Towards Conceptual Foundations of Requirements Engineering for Services

Verlaine, Bertrand; Jureta, Ivan; Faulkner, Stephane

Published in:
Proceedings of the Fifth IEEE International Conference on Research Challenges in Information Science (RCIS 2011), Gosier, Guadeloupe

Publication date:
2011

Document Version
Early version, also known as pre-print

Towards Conceptual Foundations of Requirements Engineering for Services

Bertrand Verlaine*, Ivan J. Jureta* and Stéphane Faulkner*
*PReCISE Research Center
University of Namur
{bertrand.verlaine,ivan.jureta,stephane.faulkner}@fundp.ac.be

***Abstract*—A service-oriented system should be engineered to satisfy the requirements of its stakeholders. Requirements are understood in terms of stakeholder goals, softgoals, quality constraints, preferences, tasks, and domain assumptions. The service-oriented system is viewed in terms of services, mediators, choreographies, and orchestrations, among others. To engineer the system according to requirements, it is necessary to translate the requirements into the model of the service-oriented system. To do so, we must know the relations between the conceptualization of requirements and the conceptualization of services. Towards this aim, we propose and discuss relations between the Core Ontology for Requirements and the Web Service Modeling Ontology. We formalize both ontologies in a description logic and relate them via bridge rules in a distributed description logic. The two ontologies and the bridge rules together form conceptual foundations on which to build methodologies for the requirements engineering of service-oriented systems.**

***Index Terms*—Requirements Engineering for Service-oriented Systems, Ontology Mapping**

I. Introduction

Liu, Yu & Mei recently argued that "there is a dire need for systematic methods and automated facilities to handle requirements for services" [1]. In the context described by Liu and colleagues, a *service* is a self-describing and self-contained modular application designed to execute a well-delimited task, and that can be described, published, located, and invoked over a network [2], [3]. Raising this challenge was expected: to engineer a service-oriented system which satisfies the needs of its stakeholders, it is necessary to know their requirements. Hence the importance to follow a Requirements Engineering (RE) process by which the consumers of a services-oriented system are identified, their requirements are elicited, modeled and specified [4], [5], [6]. Once the consumers' requirements are known and specified, the technologies able to satisfy them can be mobilized, including those for, e.g., service discovery, selection, invocation, composition, interoperability, and so on. Considerable attention is being invested in those technical solutions for making service-oriented systems, less so in the RE of such systems [1], [7], [8], [9].

To enable what Liu and colleagues call the systematic handling of requirements [1], a methodology for RE is needed. It usually includes four components: (1) an ontology of requirements stating the information to elicit, and which relevantly describes the properties and behaviors expected of the system-to-be and its operational environment, (2) modeling primitives for the concepts and relations of the ontology, the instances of which together form models to record the elicited information, (3) often automated methods that could be applied to the models in order to answer questions of methodological interest, such as if a model is consistent, or if the properties and behaviors it attributes to the system-to-be will satisfy the requirements, and (4) guidelines on what steps to take when applying the said sets of tools. The first three components define a Requirements Modeling Language (an RML), the fourth says how the language ought to be used.

Service-orientation needs its *own* RE methodologies, i.e., Service-oriented RE (SRE) methodologies. Before efforts are invested in making them, it is crucial to answer the following question: *does service-orientation need **new** RE methodologies, or would the specialization of existing ones be enough?* The features that distinguish the service-oriented paradigm from other paradigms for software systems engineering may suggest that novelty of service-orientation requires novelty in its RE methodologies. A more cautious view is to consider that *new RE methodologies are needed only if models of the service-oriented system include information about requirements which cannot be elicited using current RE methodologies.* If so, then the ontology for requirements — the first component of an RE methodology — would need to incorporate new concepts and relations, and as a consequence, changes would need to be made to the remaining three parts of a methodology: new modeling primitives for requirements models would be necessary, as well as new reasoning means over requirements models, and new guidelines for the use of the new components together with established ones. Whether service-oriented systems warrant *new* RE methodologies is not a settled question. It will remain open as long as significant new advances are being made in service-oriented computing. Even though an once-and-for-all answer is elusive, more specific, but still useful ones can, once we chose a particular ontology for service-oriented systems modeling and an ontology for requirements.

This paper takes the *Core Ontology for REquirements* (CORE) [10] and the *Web Service Modeling Ontology* (WSMO) [11], and establishes how the information contained in instances of the concepts and relations of CORE is transferred into the instances of the concepts and relations of WSMO. To do so, we formalize both ontologies in a description logic and relate them via bridge rules in a distributed description logic. The results are three contributions:

1) We establish that if a service-oriented system-to-be

is to be modeled via the concepts and relations of WSMO (so that the models are written using the Web Service Modeling Language (WSML) [12]), then every RE methodology based on CORE can be used as-is for the RE of the system-to-be.

2) The bridge rules from CORE to WSMO state how information gets translated from models of requirements to models of the service-oriented system. Given an instance of a CORE concept, the bridge rules tell us what WSML concepts are to be instantiated to capture the information from the CORE concept instance.
3) The two ontologies and the bridge rules form conceptual foundations for the design of RE methodologies for service-oriented systems. A SRE methodology should support the elicitation, modeling, and analysis of requirements, and then the translation of a requirements model into a model of the service-oriented system. CORE states what information ought to be elicited and represented in the model of requirements, WSMO says what information should be found in the model of the service-oriented system, and the bridge rules indicate how the information from a requirements model translates into the model of the service-oriented system. Starting from CORE, WSMO and the bridge rules, the designer of the methodology can focus on making the remaining parts, that is to say the choice of the modeling primitives for requirements, the reasoning methods, and the guidelines for use.

The rest of the paper is organized as follows. We sketch the CORE ontology (§II) and the WSMO ontology (§III), along with their formalizations. We then present and discuss the bridge rules between those two ontologies (§IV). After a survey of related work (§V), conclusions and directions for future work are given (§VI).

II. The requirements ontology

First, we briefly explain the choice of CORE as the requirements ontology (§II-A). Then, we introduce CORE (§II-B) and formalize it (§II-C).

A. Choice of a Requirements Ontology

The concept of *requirement* as well as some of its subconcepts, i.a., the notion of goal, softgoal or assumption, have been discussed at length in the research on RE (e.g., [4], [6], [13], [14], [15], [16], [17]). However, none of those works propose a simple but complete description of all types of requirements along with the relations between them. The CORE ontology offers this set of essential concepts for RE, by covering the main notions that were previously identified and described, and by defining them within a single and comprehensive ontology.

B. Outline of the Core Ontology for Requirements

The root concept of CORE is Communicated information[1], specialized as follows:

- Goal, specialized on Functional goal, Quality constraint and Softgoal;
- Plan;
- Domain assumption, specialized on Functional domain assumption, Quality domain assumption and Soft domain assumption;
- Evaluation, specialized on Individual evaluation and Comparative evaluation.

