Towards Conceptual Foundations for Service-oriented Requirements Engineering: Bridging Requirements and Services Ontologies
Verlaine, Bertrand; Dubois, Yves; Jureta, Ivan; Faulkner, Stéphane

Published in:
IET Software Journal

Publication date:
2012

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.

Download date: 06. Nov. 2018
Towards Conceptual Foundations for
Service-oriented Requirements Engineering:
Bridging Requirements and Services Ontologies*

Bertrand Verlaine, Yves Dubois, Ivan J. Jureta, Stéphane Faulkner
PReCISE Research Center
University of Namur
Rempart de la Vierge, 8
B-5000 Namur, Belgium
{bverlain, ydubois, ijureta, sfaulkne}@fundp.ac.be
February 21, 2011

Abstract

The engineering of a service-oriented system requires the specification
of functions that Web Services (ws) should provide, before ws are
built or selected. Written in a service description language, the service
specification instantiates concepts different than those used for Requirement
Engineering (Re): the former speaks in terms of operations, metrics and
bindings, while the latter manipulates, goals, evaluations and domain
assumptions. It is, however, clear that functions expected of ws to select
or build will be relevant to the stakeholders if they satisfy the stakeholders’
requirements. As a result, there is a gap between the two specifications
which must be bridged in order to ensure that the ws system is adequate
w.r.t. requirements. This paper proposes mappings between the concepts

*An initial version of this work was presented at the First International Workshop on the
Web and Requirements Engineering (WERE 2010, Sydney, Australia) [1].
of requirements ontology and those of service taxonomy induced by the WS LD and the W SLA languages. A working prototype is presented that implements the mappings and is used to translate the instances of RE concepts into instances of WS LD and W SLA concepts. The mappings and the prototype facilitate the engineering of WS systems, as fragments of WS descriptions can be generated from requirements as a first specification of a service request.

Keywords: Requirements Engineering for Service-oriented Systems, Ontology Mapping

1 Introduction

Engineering and managing the operation of increasingly complex information systems (18) is a key challenge in computing (e.g., [2, 3]). It is now widely acknowledged that degrees of automation needed in response cannot be achieved without distributed, interoperable, and modular systems. Among the various, often overlapping approaches to building such systems, service-orientation stands out in terms of its reliance on the World Wide Web infrastructure, availability of standards for describing and enabling interaction between services, attention to interoperability and uptake in industry.

A service, the central concept in Service-Oriented Computing (SOC), 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 [4, 3]. A Web Service (WS) is a service that relies on standards such as SOAP [5], WSDL [6] or UDDI [7] to enable its use, and that can be invoked over the World Wide Web. A WS is thus the technical implementation of the service concept. WSs are offered by service providers that ensure service implementations, advertise service descriptions, and provide related technical and business support. Service consumers have to find the appropriate WS among the WSs available to satisfy their requirements.

The engineering of service-oriented systems involves many issues treated in the literature –among them, infrastructure for services (e.g., [5, 7, 8]), descriptions
of services’ interfaces, capabilities, behaviours, and qualities (e.g., [6, 9, 10, 11]), service discovery (e.g., [12]), service composition (e.g., [13, 14, 15, 16]), and ontologies and ontology languages (e.g., [11, 17, 18, 19, 20, 21]). A considerable part of the research focuses on service provision problems, i.e., “the current SOA [service-oriented architecture] is producer centric” [22]. In contrast, this paper focuses on the service consumer side.

**Problem statement.** A service-oriented system will be satisfactory only if it satisfies the requirements of the system’s stakeholders. The Requirements Engineering (RE) for such systems is a promising area of inquiry that already attacked some of the key issues. RE is usually defined as the process by which the stakeholders of a system-to-be are identified, their requirements elicited in order to model the specifications of the system-to-be, which should then be implemented [23, 24, 25]. One pressing concern, which has received less attention and is the focus of this paper, is: *How to bridge the gap between a specification of requirements and WS descriptions?* A description of a WS specifies the functions that the WS can provide. It is based on such a specification that WSs are developed, or sought among available ones. Specialized languages have been designed for the description of WSs using concepts of, e.g., operation and binding, tailored to the WS description. On the other hand, requirements that these services ought to satisfy are classified according to ontologies tailored to RE, which rely on concepts such as goal, task, and domain assumption. While clearly the functions expected of WSs will be relevant to the system if and only if they satisfy the stakeholders’ requirements, the differences in the conceptualizations that underlie WS descriptions and RE specifications make it unclear how exactly to translate the content of specific requirements into WS descriptions, hence the gap.

