4717, No. 9, pp. 87-92, Innsbruck, Austria.

[7] IFIP TC5 SIG on Enterprise Interoperability, "TC5 SIG EI Aims and Scope", January 2008. Soon available at http://www.ifip.org/.

[8] D. Chen, "Enterprise interoperability framework", In Proceedings of Enterprise Modelling and Ontologies for Interoperability, EMOI - Interop 2006, CEUR Vol. 200, 2006.

[9] M.-S. Le et al. (Eds.), "Enterprise interoperability research roadmap", V4.0, July 2006. Available at ftp://ftp.cordis.europe.eu/pub/ist/doc/directorate_d/ebusiness/ei-roadmap-final_eng.pdf.

[10] M.P. Papazoglou, D. Georgakpoulos, "Introduction to special issue on Service Oriented Computing", Communications of the ACM, Vol. 46, No. 10, 2003, pp. 24-28.

[11] R. Hull, M. Benedikt, V. Christophides, and J. Su. "E-Services: A Look Behind the Curtain". In *Proceedings of the PODS 2003 Conference*, San Diego, CA, USA, 2003.

[12] G. Alonso, F. Casati, H. Kuno, V. Machiraju, "*Web Services. Concepts, Architectures and Applications*", 2004, Springer-Verlag, Berlin Heidelberg.

[13] K. Verma, K. Gomadam, A. P. Sheth, et al.. "The METEOR-S Approach for Configuring and Executing Dynamic Web Processes", Technical Report , 2005.

[14] S. Topouzidou. "SODIUM, Service-Oriented Development In a Unified framework", Final report IST-FP6-004559. http://www.atc.gr/sodium.

[15] J. Pathak, S. Basu, V, Honavar. "Modeling Web Services by Iterative Reformulation of Functional and Non-functional Requirements", In *proceedings the 4th International Conference on Service-Oriented Computing (ICSOC)*, Chicago, USA, December 2006, pp. 314-326.

[16] The SeCSE team. "Designing and Deploying Service-Centric Systems: The SeCSE Way", In *proceedings of Workshop: Service Oriented Computing: a look at the Inside*, ICSOC 2007, Vienna, Austria, September 2007.

[17] SeCSE Project. "Specification Language Definition". *SeCSE project deliverable* (A1D23), 2007.

[18] D. Roman, et al.. "*Web Service Modeling Ontology"*, Applied Ontologies, vol. 1, pp. 77-106, 2005.