A basic idea of CORE is that requirements are communicated by the stakeholders to the requirements engineer, so that the latter classifies requirements based on how and what was communicated [10], [18]. The Communicated information class is a catchall one, the instances of which are propositions communicated by the stakeholders. Once a communicated information is available, the question to ask is what mode was that proposition communicated in. The notion of mode (or modus in linguistics) reflects the idea that we can distinguish between the content of a communication and the intentional state it was communicated in, whereby different kinds of mode correspond to different intentional states of the stakeholder. If the stakeholder tells the engineer that she believes that some condition holds in the operating environment of the system-to-be, then the proposition stating the condition is an instance of the Domain assumption class. If she instead desires that the condition be made to hold by the system-to-be, then the proposition is an instance of the Goal class. In case an intention to perform particular actions is conveyed, which may then be delegated to the system-to-be, the engineer classifies the propositions describing these actions as instances of the Plan class. Since stakeholders can also indicate that they prefer some goals to be satisfied than others, or that some of them must be satisfied, while others are optional, CORE includes the class of Evaluation.

CORE distinguishes three kinds of goals. Functional goal refers to a desired condition the satisfaction of which is verifiable and is binary, i.e., it is either satisfied or not. Functional goal always represents either an event, a state or a process. The Quality constraint class defines desired values of non-binary measurable properties of the system-to-be (e.g., how many seconds it takes to answer a query). It has thus a quality space with a shared structure. As functional goals and quality constraints are not necessarily known at the very start of the RE process, the Softgoal class is instantiated to capture requirements that refer to vague properties of the system-to-be (e.g., that it is "fast"). Same specialization applies to the Domain assumption class, which has its functional variant (which refers to binary properties of the system-to-be and/or its environment), its quality variant, Quality domain assumption, and its soft variant, Soft domain assumption. Finally, evaluations can qualify positively or negatively individual requirements (as instances of Individual evaluation) or via instances of Comparative evaluation which compare goals, domain assumptions, and/or plans.

[1] A CORE concept is denoted as Class and an instance thereof is denoted instance.

TABLE I
CORE WRITTEN IN DESCRIPTION LOGIC $\mathcal{SIN}$

1 :	COMMUNICATED INFORMATION	$\equiv$	GOAL $\sqcup$ PLAN $\sqcup$ DOMAIN ASSUMPTION $\sqcup$ EVALUATION
2 :	$\bot$	$\sqsubseteq$	GOAL $\sqcap$ PLAN $\sqcap$ DOMAIN ASSUMPTION $\sqcap$ EVALUATION
3 :	**refine**	$\equiv$	**refined-by**$^-$
4 :	**refined-by**	$\equiv$	**refine**$^-$
5 :	$\top$	$\sqsubseteq$	$\forall$ **refine**.COMMUNICATED INFORMATION
6 :	$\forall$ **refine**.GOAL	$\equiv$	FUNCTIONAL GOAL $\sqcup$ QUALITY CONSTRAINT $\sqcup$ SOFTGOAL
7 :	$\bot$	$\sqsubseteq$	FUNCTIONAL GOAL $\sqcap$ QUALITY CONSTRAINT $\sqcap$ SOFTGOAL
8 :	**approximate**	$\equiv$	**approximated-by**$^-$
9 :	**approximated-by**	$\equiv$	**approximate**$^-$
10:	SOFTGOAL	$\sqsubseteq$	$\exists$ **approximate**.QUALITY CONSTRAINT
11:	$\forall$ **refine**.DOMAIN ASSUMPTION	$\equiv$	FUNCTIONAL DOMAIN ASSUMPTION $\sqcup$ QUALITY DOMAIN ASSUMPTION $\sqcup$ SOFT DOMAIN ASSUMPTION
12:	$\bot$	$\sqsubseteq$	FUNCTIONAL DOMAIN ASSUMPTION $\sqcap$ QUALITY DOMAIN ASSUMPTION $\sqcap$ SOFT DOMAIN ASSUMPTION
13:	SOFT DOMAIN ASSUMPTION	$\sqsubseteq$	$\exists$ **approximate**.QUALITY DOMAIN ASSUMPTION
14:	$\forall$ **refine**.EVALUATION	$\equiv$	COMPARATIVE EVALUATION $\sqcup$ INDIVIDUAL EVALUATION
15:	$\bot$	$\sqsubseteq$	COMPARATIVE EVALUATION $\sqcap$ INDIVIDUAL EVALUATION

C. The CORE Ontology in Description Logic

We use the Description Logic (DL) $\mathcal{SIN}$ [19] to rewrite each ontology. This rewriting is needed to map the CORE and WSMO concepts (see §IV) with the Distributed Description Logic (DDL) [20] (see §IV-C).

Brief overview of the DL rationale. The DL languages are used to represent and to structure the knowledge of a domain of interest. In the scope of this work, these are the requirements and the service domains. The basic DL language is $\mathcal{AL}$. It allows us to specify, i.a., the intersection of two concepts ($A \sqcap B$), the universal restriction ($\forall r.A$: all instances having the relation r with at least one instance of A) and the limited existential quantification ($\exists r.\top$: it exists instances having the relation r). In addition to these constructors, we use for formalizing CORE and WSMO the union ($A \sqcup B$), the full existential quantification ($\exists r.B$: instances of B having the relation r), the equivalence ($A \equiv B$: each instance of A has a corresponding instance in B, and inversely), the subsume relation ($A \sqsubseteq B$: each instance of A has a corresponding instance in B), the inverse relation (r^-: a relation r has the opposite meaning of r^-) and the number restriction ($\leq n\ r.A$: each instance of A has at most n relation r; $\geq n\ r.B$: each instance of B has at least n relation r). All these constructors are part of the $\mathcal{SIN}$ logic.

Line 1 of Table I defines the root concept of the CORE ontology. Requirements expressed during RE are categorized into the four main classes of CORE, i.e., `Goal` defined by Line 6, `Plan` which has no subclasses, `Domain assumption` defined by Line 11 and `Evaluation` defined by Line 14. This specialization enables to classify requirements in the end-leaves of the CORE ontology, i.e., `Quality constraint`, `Soft domain assumption`, `Comparative evaluation`, and so on. Detailed informal definitions of the CORE concepts are not repeated here. Unchanged `softgoals` and `soft domain assumptions` cannot be propagated to the level of service descriptions because of their vagueness. They need to be replaced by more precise requirements. Just as, say, imprecise `goals` are refined, so are `softgoals` and `soft domain assumptions` approximated [18], [10], whereby their approximations involve the identification of `quality constraints` and `quality domain assumptions`, while `comparative evaluations` may indicate how alternative `quality constraints` or `quality domain assumptions` may rate in terms of relative desirability. Lines 10 and 13 capture these ideas.

III. THE SERVICE MODELING ONTOLOGY

This section has the same structure as §II: we first explain the choice of WSMO as the service ontology (§III-A), then we sketch (§III-B) and formalize it (§III-C).

A. Choice of a Service Ontology

We have chosen between two ontologies: the WSMO ontology and the Semantic Markup for Web Services (OWL-S) ontology [21], [22], previously called DAML-S.