**Contributions.** This paper is a first step towards addressing the gap between RE specifications and WS descriptions by mapping the concepts of the Core Ontology for REquirements (CORE) [26] to the concepts of the Web Service Description Language (WSDL) [6] proposed by the World Wide Web Consortium (W3C) and the IBM’s Web Service Level Agreement (WSLA) formalism [27]. Two
contributions are made. Firstly, the mappings between the two representations of requirements are presented both informally and in the Distributed Description Logic formalism, and the rationale for the mappings is discussed. Once the mappings are available and a specification of requirements is given, it is possible to facilitate the writing of WS descriptions in WSLA/WSDL by translating the specification of the requirements captured by propositions into fragments of WSLA/WSDL descriptions. The second contribution is the working prototype tool that implements the mappings, allowing thereby the translation of the instances of RE concepts into instances of WSLA/WSDL concepts. The mappings and the prototype facilitate the engineering of WS systems, as fragments of service descriptions can be generated from requirements.

Organization. The remaining parts of this paper are structured as follows. First, we discuss our technological choices and briefly present the selected ontologies and technologies on which our mappings are built (§2). Then, the formalization of the two conceptualizations is presented (§3), followed by the mapping between them (§4). This mapping allows us to build a tool which should help requirements engineers to specify the service consumers’ requirements and translate them into initial WS descriptions (§5). Finally, we briefly relate comparable research efforts (§6) before drawing up conclusions and summarizing relevant directions for future work (§7).

2 Baseline

To bridge the gap between the requirements expressed by the service consumer and the specifications of service requests, we use a requirements ontology (§2.2) and we build a service taxonomy (§2.3). Below, we discuss our choices of the ontologies (§2.1), namely CORE as the RE ontology, while we work with the WSDL and WSLA languages at the service level.
2.1 Choices of Ontologies

An ontology is a set of concepts and relations, where a concept defines the properties that every member of its class should have, and a relation defines joint properties of a set of members, each of which participants in the same or different class. An ontology is thus an explicit specification of a particular conceptualization shared by a community [28]. We ought to distinguish top-level ontologies which “describe very general concepts [regardless of] a particular problem or domain” [29]. They are shared by large communities of users. At the second level, there are the domain ontologies and the task ontologies respectively used for the vocabulary description of a generic domain, and the description of a generic task/activity. A domain ontology or a task ontology specialize the terms of the top-level ontologies in, respectively, a domain centric way or in a task centric way. At the lower level, there are application ontologies which describe “concepts depending on both a particular domain and task” [29]. Thus, these low-level ontologies specialize both a domain ontology and a task ontology.

On the RE side, our choice is CORE (cf., §2.1.1). This ontology specifies the domain of requirements and their relations that stakeholders may express concerning a system-to-be.

On the service side, we decide to build our own service taxonomy (cf., §2.1.2). A taxonomy is a structured description of objects into classes related to a specific domain. In the scope of our work, two relevant differences between an ontologies and a taxonomy must be underlined. First, an ontology must be founded upon a kind of formalism. In contrast, this is not required to build a taxonomy in which definitions in natural language can be used. Secondly, while particular relations between the concepts can be specified in an ontology, only two relations can be used in taxonomies. Those two relations are the subsumption relation (is-a) –the class A has the relation is-a with the class B means that A is a subclass of B–, and the membership relation (is-of) –the object c has the relation is-of with the class C means that c can be classified in C.
2.1.1 A conceptualization for requirements

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., [23, 25, 30, 31, 32, 33, 34]), CORE offers a simple set of essential concepts for RE, by covering the main notions that were previously identified and used, and by defining them within a single ontology.

