Towards Conceptual Foundations of Requirements Engineering for Services

Verlaine, Bertrand; Jureta, Ivan; Faulkner, Stéphane

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

Link to publication
Citation for published version (HARVARD):

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.
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 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 WSMO 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.e., 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.
C. The Core Ontology in Description Logic

We use the Description Logic (DL) $SLN$ [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 $AL$. It allows us to specify, i.e., the intersection of two concepts ($A \cap B$), the universal restriction ($\forall r.A$: all instances having the relation $r$ with at least one instance of $A$) and the limited existential quantification ($\exists r.T$: it exists instances having the relation $r$). In addition to these constructors, we use for formalizing Core and WSMO the union ($A \cup B$), 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 ($A \subseteq B$: 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 $SLN$ logic.

Line 1 of Table I defines the root concept of the Core ontology. Requirements expressed under OE 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\footnote{The European Semantic 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/} 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)
The WSMO concept ontology mediator, and the definition of concept, relation, 23 service being described, and so on used, the unique identifier of the service, the version of the MOF Meta Object Facility ([24]. Its model layer is composed of four classes:

11 of a service, i.e., its concepts and their relations

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\(^4\):

```xml
Class goal
  hasNonFunctionalProperties type nonFunctionalProperties
  importsOntology type ontology
  usesMediator type ooMediator, ggMediator
  requestsCapability type capability multiplicity = single-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.
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, Post-condition, 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 III

**WSMO written in description logic**

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>16:</td>
<td>( \text{GOAL} \equiv \text{Non functional property} \sqcap \text{Capability} \sqcap \text{Interface} )</td>
</tr>
<tr>
<td>17:</td>
<td>( \text{GOAL} \sqsubseteq \text{Service user needs} )</td>
</tr>
<tr>
<td>18:</td>
<td>( \text{GOAL} \sqsubseteq 1 \text{composed-by} \cdot \text{Capability} )</td>
</tr>
<tr>
<td>19:</td>
<td>( \text{compose} \equiv \text{composed-by} \cdot )</td>
</tr>
<tr>
<td>20:</td>
<td>( \text{composed-by} \equiv \text{compose} \cdot )</td>
</tr>
<tr>
<td>21:</td>
<td>( \text{Non functional property} \equiv \text{Behavioural service property} \sqcap \text{Non-restrictive service property} )</td>
</tr>
<tr>
<td>22:</td>
<td>( \text{Capability} \equiv \text{Non functional property} \sqcap \text{Shared variables} \sqcap \text{Precondition} \sqcap )</td>
</tr>
<tr>
<td>23:</td>
<td>( \text{Axiom} \sqsubseteq \text{Precondition} \sqcap \text{Assumption} \sqcap \text{Postcondition} \sqcap \text{Effect} )</td>
</tr>
<tr>
<td>24:</td>
<td>( \text{Axiom} \equiv \text{Logical expression} \sqcap \text{Non-restrictive axiom property} )</td>
</tr>
<tr>
<td>25:</td>
<td>( \text{Interface} \equiv \text{Non functional property} \sqcap \text{Choreography} \sqcap \text{Orchestration} )</td>
</tr>
<tr>
<td>26:</td>
<td>( \text{Choreography} \equiv \text{State machine} )</td>
</tr>
<tr>
<td>27:</td>
<td>( \text{State machine} \equiv \text{State} \sqcap \text{Transition rule} )</td>
</tr>
<tr>
<td>28:</td>
<td>( \text{State machine} \sqsupseteq 2 \text{sequence} \cdot \text{State} )</td>
</tr>
<tr>
<td>29:</td>
<td>( \text{sequence} \equiv \text{sequenced-by} \cdot )</td>
</tr>
<tr>
<td>30:</td>
<td>( \text{sequenced-by} \equiv \text{sequence} \cdot )</td>
</tr>
</tbody>
</table>
The service consumer can express a desire, i.e., a goal, concerning other types of system-to-be, which can be a software or not. 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 fulfill 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 21st of July.

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

<table>
<thead>
<tr>
<th></th>
<th>Behavioural service property</th>
<th>Precondition</th>
<th>Assumption</th>
<th>Postcondition</th>
<th>Effect</th>
<th>Choreography</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>V</td>
<td>X</td>
<td>X</td>
<td>V</td>
<td>V</td>
<td>V</td>
</tr>
<tr>
<td>Plan</td>
<td>X</td>
<td>V</td>
<td>V</td>
<td>X</td>
<td>X</td>
<td>V</td>
</tr>
<tr>
<td>Domain assumption</td>
<td>X</td>
<td>V</td>
<td>V</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Evaluation</td>
<td>V</td>
<td>V</td>
<td>V</td>
<td>V</td>
<td>V</td>
<td>V</td>
</tr>
</tbody>
</table>

\(^3\)In Table IV, the sign ”\(V\)” means that the WSMO class is mapped with the corresponding CORE class. Otherwise, the sign ”\(X\)” 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, $A \rightarrow B$, 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
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 assumptions. 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.

- **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\textsuperscript{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 al. [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, on 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 WSMO [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 WSMO specification.

REFERENCES

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

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

H. E. Bouhissi, M. Malki, and D. Bouchiha, “Towards WSMO Ontology

J. Nern, G. Agre, T. Atanasova, Z. Marinova, A. Micsik, L. Kovcs,