Here are the main arguments which lead us to choose the WSMO ontology:

1) In constrast to OWL-S, WSMO separate the service consumer view from the service provider view. Research has mainly focused on the service provider perspective [8]. It is relevant to separate the two viewpoints in order to propose a SRE methodology which can accommodate the two perspectives and thereby ensure a separation of concerns.
2) From a practical point of view, the ESSI working group[2] proposes several languages, technologies and tools specifically built to support the use of the WSMO ontology. These are mainly the WSML language [12], [27] which allows to specify the instances of the WSMO concepts, and the Web Service Execution Environment (WSMX)

[2]The European Seamntic Systems Initiative (ESSI) is now part of the community of the Semantic Technology Institute International. See respectively http://www.essi-cluster.org/ and http://community.sti2.org/.

TABLE II
THE SUBCLASSES OF THE WSMO CLASS

WSMO concept	Purpose
NonFunctionalProperties	Specification of the service aspects which are not directly related to its core functionality [23], among others: Accuracy, Financial, Network-related QoS, Reliability, Scalability and Security. The other properties are mainly related to the characteristics of the service description itself and do not add any constraints over what a service can or cannot do, or how it can do it [24].
Ontology	Used to import other ontologies.
__Mediator	Used to resolve the heterogeneity problems between the linked elements in the service specification, i.e., *ontologies* or *goals*.
Capability	Specification of the service functionality which is unique. *Capability* is specialized on *Shared variables*, *Precondition*, *Assumption*, *Postcondition* and *Effect*.
Shared variables	Specification of the variables shared by the other *Capability*'s subclasses.
Precondition	Specification of the required state of the data space, i.e., the input data, before the execution of the service in order to enable the service to provide its value.
Assumption	Specification of "the [required] state of the world before the execution of the service. Otherwise, the successful provision of the service is not guaranteed" [23].
Postcondition	Specification of the guaranteed state of the data space, i.e., the output data, after the successful execution of the service.
Effect	Specification of the guaranteed state of the world after the successful execution of the service.
Interface	Description of how the *capability* of the service can be fulfilled. *Interface* is specialized on *Choreography* and *Orchestration*.
Choreography	Specification of the communication process to follow in order to use the service. To describe these interactions between the service and its user, WSMO uses the notion of Abstract State Machine (ASM) [25]. In our case, a *choreography* is a state machine having its states described in terms of *concepts*, their *relations* and *functions* as well as its transition rules from one state to the other [23], [26].
Orchestration	Specification of possible functionalities used statically of dynamically by the service from other services in order to achieve its *capability*.

tool [28], [29] which supports the description of offers and requests of semantic services. It also supports the service publishing, discovering and selection processes. Working with WSMO along with the technical solutions supporting its use opens more perspectives than OWL-S for the future work.

3) WSMO allows to describe each of its elements with non-functional properties while OWL-S restricts the use of non-functional properties to the description of the profiles.

B. Outline of the WSMO Ontology

The WSMO ontology "provides the conceptual underpinning [...] for semantically describing all relevant aspects of [...] services in order to facilitate the automatization of discovering, combining and invoking electronic service over the Web" [11]. In other words, WSMO aims at describing all the relevant aspects of a service, i.e., its concepts and their relations [11], [23], [24]. Its model layer is composed of four classes: *Ontology*[3], *Service*, *Goal* and *Mediator*, which are specified through the Meta Object Facility (MOF) notation [30].

The *Ontology* class "provides the [agreed] terminology used by other WMSO [top-level classes] to describe the relevant aspects of the domain of discourse" [24]. An *Ontology* consists of non-functional properties (i.a., the creator name, the language used, the unique identifier of the service, the version of the service being described, and so on [23]), imported ontologies, ontology mediator, and the definition of concept, relation, function, instance and axiom. For further details, see [11], [23], [24].

The *Goal* class models the service consumer view of the service she is looking for by representing her needs in terms of functionality, behavior and QoS [11]. The MOF model of *Goal* is as follows[4]:

Class goal
  hasNonFunctionalProperties **type** nonFunctionalProperties
  importsOntology **type** ontology
  usesMediator **type** ooMediator, ggMediator
  requestsCapability **type** capability $multiplicity = single\text{-}valued$
  requestsInterface **type** interface

Table II gives the definitions of the *Goal* subclasses. Note the attributes *Precondition*, *Assumption*, *Postcondition* and *Effect* are all *axioms*, i.e., logical expressions. The *Capability* and the *Interface* classes have three others subclasses than those underlined in Table II: *nonFunctionalProperties*, *ontology* and *__Mediator*. Seeing that they have no significance in the scope of this work –they are only used for intra-service relations–, they are ignored.

The third class of the model layer of WSMO is *Service*. This class describes the computational entity exposing a service through a Web-based interface (e.g., a Web Service

[3]A WSMO concept is denoted as *Class* and an instance thereof is denoted *instance*.

[4]Note each attribute has a default implicit multiplicity set to multi-valued. Otherwise, the multiplicity is explicit.

TABLE III
WSMO WRITTEN IN DESCRIPTION LOGIC $\mathcal{SIN}$

16:	GOAL	$\equiv$ NON FUNCTIONAL PROPERTY $\sqcap$ CAPABILITY $\sqcap$ INTERFACE
17:	GOAL	$\sqsubseteq$ SERVICE USER NEEDS
18:	GOAL	$\sqsubseteq =1$ **composed-by**.CAPABILITY
19:	**compose**	$\equiv$ **composed-by**$^-$
20:	**composed-by**	$\equiv$ **compose**$^-$
21:	NON FUNCTIONAL PROPERTY	$\equiv$ BEHAVIOURAL SERVICE PROPERTY $\sqcup$ NON-RESTRICTIVE SERVICE PROPERTY
22:	CAPABILITY	$\equiv$ NON FUNCTIONAL PROPERTY $\sqcap$ SHARED VARIABLES $\sqcap$ PRECONDITION $\sqcap$ ASSUMPTION $\sqcap$ POSTCONDITION $\sqcap$ EFFECT
23:	AXIOM	$\sqsupseteq$ PRECONDITION $\sqcup$ ASSUMPTION $\sqcup$ POSTCONDITION $\sqcup$ EFFECT
24:	AXIOM	$\equiv$ LOGICAL EXPRESSION $\sqcap$ NON-RESTRICTIVE AXIOM PROPERTY
25:	INTERFACE	$\equiv$ NON FUNCTIONAL PROPERTY $\sqcap$ CHOREOGRAPHY $\sqcap$ ORCHESTRATION
26:	CHOREOGRAPHY	$\sqsubseteq$ STATE MACHINE
27:	STATE MACHINE	$\equiv$ STATE $\sqcap$ TRANSITION RULE
28:	STATE MACHINE	$\sqsubseteq \geq 2$ **sequence**.STATE
29:	**sequence**	$\equiv$ **sequenced-by**$^-$
30:	**sequenced-by**	$\equiv$ **sequence**$^-$

(WS) based on the WSDL [31], SOAP [32] and UDDI [33] technologies) [11], [24]. A *service* has the same types of attributes as a *goal*, i.e., *NonFunctionalProperties*, *Ontology*, *__Mediator*, *Capability* and *Interface*.