2.1.2 A conceptualization for service

There are two significant views on the service notion in the SOC: a syntactical view and a semantic view. This distinction comes mainly as a response to the service interoperability problem [35], which is one of the most significant issues in SOC. The first view is mainly supported by WS technologies and Web technologies such as WSDL [6], Universal Description Discovery and Integration (UDDI) [7], Hypertext Transfer Protocol (HTTP) [36], SOAP [5], WS-MA [37] or Web Service Agreement (WS-Agreement) [38]. Most of the WS technologies are based on the eXtensible Markup Language (XML) [39] which structures the information, and describes it to allow an informal interpretation. The second view on a service conceptualization is based on technologies using logic languages and domain/task ontologies to describe the service capabilities, e.g., Web Service Modeling Language (WSML) [40], QoSOnt [41], OWL-S (previously named DAML-S) [42], WSDL-S [43] or SAWSDL [44]. Their common objective is to make the informational content amenable to processing by a computer.

In this work, we choose the syntactic view on the service-oriented paradigm\(^1\). Therefore, all conceptualizations built within the semantic view are excluded. Seeing that all service ontologies or taxonomies, e.g., WSMO, OWL-S and the Semantic Web Services Ontology (SWSO) [47], fit into the semantic view on the service-oriented paradigm, we build our own WS taxonomy. This taxonomy has to be wide enough to cover functional and non-functional characteristics of WSS. Given that there is not any syntactic technology, which satisfactorily

\(^1\)We have discussed elsewhere [45, 46] the mapping based on a semantic view on the service-oriented paradigm.
covers all those characteristics, we need at least two technologies, one for the functional features and one for the non-functional features of WSS. There is one attempt—the Web Service Offerings Language (WSOL) [48]—to encompass whole WS characteristics. However, WSOL proves to be inefficient concerning this targeted objective: this technology still needs WSDL to work and only allows to specify some of the non-functional characteristics compared with, e.g., WS-Agreement or WSLA.

With regard to the functional characteristics, an Interface Definition Language (IDL) is needed [49]. An IDL gives a framework to specify a machine-readable interface for computational components, such as WSS, independently of the coding languages and underlying technologies used. The WSDL language, which has the status of recommendation by the World Wide Web Consortium (W3C), is an appropriate IDL. The WS community uses and/or advises this language for the engineering of Service-Oriented Architectures (SOA) [35, 3, 50, 51, 52, 53]. This technology is also applied in the computing industry (e.g., [54, 55, 56, 57, 58]).

In relation to non-functional characteristics, and thereby Quality of Service (QoS), the main technologies proposed in the literature are WSLA [37], WS-Agreement [38], SLAmp [59, 60] and Universal Service Description Language (USDL) [61]. SLAmp, which can describe the two involved parties and their responsibilities during the WS use, divides Service Level Agreements (SLAs) into horizontal contracts (e.g., between two equal parties) and vertical contracts (e.g., between entities in different layers). This language focuses on WS-based Internet services such as Application Service Provision, Internet Service Provision and Storage Service Provision. Moreover, SLAmp does not allow to specify financial terms associated with the SLA. USDL can be used to specify SLAs for services—it is thus not only focused on WSS—which must be associated to another language specific to the service-oriented paradigm. The authors chose WS-Agreement. Clearly, SLAmp and USDL do not answer to our needs. Concerning WS-Agreement, this technology has one drawback in comparison with WSLA: it does not allow to describe obligations of parties as WSLA allows it. The obligation is an explicit duty that a party has to achieve in regard to the service level objectives (SLOS)
specified in the SLA document. Furthermore, WSLA is expressively built to complete WSDL, our first choice on which we base our service taxonomy.