The last class, *Mediator*, "describes elements that overcome [data, process and protocol] interoperability problems between different WSMO elements" [24]. It can help to resolve data, process, and protocol incompatibilities.

In the scope of our project, i.e., the specification of the service consumer requirements within the service oriented paradigm, the relevant class of WSMO to work with is the *Goal* class because it allows to specify the requirements of service consumers. Therefore, this class is formalized in description logic (see Table III), and mapped with the CORE ontology (see §IV).

C. The WSMO Goal Concept in Description Logic

For the WSMO formalization, we ignore attributes having the *Ontology* type or the *__Mediator* type which are normally present in each main class, i.e., classes formalized by Lines 16, 22 and 25.

Line 16 defines the root concept of the WSMO ontology. A *goal* describes the user's needs concerning the service use (Line 17) and it has only one *capability* (Line 18). *Non functional property* (Line 21) is composed of the properties of the service concerning its behavior as well as properties of the service description (e.g., creation date, author(s), identifier, and so on). These do not constrain the service. This distinction is not expressly made in the WSMO ontology, but the distinction between two types of *non-functional properties* is admitted [11]. The *Capability* class (Line 22) consists of several different axioms (Line 23) apart from some *non-restrictive properties*. An axiom represents a logical expression (Line 24). The *Choreography* composing an *Interface* (Line 25) is a state machine (Line 26) organizing the transition between several states (Lines 27 and 28). The *Orchestration* class is not deeper detailed in Table III (see §IV-A for the argument).

IV. BRIDGING THE TWO ONTOLOGIES

The objective of this section is to link similar classes of each ontology representing the same objects, i.e., service consumer's requirements concerning a service to select. The methodology observed is as follows. First, the relevant classes of the WSMO ontology are highlighted (§IV-A) to be then mapped to the four main CORE classes (§IV-B). Afterwards, we refine and formalize the mapping between the two ontologies: each relevant WSMO class is mapped to the corresponding class(es) of the CORE ontology (§IV-C).

In this work, we do not care about the specification of the needs expressed by the service consumer, i.e., with which language the requirements and the service request can be specified. Nevertheless, we illustrate the discussion about our mappings with the WSML language, one of the potential languages to represent syntactically the WSMO ontology, and with service consumer's requirements expressed in natural language. A hotel booking example is used, for which the domain ontologies, i.e., the ontology of a hotel, of a booking, of a credit card, of specific QoS service properties, and so on are supposed known and shared among the stakeholders. In §IV-B, requirements expressed by the service consumer are stated in natural language. Those requirements are classified into one of the main CORE classes, i.e., `Goal`, `Plan`, `Domain assumption` or `Evaluation`. In §IV-C, they are refined and, for some of them, the corresponding WSML specification is proposed.

A. Relevant WSMO Classes

Relevant WSMO classes selected for the mapping are: *Behavioural service property*, *Precondition*, *Assumption*, *Postcondition*, *Effect* and *Choreography*. The other WSMO classes are not selected for the following reasons which are related to the definition of the CORE concept `Communicated information`:

- *Non-restrictive service property*: the objectives of the service user are to set constraints concerning the service she will use, i.e., what the service does and/or how the

TABLE IV
MAPPING OF THE WSMO CLASSES WITH THE CORE CLASSES [5].

	Behavioural service property	*Precondition*	*Assumption*	*Postcondition*	*Effect*	*Choreography*
Goal	✓	✗	✗	✓	✓	✓
Plan	✗	✓	✓	✗	✗	✓
Domain assumption	✗	✓	✓	✗	✗	✗
Evaluation	✓	✓	✓	✓	✓	✓

service does it. This concept cannot be used for that purpose.

- *Shared variable*: All *shared variables* are indirectly used in at least one of the following classes *Precondition*, *Assumption*, *Postcondition* and *Effect* which have been selected. Moreover, the service consumer cannot evaluate the definition of a variable but only what he has to give as input data and what he receives as output data.
- *Orchestration*: This class referencing the functionalities provided by other services is indirectly included in *Capability* which resume the global functionality of the service being described. This (global) service can call other services in order to fulfil its capability; that is specified by an *orchestration*.

B. Mapping the WSMO Classes to the Four Main CORE Classes

For each possible association between a WSMO class and a CORE class, we check if a service consumer can express a requirement corresponding to the content of the WSMO class. Table IV summarizes the development of this step; explanations of the mappings follow.

A goal represents some conditions not yet satisfied that the service consumer desires[6] to see become true in the future system [10]. Requirements 1 and 2 are examples of goals related to the hotel booking example.

Requirement 1. goal: The service consumer wants to book a hotel.

Requirement 2. goal: The service consumer wants that the service answers quickly.

The service consumer can express a desire, i.e., a goal, concerning QoS properties which are *Behavioural service properties* (e.g., Requirement 2). Goal is not mapped with *Precondition* and *Assumption*. By specifying variables to be provided in order to use the service or the state of the world before the service use, *preconditions* and *assumptions* do not correspond to conditions not yet satisfied that the service consumer wants to see become true after the service use. A *postcondition* and an *effect* respectively describe the state of some variables or the state of the world after the service use. This corresponds to a goal that the service consumer wants to see achieved by the service. Requirement 1 captures these kinds of stakeholders' needs; this requirement must be refined (see §IV-C). The service consumer can express desires concerning the communication process, i.e., which and how are the messages exchanged between the service consumer and the service provider. This communication process can be specified through the *Choreography* class.

Secondly, a plan can be defined as the specification of the service consumer's intentions to perform, conditionally or not, the action(s) described (e.g., Requirement 3).

Requirement 3. plan: The service consumer intends to pay for the accommodation.

Plan is not mapped with *Behavioural service property* because the service consumer cannot have intentions concerning elements belonging to *Behavioural service property*, which can only be used to describe the QoS level delivered by the service provider. An intention, i.e., a plan, of the service consumer can be to provide data to the service which is specified into a *precondition*. Similarly, he can promise to carry out some actions in order to bring the state of the world in a predetermined shape, i.e., an *assumption* (Requirement 3 is an example of such intentions). The values of variables or the state of the world after the service execution do not concern the service consumer's intentions; this is why Plan is not mapped with *Postcondition* or with *Effect*. Lastly, the service consumer can intend to send specific messages and data during the communication process with the service provider. This allows us to map Plan to *Choreography*.

A domain assumption means that the content of the communicated information expressed is believed true by the service consumer or, through the speech act expressed by the latter, the domain assumption makes the content of the communicated information true (e.g., the service consumer says: *"My Web server will use a Linux Operating System (OS)"*, and then his computer department actually uses a Linux OS). Requirements 4 and 5 are examples of domain assumptions.

Requirement 4. domain assumption: The service consumer believes that the provided credit card number is correct.

Requirement 5. domain assumption: The service consumer believes that he will land the 21^{st} of July.

Domain assumption is not mapped with *Behavioural service property*. Because most of the QoS properties are adaptable to

[5] In Table IV, the sign "✓" means that the WSMO class is mapped with the corresponding CORE class. Otherwise, the sign "✗" is used.