2.2 Overview of the Core Ontology for REquirements

The root concept of the core ontology is Communicated information\(^2\), specialized as follows [62]:

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

A basic idea in core is that requirements are communicated by the stakeholders to the requirements engineer, so that the latter classifies requirements based on what was communicated and how it was communicated. The Communicated information concept is a catchall one; its instances are propositions communicated by the stakeholders. Once an instance of that concept 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 concept. 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 concept. 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

\(^2\)A core concept is written Concept and starting with an uppercase letter, while an instance thereof starts with a lowercase letter instance.
as instances of the Plan concept. 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 concept of Evaluation. Propositions belonging to this concept convey evaluations arising out of emotions of the stakeholders.

CORE distinguishes three kinds of goals. The Functional goal concept refers to a desired condition for which its satisfaction is verifiable, i.e., the comparison scale is shared among the stakeholders and the requirements engineer(s), and is binary, i.e., the functional goal is either satisfied or not. A quality constraint defines the desired value of a non-binary measurable property of the system-to-be (e.g., how many seconds it takes to answer a query). As functional goals and quality constraints are not necessarily known at the very start of the RE process, the Softgoal concept is instantiated to capture requirements which refer to vague properties of the system-to-be (e.g., a “fast” answer to the queries). Same specialization applies to the Domain assumption concept, which has its functional variant—a functional domain assumption 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, Evaluation can qualify individual requirements through the Individual evaluation concept, or compares goals, domain assumptions, and/or plans through the Comparative evaluation concept.

2.3 Overview of the Web Service taxonomy

IBM’s WSLA technology [27] intends to specify contracts, called SLA’s. They state constraints on QoS properties of WSs. While WSLA focuses on the QoS levels of WSs, WSDL [6], the second formalism chosen, allows to specify the functional characteristics of WSs.³

³Note WS-DL allows managing some possible use failures by the specification of fault conditions and repair actions, which certainly is relevant given that WS oriented systems are often distributed and given potential Web server breakdowns. We leave out this aspect of WSDL for future work (cf., §7.1).
The WSLA concepts are Party, Service definition, Metric and Obligations. The WSDL concepts are Operation, Binding and Service. We retain the following four of these seven concepts:

1. **Metric** identifies an observable QoS property of a WS, and indicates its measurement directive(s), i.e., it specifies how that QoS property can be accessed and/or computed [37, 27].

2. The **Obligations** concept defines the guaranteed QoS level of the WS identified in the service definition as well as constraints imposed on the metrics and triggered actions [27, 37]. The two subconcepts of the Obligations are:
   - (a) **Service level objective** which defines the different QoS levels regarding the observable characteristics –described in a metric – of the WS, and
   - (b) **Action guarantee** which groups promises of the signatory parties and/or of third parties concerning the achievement of an action when a determined precondition occurs.

3. **Operation** defines the interaction between the service provider and the other parties involved in the interaction, as a sequence of input and output messages [63, 6].

4. **Binding** specifies concrete message format and transmission protocol details concerning the WS use [63].

**Party**, **Service definition**, and **Service** are not retained as concepts of our WSLA/WSDL taxonomy for the following reasons:

- Instances of **Party** identify the WS provider, the WS consumer and possible third parties, which may be stakeholders expressing requirements w.r.t. the service they would like to use. As the definition of the requirements problem abstracts from these identifiers, we do not carry at the service level the information on which stakeholder gave which requirement.

---

4 An WSLA or an WSDL concept is written as *Concept* and an instance of one of those concepts is depicted as *instance*.

5 Note the precondition can simply be *always*. 

10
• A *Service definition* instance is not directly evaluated by the WS consumer. Its purpose is to link a WSLA specification of a WS to a document which describes the functional characteristics of that WS. As we use WSDL, the WS consumer—i.e., the stakeholder—can directly evaluate the functional characteristics through the WSDL document.

• *Service* is not relevant in the present discussion, as the actual Web location of the WS is unimportant. Only its presence or absence is crucial. The possible unresponsiveness of the WS could be evaluated through other selected concepts, e.g., an *obligations*.

3 Formalization of CORE and WSLA/WSDL

In order to formalize the bridging of CORE with the WSLA/WSDL taxonomy, we use the description logic $\mathcal{SIN}$ [64] to rewrite each conceptualization. This rewriting allows us to connect WSDL to WSLA (to get what we refer to as WSLA/WSDL taxonomy), and then CORE to WSLA/WSDL (see §4.3).

3.1 The CORE ontology in description logic

Table 1 is based on the definitions and axioms of the CORE ontology given in §2.2. Line 1 defines the root concept of CORE. Requirements expressed during the RE process are classified into the four main classes of CORE, i.e., *Goal*, *Plan*, *Domain assumption* and *Evaluation*, and finally in the leaves of CORE, i.e., *Quality constraint*, *Soft domain assumption*, *Comparative evaluation*, and so on (see Lines 6, 11 and 14). 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: given their inexplicit nature, they need to be replaced by more precise requirements. Just as, say, imprecise *goals* are refined, so are *softgoals* and *soft domain assumptions* approximated [26, 62], whereby their approximation involves the identification of *quality constraints* and *quality domain assumptions*, while *comparative evaluations* may indicate how alternative *quality constraints* or *quality domain assumptions* may be rated in terms
of relative desirability. Lines 10 and 13 reflect this in the formalized ontology.

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Communicated information ≡ Goal ⊔ Plan ⊔ Domain assumption ⊔ Evaluation</td>
</tr>
<tr>
<td>2</td>
<td>refine ≡ refined-by⁻</td>
</tr>
<tr>
<td>3</td>
<td>refined-by ≡ refine⁻</td>
</tr>
<tr>
<td>4</td>
<td>⊤ ⊑ ∀ refine. Communicated information</td>
</tr>
<tr>
<td>5</td>
<td>∀ refine. Goal ≡ Functional goal ⊔ Quality constraint ⊔ Softgoal</td>
</tr>
<tr>
<td>6</td>
<td>⊥ ⊑ ∀ refine. Functional goal ⊔ Quality constraint ⊔ Softgoal</td>
</tr>
<tr>
<td>7</td>
<td>approximate ≡ approximated-by⁻</td>
</tr>
<tr>
<td>8</td>
<td>approximated-by ≡ approximate⁻</td>
</tr>
<tr>
<td>9</td>
<td>Softgoal ⊑ ∃ approximate. Quality constraint</td>
</tr>
<tr>
<td>10</td>
<td>∀ refine. Domain assumption ≡ Functional domain assumption ⊔ Quality domain assumption ⊔ Soft domain assumption</td>
</tr>
<tr>
<td>11</td>
<td>⊥ ⊑ ∀ refine. Functional domain assumption ⊔ Quality domain assumption ⊔ Soft domain assumption</td>
</tr>
<tr>
<td>12</td>
<td>Soft domain assumption ⊑ ∃ approximate. Quality domain assumption</td>
</tr>
<tr>
<td>13</td>
<td>∀ refine. Evaluation ≡ Comparative evaluation ⊔ Individual evaluation</td>
</tr>
<tr>
<td>14</td>
<td>⊥ ⊑ Comparative evaluation ⊔ Individual evaluation</td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

### 3.2 The WSLA/WSDL taxonomy in description logic

Table 2 is based on publications about the WSLA formalism [27, 37] and on the W3C recommendations concerning WSDL 2.0 [6, 63, 65]. In Tables 2 and 4, the prefixes “WSLA;” and “WSDL;” indicate that the concept respectively belongs to WSLA or to WSDL. In Table 4, the prefix “CORE;” indicates that the concept belongs to the core ontology. Line 17 (WSLA) states the use of the WSLA specification as a proposal or an agreement. The latter is the primary purpose of WSLA. A proposal could be suggested either by a WS consumer or a WS provider. Requirements concerning non-functional WS properties are specified via WSLA. “COMMITMENT”, used in Lines 18, 21 and 41, refers to a promise to achieve (conditionally or not) a predetermined task. “SLA PARAMETER” is an observable characteristic used to evaluate the QoS of the WS as well as their measurement process (Lines 34 and 37). Line 36 uses Distributed Description Logic (DDL) [66] in order to bridge WSLA with WSDL: in this context, the sign ⊑ means that the “WSLA:OPERATION” concept subsumes the “WSDL:OPERATION” concept.
Line 44 has the same purpose as Line 17, but for the WSDL-oriented part of the formalized taxonomy. Line 54 covers the **Operation** concept: by ordering the messages exchanged between the WS provider and the WS consumer, it organizes the data flow. Though this data exchange flow, the actual service provided by the WS is structured. It enables to know what is the functionality of the service provided.