[6] This informal definition as well as the following ones are about a service consumer. The CORE ontology can be used to convey requirements of a stakeholder concerning other types of system-to-be, which can be a software or not.

the requirements of the service consumer –the service provider proposes the same functional service with different levels of QoS [34]–, the service consumer is not expected to have any beliefs about the QoS properties of the service she is looking for. She also cannot make true a QoS property alone seeing that it must be negotiated –to some extent– with the service provider. Requirements capturing beliefs expressed by the service consumer –domain assumptions– can be related to the value of some variables, i.e., a *precondition*, or to the state of the world, i.e., an *assumption*, that he has to satisfy before the service use whatever their real state. Requirement 4 and 5 are examples of, respectively, a *precondition* and of an *assumption*. The Domain assumption class is not mapped with *Postcondition* and *Effect*: the values of the output data and the state of the world after the service use shape the added value of the service. When the service consumer looks for a specific service among services fully specified, he is not expected to express any beliefs about what the service effectively does, captured in *postconditions* and *effects*. A "belief" concerning an output should be seen as a constraint on the service sought, i.e., a goal. Moreover, the results of the service use are carried out by the service provider. This means that the service consumer cannot make true the content of a *postcondition* or an *effect* by himself. Lastly, the service consumer is not expected to express beliefs about the description of how the data is exchanged with the service provider who (almost) always sets the communication process. Likewise, the service consumer cannot make true a specific communication process which partially depends on the service provider. So, Domain assumption is not mapped with *Choreography*.

Finally, the Evaluation class specifies the preferences (or the appraisal) of the service consumer about a single condition or between conditions that may hold. Requirements 6, 7 and 8 are examples of evaluations which are related to the hotel booking example.

Requirement 6. evaluation: The service consumer prefers a response time of 500 ms to a response time of 700 ms.

Requirement 7. evaluation: The service consumer does not appreciate paying the accommodation at its booking time.

Requirement 8. evaluation: The service user appreciates a large availability.

Evaluation is bridged to the six WSMO concepts seeing that the service consumer can express a preference between two WSMO instances, or he can appraise negatively or positively a specific communicated information.

C. The Mappings Between CORE and WSMO

Table V illustrates the mapping between the CORE and the WSMO ontologies using the DDL [20]. In that table, the DDL sign used, $A \xrightarrow{\equiv} B$, means that the mapping is complete, i.e., each instance of the concept A can be bridged with one of the instances of the corresponding concept B. Each concept used in Table V is prefixed with *'CORE:'* or with *'WSMO:'* respectively to remind that the class belongs to CORE or to WSMO.

In CORE, a goal is either a functional goal or a quality constraint. Given that a softgoal is vague, early requirements classified as softgoals must be approximated to become quality constraints. For instance, Requirement 2 is a softgoal; it is approximated by Requirement 9, which is a quality constraint.

The non-functional properties of a service, i.e., its *behavioural service properties*, are not related to an event, a process or a state, but they describe some measurable quality aspects of the service. The *Behavioural service property* class is thus mapped with the quality constraint class as formalized through Line 32. Note Specification 1 specifies a *behavioural service property* captured by Requirement 9. The *Postcondition* class describes the state of the data space. It can be linked with Functional goal (Line 31) because the targeted data space, which is a state, is reached or not. With a similar reasoning, the *Effect* class can be linked with Functional goal. The last class to map with a subclass of Goal is *Choreography*. The latter is also mapped with Functional goal (Line 31) because a *choreography* describes a process (i.e., the communication process). Requirements 10 and 11 refine Requirement 1 by using, e.g., the "HOW" question [35], [36]; Requirement 10 is specified in Specification 2.

Requirement 9. quality constraint: The service consumer wants an answer within 500ms.

```
Specification 1.
 nonFunctionalProperty
   performance hasValue ?performance
     definedBy
       ?performance[responseTime hasValue ?responseTime]
           memberOf QoSproperty
       and ?responseTime[delay hasValue ?time, units
           hasValue millisecond]
       and lessEqual(?time,500)
```

Requirement 10. functional goal: The service consumer wants to receive the booking information.

```
Specification 2.
 postcondition
   nonFunctionalProperty
     description hasValue "The output has to be the address
         of the hotel, the booking number as well as the arrival
         and departure date"
   endNonFunctionalProperty
   definedBy
     ?hotelBookingInfo memberOf booking
     and ?hotelBookingInfo[hasAddress hasValue ?address,
         hasArrivalDate hasValue ?arrivalDate,
         hasDepartureDate hasValue ?departureDate,
         hasBookingId hasValue ?bookingId]
```

Requirement 11. functional goal: The service consumer wants that the nights paid are actually booked for him.

The Plan class does not have subclasses. Line 33 formalizes

TABLE V
THE MAPPING BETWEEN CORE AND THE WSMO CLASSES FORMALIZED WITH DDL

31:	*CORE*:FUNCTIONAL GOAL	$\stackrel{\equiv}{\longrightarrow}$	*WSMO*:POSTCONDITION ⊔ *WSMO*:EFFECT ⊔ *WSMO*:CHOREOGRAPHY
32:	*CORE*:QUALITY CONSTRAINT	$\stackrel{\equiv}{\longrightarrow}$	*WSMO*:BEHAVIOURAL SERVICE PROPERTY
33:	*CORE*:PLAN	$\stackrel{\equiv}{\longrightarrow}$	*WSMO*:PRECONDITION ⊔ *WSMO*:ASSUMPTION ⊔ *WSMO*:CHOREOGRAPHY
34:	*CORE*:FUNCTIONAL DOMAIN ASSUMPTION	$\stackrel{\equiv}{\longrightarrow}$	*WSMO*:PRECONDITION ⊔ *WSMO*:ASSUMPTION
35:	*CORE*:QUALITY DOMAIN ASSUMPTION	$\stackrel{\equiv}{\longrightarrow}$	*WSMO*: ⊥
36:	*CORE*:INDIVIDUAL EVALUATION	$\stackrel{\equiv}{\longrightarrow}$	*WSMO*:BEHAVIOURAL SERVICE PROPERTY ⊔ *WSMO*:PRECONDITION ⊔ *WSMO*:ASSUMPTION ⊔ *WSMO*:POSTCONDITION ⊔ *WSMO*:EFFECT ⊔ *WSMO*:CHOREOGRAPHY
37:	*CORE*:COMPARATIVE EVALUATION	$\stackrel{\equiv}{\longrightarrow}$	*WSMO*:BEHAVIOURAL SERVICE PROPERTY ⊔ *WSMO*:PRECONDITION ⊔ *WSMO*:ASSUMPTION ⊔ *WSMO*:POSTCONDITION ⊔ *WSMO*:EFFECT ⊔ *WSMO*:CHOREOGRAPHY

the content of Table IV concerning the Plan class. Requirements 12 and 13 refine Requirement 3. The WSML excerpt of Requirement 13 is given in Specification 3.

Requirement 12. plan: The service consumer promises to provide his credit card data when booking.