4 Mapping of CORE with WSLA/WSDL

We introduce a simple but comprehensive case study (§4.1) first below. It will be used to illustrate the mappings developed later (§4.2–§4.3) to relate the requirements expressed as natural language statements, and the corresponding instances of the service taxonomy concepts specified in the WSDL and the WSLA formalisms.

4.1 A scenario: the trucking company

An entrepreneur owns an express transport company and would like to optimize the routes taken by his trucks. Orders and clients data are centralized in his existing **IS** where the routes of each truck are calculated depending on urgent/deleted orders, truck breakdowns, delays, traffic jams, and so on. He has equipped all his trucks with a navigation system based on both the GPS and the UMTS technologies. The GPS device allows to locate the truck and to help the driver in finding the appropriate route, while the UMTS technology allows his **IS** to exchange data with the system embedded in the trucks, which includes the GPS device. The company owner would like that an **IS** sends the data needed in real time to the trucks when the previous job is ending. To avoid wasting time, the device can directly find the way with the coordinates (longitude and latitude) of the client. However, his current **IS** only stores the postal addresses of the delivery locations given by the clients when they order a transport of goods. In this way, the software engineer in charge of this improvement would like to use a service available on the Web, i.e., a WS. The main functionality of this
ws is to provide the coordinates, i.e., the longitude and the latitude, when it receives a postal address.

Requirements related to this case study are refined and specified throughout the next sections (§4.2–§4.3).
4.2 Bridging the service concepts with the four main CORE classes

The first step to achieve is the classification of the WSLA/WSDL concepts into one of the four main classes of CORE, i.e., in Goal, Plan, Domain assumption and/or Evaluation. Depending on how the consumer expressed the requirements, we categorize them in the relevant CORE concept. Then, we verify if the WSLA or the WSDL specification allows the representation of what the requirement conveys. Otherwise, some requirements could be lost during the mapping (cf., Requirement 9).

Table 3, based on the definitions of the CORE concepts and of the WSLA/WSDL concepts, illustrates this classification; explanations and illustrative requirements based on the case study are given afterwards.\(^6\)

Table 3: Classification of WSLA and WSDL concepts into the first four CORE concepts. The sign $\checkmark$ means that the WSLA or WSDL concept is mapped with the corresponding CORE concept. Otherwise, the sign $\times$ is used.

<table>
<thead>
<tr>
<th>WSLA concept</th>
<th>WSDL concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metric</td>
<td>Obligations</td>
</tr>
<tr>
<td>Goal</td>
<td>$\checkmark$</td>
</tr>
<tr>
<td>Plan</td>
<td>$\checkmark$</td>
</tr>
<tr>
<td>Domain assumption</td>
<td>$\checkmark$</td>
</tr>
<tr>
<td>Evaluation</td>
<td>$\times$</td>
</tr>
</tbody>
</table>

A goal captures conditions not yet satisfied that the service consumer desires to see become true in the future [62]. Requirements 1, 2 and 3 are examples of goals based on the developed scenario.

Requirement 1. goal: The owner wants that the average availability of the WS

\(^6\)Complete examples showing the mapping of requirements to one concept of the service taxonomy is given later (§4.3), where a complete mapping is developed.
Requirement 2. goal: The availability of the service must be high.

Requirement 3. goal: The service has to translate a postal address into coordinates.