Requirement 13. plan: The service consumer will have enough money to book the hotel.

```
Specification 3.
 assumption
   nonFunctionalProperty
     description hasValue "The credit card balance is
         sufficient"
   endNonFunctionalProperty
   definedBy
     ?balanceCard memberOf card
     and ?bookingPrice memberOf booking
     and greaterEqual(?balanceCard,?bookingPrice)
```

A domain assumption is either a functional domain assumption or a quality domain assumption; potential soft domain assumptions must be approximated in order to share a common measurement scale and thus become quality domain assumption. The *Precondition* class describes the state of the data space before the service use (e.g., the *precondition* captured Requirement 14, which refines Requirement 4, and specified in Specification 4). As formalized through Line 34, it can be linked with Functional domain assumption because the targeted data space before the service use, which is a state, is satisfied or not. With a similar reasoning, the *Assumption* class can be linked with Functional domain assumption. Requirement 15, refined from Requirement 5, is a functional domain assumption. Note that there is no mapping between a WSMO class and Quality domain assumption (formalized by Line 35). This situation was expected due to the nature of the technologies used to implement services, e.g., WS technologies. The quality properties of the environment mainly concern the network infrastructure which is inherently unreliable. Therefore, the service consumer cannot believe true or make true the value of a non-functional characteristic of the environment.

Requirement 14. functional domain assumption: The service consumer believes that the provided credit card number coincides with the actual number of his credit card.

```
Specification 4.
 precondition
   nonFunctionalProperty
     description hasValue "The credit card number is provided
         as input"
   endNonFunctionalProperty
   definedBy
     ?cardInfo memberOf card
     and ?cardInfo[hasCardNumber hasValue ?cardNumber,
         hasValidityDate hasValue ?validityDate]
```

Requirement 15. functional domain assumption: He believes that his plane will land the 21^{st} of July.

An evaluation is either specialized on an individual evaluation or a comparative evaluation. The *Behavioural service property* class is linked with the two subclasses of Evaluation: the service consumer can compare two *behavioural service properties* (e.g., Requirement 16 refining Requirement 6), but he can also rate negatively of positively a single *behavioural service property* (e.g., Requirement 17 refined from Requirement 8). With a similar reasoning, we can say that *Precondition*, *Assumption*, *Postcondition*, *Effect* and *Choreography* are all mapped with Individual evaluation and with Comparative evaluation. This is formalized through the Lines 36 and 37. A last example is Requirement 18, refined from Requirement 7, which is an individual evaluation of the plan stated in Requirement 12.

Requirement 16. comparative evaluation: The service consumer prefers a response time of 500 ms to a response time of 700 ms.

Requirement 17. individual evaluation: The service consumer appreciated an availability of 97%.

Requirement 18. individual evaluation: The service consumer can negatively assess to pay the booking in advance.

Clearly, the WSML language does not offer the constructors to specify comparative evaluations and individual evaluations expressed by the service consumers. If WSML is chosen to specify the WSMO concept, then some requirements of a service consumer will not be specified using WSML. This issue is left for future work.

V. Related Work

Most often, the service engineering field is interested in the service provider's point of view (e.g., [37], [38], [39]) or focuses on the monitoring of the service consumer's requirements (e.g., [40], [41]). They study languages, models, methodologies and/or techniques used to (re)engineer a service which is then exposed through a Web-based interface and monitored by its provider.

Some research projects address the issue of service selection based on consumers' requirements. However, the requirements elicitation task is rarely well defined, and the links between RE and the service oriented paradigm are not as clear (e.g., [42], [43], [44]). Only a few works cover the service engineering from the customer point of view, i.e., how the elicitation of her needs can be achieved and then used in order to select the more accurate service. The targeted goal is clearly to reach an automation of most tasks of the SRE process (i.e., mainly the consumer requirements elicitation and specification, the service selection and eventually the service composition at runtime, and the service monitoring).

Grandry et all. [45] propose a framework to capture requirements related to QoS and manage them. One important feature of the technique is the traceability of the requirements from the business level to the software level. However, this work does not propose a solution to express the elicited requirements into machine-processable technologies. They only focus on non-functional requirements.

Rolland et all. [46] introduce a model for Intentional Service Modelling (ISM). Service providers have to describe their WSs in an intentional way. Service consumers use an intentional matching mechanism to select potential WSs. In this work, the RE process to elicit the service consumer needs is not well defined and some potential requirements cannot be taken into account (mainly QoS related requirements). Another relevant paper [47] uses the ISM approach. The authors improve the work of Rolland et al. by taking into account the QoS levels of WSs during the matching and selection steps. The Service-Based Applications (SBA) must be modeled in terms of stakeholders' requirements, and not in terms of technical and procedural aspects. Similar to the work of Rolland and colleagues, the use of ISM requires that both the service consumers and/or the requirements engineers, and the service providers learn to use and work within the ISM approach.

In [48], the authors propose a method and a tool which allow the service users to express their requirements. The tool analyzes them in order to help the users during the requirements refinement process and in discovery of errors and conflicts. The authors create their own meta-model for the four elements required in service consumption (i.e., role, goal, process and service).

The last significant work related to ours is the Service Centric System Engineering (SeCSE) project [49]. This project aims at increasing the accuracy of services selected based on textual requirements expressed by the service consumer. SeCSE takes into account two main challenges present in all RE projects: the incompleteness and the ambiguity of the propositions. UCaRE, based on the VOLERE methodology, is the application used by requirements engineers to model textual requirements expressed by service consumers. The service selection is based on a discovery algorithm, EEDiE, which uses WordNet and focuses on the disambiguation and completeness of the requirements [49], [50]. The scope of our work is more restricted than the whole SeCSE solution. Our main contribution is the use of comprehensive ontologies, i.e., CORE and WSMO, covering all main concepts that can be used respectively in RE and in service requests engineering. Only two types of requirements can be modeled with the SeCSE system: functional and quality requirements. Some potential requirements, such as evaluation or assumption, cannot be differentiated from the others. Moreover, by using WSMO as the service ontology, we allow to specify the service proposals and the service requests with semantics languages such as WSML. SeCSE has been built to select services expressed with the traditional WS technologies such as UDDI and WSDL. However, semantic services are more and more presented as the next significant evolution of the service oriented paradigm [23], [51].

VI. Conclusions

The service oriented paradigm has been raising new issues, including the need for methodologies enabling the elicitation of the service consumer's requirements. Because software consumers often evaluate the system built in comparison with their needs, a correct elicitation and management of those requirements are significant issues to solve. An RE methodology should enable us to solve these issues, while being adapted to the specifics of service-oriented computing. This is why we present a conceptual and formal mapping between a requirements ontology and a service ontology: the correspondences between the concepts of the requirements ontology and those of the service ontology enable, on the one hand, RE engineers to handle concepts and relations they know, and, one the other hand, to capture and to specify the service requirements in technologies used in service computing.

The work proposed in this paper allows (i) to consider the creation or the modification of an RE methodology adapted to the service oriented paradigm which can now be grounded on the conceptual mapping between CORE and WSMO, (ii) to identify how the requirements concerning a service request can be transferred into the service world in order to select the more accurate service compared with the service consumer's requirements, and (iii) to move a step closer to the use of requirements expressed by the service consumer in service requests building.

The formal links between the specifications of the requirements and the elements of the service request is also an important work for the requirements management at runtime: when a service request cannot be satisfied any longer, the service selection system could analyze the problematic requirement(s) and then compute a new service request still satisfying the service consumer's requirements, but which is less preferred by him. It is impossible to keep all the candidate service requests

and all the consumers' requirements expressed if the latter are directly specified within a single service request based on formalisms such as WSDL or WSML.

A. Future Work

The abstract mapping introduced in this paper need a syntactical language in order to specify instances belonging to the concept for each ontology used. Based on the proposed links, future work will focus on the *selection* and on the *integration* of a language both for the CORE ontology and the WSMO ontology. The best choice for the service ontology seems to be WSML [12] despite the possible gaps underlined in the end of §IV-B. It has been designed especially for the WSMO meta-model. Concerning the RE ontology, a whole RE methodology must be chosen among the current possibilities and eventually adapted to our application domain, i.e., the service oriented paradigm. This RE methodology might be built upon Techne [52] which is an abstract requirements modeling language. By *integration of languages* we mean the creation of automatic rules to translate instances of CORE concepts into instances of the corresponding WSMO concepts; in other words, the creation of translation rules between, e.g., the future methodology built upon Techne and the WSML specification.

REFERENCES

[1] L. Liu, E. S. K. Yu, and H. Mei, "Guest Editorial: Special Section on Requirements Engineering for Services - Challenges and Practices," *IEEE Transactions on Services Computing*, vol. 2, no. 4, pp. 318–319, 2009.

[2] M. P. Papazoglou and D. Georgakopoulos, "Service-oriented Computing," *Commun. ACM*, vol. 46, no. 10, pp. 24–28, 2003.

[3] S. A. McIlraith and D. L. Martin, "Bringing Semantics to Web Services," *IEEE Intelligent Systems*, vol. 18, no. 1, pp. 90–93, 2003.

[4] A. van Lamsweerde, *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley, 2009.

[5] B. Nuseibeh and S. M. Easterbrook, "Requirements Engineering: A Roadmap," in *ICSE - Future of SE Track*, 2000, pp. 35–46.

[6] B. H. C. Cheng and J. M. Atlee, "Research Directions in Requirements Engineering," in *Proceedings of the International Conference on Software Engineering (ISCE 2007) and Workshop on the Future of Software Engineering (FOSE 2007)*. ACM, 2007, pp. 285–303.

[7] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-Oriented Computing: a Research Roadmap," *International Journal of Cooperative Information Systems*, vol. 17, no. 2, pp. 223–255, 2008.

[8] W.-T. Tsai, Z. Jin, P. Wang, and B. Wu, "Requirement Engineering in Service-Oriented System Engineering," in *Proceedings of ICEBE 2007, IEEE International Conference on e-Business Engineering and the Workshops SOAIC 2007, SOSE 2007, SOKM 2007*. IEEE Computer Society, 2007, pp. 661–668.

[9] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-Oriented Computing: State of the Art and Research Challenges," *IEEE Computer*, vol. 40, no. 11, pp. 38–45, 2007.

[10] I. J. Jureta, J. Mylopoulos, and S. Faulkner, "A Core Ontology for Requirements," *Applied Ontology*, vol. 4, no. 3-4, pp. 169–244, 2009.

[11] D. Roman, U. Keller, H. Lausen, J. de Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel, "Web Service Modeling Ontology," *Applied Ontology*, vol. 1, no. 1, pp. 77–106, 2005.

[12] J. de Bruijn, H. Lausen, A. Polleres, and D. Fensel, "The Web Service Modeling Language WSML: An Overview," in *The Semantic Web: Research and Applications, 3rd European Semantic Web Conference (ESWC 2006)*, ser. Lecture Notes in Computer Science, vol. 4011. Springer, 2006, pp. 590–604.

[13] P. Zave, "Classification of Research Efforts in Requirements Engineering," *ACM Computing Survey*, vol. 29, no. 4, pp. 315–321, 1997.

[14] A. van Lamsweerde, "Goal-Oriented Requirements Engineering: A Guided Tour," in *Proceedings of the Fifth IEEE International Symposium on Requirements Engineering (RE 2001)*. IEEE Computer Society, 2001, p. 249.

[15] L. Chung and J. C. Sampaio do Prado Leite, "On Non-Functional Requirements in Software Engineering," in *Conceptual Modeling: Foundations and Applications*, ser. Lecture Notes in Computer Science, vol. 5600. Springer, 2009, pp. 363–379.

[16] E. S. K. Yu, "Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering," in *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE 1997)*. IEEE Computer Society, 1997, pp. 226–235.

[17] G. Regev and A. Wegeman, "Where do Goals Come From: the Underlying Principles of Goal-Oriented Requirements Engineering," in *Proceedings of the 13th International Conference on Requirements Engineering (RE 2005)*. IEEE Computer Society, 2005, pp. 353–362.

[18] I. J. Jureta, J. Mylopoulos, and S. Faulkner, "Revisiting the Core Ontology and Problem in Requirements Engineering," in *Proceedings of the 16th IEEE International Requirements Engineering Conference (RE 2008)*. IEEE Computer Society, 2008, pp. 71–80.

[19] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, Eds., *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[20] A. Borgida and L. Serafini, "Distributed Description Logics: Assimilating Information from Peer Sources," *Journal on Data Semantics I*, vol. 2800, pp. 153–184, 2003.

[21] A. Ankolenkar, M. Burstein, J. R. Hobbs, O. Lassila, D. L. Martin, D. McDermott, S. A. McIlraith, S. Narayanan, M. Paolucci, T. R. Payne, and K. Sycara, *OWL-S: Semantic Markup for Web Services 1.1*, D. L. Martin, Ed. DAML Services Coalition, 2004.

[22] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara, "OWL-S: Semantic Markup for Web Services," World Wide Web Consortium (W3C), Tech. Rep., 2004. [Online]. Available: http://www.w3.org/Submission/OWL-S/

[23] D. Fensel, H. Lausen, A. Polleres, J. d. Bruijn, M. Stollberg, D. Roman, and J. Domingue, *Enabling Semantic Web Services: The Web Service Modeling Ontology*. Springer, 2006.

[24] D. Roman, H. Lausen, and U. Keller, "D2v1.2 Web Service Modeling Ontology (WSMO)," The ESSI WSMO working group, Tech. Rep., April 2005. [Online]. Available: http://www.wsmo.org/TR/d2/v1.2/D2v1-2_20050414.pdf

[25] Y. Gurevich, "Evolving algebras 1993: Lipari guide," in *Specification and Validation Methods*. Oxford University Press, 1993, pp. 9–36.

[26] D. Roman, J. Scicluna, C. Feier, M. Stollberg, and D. Fensel, "D14v0.1 Ontology-based Choreography and Orchestration of WSMO Services," The ESSI WSMO working group, Tech. Rep., March 2005. [Online]. Available: http://www.wsmo.org/TR/d14/v0.1/

[27] The WSML working group members, "D16.1v1.0 WSML Language Reference," The ESSI WSML working group, Tech. Rep., 2008. [Online]. Available: http://www.wsmo.org/TR/d16/d16.1/v1.0/

[28] A. Haller, E. Cimpian, A. Mocan, E. Oren, and C. Bussler, "WSMX - A Semantic Service-Oriented Architecture," in *Proceedings of the 2005 IEEE International Conference on Web Services (ICWS 2005)*. IEEE Computer Society, 2005, pp. 321–328.

[29] M. Kerrigan, "Web service selection mechanisms in the Web Service Execution Environment (WSMX)," in *Proceedings of the 2006 ACM Symposium on Applied Computing (SAC 2006)*. ACM, 2006, pp. 1664–1668.

[30] Object Management Group, *Meta Object Facility (MOF) Core Specification Version 2.0*, 2006. [Online]. Available: http://www.omg.org/cgi-bin/doc?formal/2006-01-01

[31] R. Chinnici, J.-J. Moreau, A. Ryman, and S. Weerawarana, "Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language," World Wide Web Consortium (W3C), W3C Recommendation, 26 June 2007. [Online]. Available: http://www.w3.org/TR/wsdl20/

[32] World Wide Web Consortium (W3C), "Simple Object Access Protocol (SOAP)," Tech. Rep., 2003. [Online]. Available: http://www.w3.org/TR/soap

[33] OASIS UDDI Specification TC, "UDDI Spec Technical Committee Draft 3.0.2," OASIS, OASIS Committee Draft, 2004. [Online]. Available: http://uddi.org/pubs/uddi_v3.htm

[34] Q. Yu, X. Liu, A. Bouguettaya, and B. Medjahed, "Deploying and managing Web services: issues, solutions, and directions," *VLDB Journal*, vol. 17, no. 3, pp. 537–572, 2008.

[35] A. van Lamsweerde, "Requirements Engineering in the Year 00: A Research Perspective," in *Proceedings of the 22nd International Conference on Software Engineering (ICSE'2000)*. ACM Press, 2000, pp. 5–19.

[36] A. van Lamsweerde, R. Darimont, and P. Massonet, "Goal-Directed Elaboration of Requirements for a Meeting Scheduler: Problems and Lessons Learnt," in *Proceedings of the Second IEEE International Symposium on Requirements Engineering (RE'95)*. IEEE Computer Society, 1995, pp. 194–203.

[37] G. Agre and I. Dilov, "How to Create a WSMO-Based Semantic Service Without Knowing WSML," in *Web Information Systems Engineering - WISE 2007 International Workshops*, ser. Lecture Notes in Computer Science, vol. 4832. Springer, 2007, pp. 217–235.

[38] H. E. Bouhissi, M. Malki, and D. Bouchiha, "Towards WSMO Ontology Specification From Existing Web Services," in *Proceedings of the 2nd Conférence Internationale sur l'Informatique et ses Applications (CIIA'09)*, ser. CEUR Workshop Proceedings, vol. 547. CEUR-WS.org, 2009.

[39] J. Nern, G. Agre, T. Atanasova, Z. Marinova, A. Micsik, L. Kovcs, J. Saarela, and T. Westkaemper, "INFRAWEBS Semantic Web Service Development on the Base of Knowledge Management Layer," *International Journal on Information Theories & Applications*, vol. 13, pp. 161–168, 2006.

[40] W. N. Robinson, "Monitoring Web Service Requirements," in *Proceedings of the 11th IEEE International Conference on Requirements Engineering (RE 2003)*. IEEE Computer Society, 2003, pp. 65–74.

[41] Q. Wang, J. Shao, F. Deng, Y. Liu, M. Li, J. Han, and H. Mei, "An Online Monitoring Approach for Web Service Requirements," *IEEE Transactions on Services Computing*, vol. 2, no. 4, pp. 338–351, 2009.

[42] Y. Hao, Y. Zhang, and J. Cao, "WSXplorer: Searching for Desired Web Services," in *Proceedings of the 19th International Conference on Advanced Information Systems Engineering (CAiSE 2007)*, ser. Lecture Notes in Computer Science, vol. 4495. Springer, 2007, pp. 173–187.

[43] G. Spanoudakis, A. Zisman, and A. Kozlenkov, "A Service Discovery Framework for Service Centric Systems," in *Proceedings of the 2005 IEEE International Conference on Services Computing (SCC 2005)*. IEEE Computer Society, 2005, pp. 251–259.

[44] K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar, and J. Miller, "METEOR-S WSDI: A Scalable P2P Infrastructure of Registries for Semantic Publication and Discovery of Web services," *Journal of Information Technology and Management*, vol. 6, no. 1, pp. 17–39, 2005.

[45] E. Grandry, E. Dubois, M. Picard, and A. Rifaut, "Managing the Alignment between Business and Software Services Requirements from a Capability Model Perspective," in *Towards a Service-Based Internet: First European Conference, ServiceWave 2008*, ser. Lecture Notes in Computer Science, vol. 5377. Springer, 2008, pp. 171–182.

[46] C. Rolland, R. S. Kaabi, and N. Kraïem, "On ISOA: Intentional Services Oriented Architecture," in *Proceedings of the 19th International Conference on Advanced Information Systems Engineering (CAiSE 2007)*, ser. Lecture Notes in Computer Science, vol. 4495. Springer, 2007, pp. 158–172.

[47] M. Driss, N. Moha, Y. Jamoussi, J.-M. Jézéquel, and H. H. B. Ghézala, "A Requirement-Centric Approach to Web Service Modeling, Discovery, and Selection," in *Proceedings of the 8th International Conference on Service-Oriented Computing (ICSOC 2010)*, ser. Lecture Notes in Computer Science, vol. 6470, 2010, pp. 258–272.

[48] H. Chen and K. He, "A Method for Service-Oriented Personalized Requirements Analysis," *Journal of Software Engineering and Applications*, vol. 4, no. 1, pp. 59–68, 2011.

[49] K. Zachos, N. A. M. Maiden, X. Zhu, and S. Jones, "Discovering Web Services to Specify More Complete System Requirements," in *Proceedings of the 19th International Conference on Advanced Information Systems Engineering (CAiSE 2007)*, ser. Lecture Notes in Computer Science, vol. 4495. Springer, 2007, pp. 142–157.

[50] K. Zachos and N. A. M. Maiden, "Inventing Requirements from Software: An Empirical Investigation with Web Services," in *Proceedings of the 16th IEEE International Requirements Engineering Conference (RE 2008)*. IEEE Computer Society, 2008, pp. 145–154.

[51] R. Studer, S. Grimm, and A. Abecker, Eds., *Semantic Web Services, Concepts, Technologies, and Applications*. Springer, 2007.

[52] I. J. Jureta, A. Borgida, N. A. Ernst, and J. Mylopoulos, "Techne: Towards a New Generation of Requirements Modeling Languages with Goals, Preferences, and Inconsistency Handling," in *Proceedings of the 18th IEEE International Requirements Engineering Conference (RE 2010)*. IEEE Computer Society, 2010, pp. 115–124.