Goal is mapped with the four WSLA/WSDL concepts. The consumer can express her desire about the presence or absence of a particular observable property, i.e., a metric, which can be included in the future electronic agreement (e.g., Requirement 1). The ws consumer can also express her desire (i) to set the value of a service level objective to a specific number (e.g., Requirement 2 once approximated), and/or (ii) that a party involved in the future agreement achieves a particular action specified via an action guarantee. Those two kinds of desires can be specified in an WSLA proposal as obligations. Concerning the Operation and Binding concepts, the service consumer can respectively indicate her desire about a precise pattern of exchanged messages with particular input and output (e.g., Requirement 3), and/or her desire about a particular message format and a specific transmission protocol. These two requirements can be specified inside an operation –where the important pieces of information for the ws consumer is the output in which he sends his core data, and the input in which he receives the relevant data for his business activity– or a binding.

A plan catches intentions that the service consumer intends to perform. Requirement 4 is an example of a plan.

Requirement 4. plan: The is will communicate based on the SOAP-over-HTTP middleware.

The Plan concept is also mapped with all WSLA/WSDL concepts. The ws consumer can express her intention to perform the measurements of QoS properties via a metric and then deliver the results to other parties. The ws consumer can aim at performing an action guarantee, instance of Obligation. The ws consumer can also promise to send predetermined messages, which are specified inside an operation, or to use particular message formats and/or communication protocols which can be specified through a binding (e.g., Requirement 4).
A domain assumption indicates that its content is believed true by the service consumer, or that its content is made true by the service consumer’s speech act as illustrated by Requirement 5.

**Requirement 5. domain assumption:** the truck company owner intends to compute the average response time of the service use.

The Domain assumption concept is only mapped with Metric: a ws consumer can suggest a description of an observable parameter that she believes true regardless of the actual state of affairs. She also has the capacity to structure and to organize herself the measurements of some observable parameters (e.g., Requirement 5). On the other hand, Domain assumption is not mapped with Obligations, Operation and Binding respectively because (i) action guarantees can only be promised or desired by a party and service level objectives result from a negotiation so that a ws consumer is not expected to have beliefs about them, and she cannot make them true alone, (ii) it seems inappropriate to assume that a ws consumer would believe in particular messages sent by the ws provider without any information about them neither about the (future) ws provider and she cannot make the messages exchange pattern true alone, and (iii) a ws consumer dealing with the communication protocol or the message format is expected to have some basic knowledge about those kinds of technologies, and she cannot make them true alone; otherwise, she is expected not to worry about the way messages are formatted and sent.

An evaluation captures the preference, or the appraisal, of the ws consumer about a single condition (e.g., Requirements 6 and 7), or between conditions that may hold (e.g., Requirements 8 and 9).

**Requirement 6. evaluation:** A response time of 600ms is appraised.

**Requirement 7. evaluation:** A response time of 400ms is appraised.

**Requirement 8. evaluation:** A response time of 400ms is preferred to a response time of 600ms.

**Requirement 9. evaluation:** The use of the middleware SOAP-over-HTTP is
preferred to the middleware SOAP-over-JMS.

During the re process, a ws consumer can express appraisals or preferences of/between goals, domain assumptions and plans, i.e., the conditions evaluated by the service consumers. Unfortunately, only appraisals and preferences about obligations can be specified through the WSLA/WSDL languages (e.g., Requirement 7, 6 and 8). This lack of expressiveness of the WSDL and WSLA languages compared to CORE leads to possible gaps: some evaluations could be lost during their translation to the WSLA/WSDL taxonomy. For example, Requirement 9 cannot be specified with the WSLA and/or WSDL languages, although it can be expressed by the truck owner, and more generally by any ws consumer. Given the scope of this paper, we let the discussion of this issue for future work.

4.3 The mappings between CORE and WSLA/WSDL

In Table 4, we use DDL [66] to formalize the mapping between CORE and the WSLA/WSDL taxonomy. In the mappings, concepts are prefixed by the name of the taxonomy they belong to. The sign $\equiv \rightarrow$ means that the mapping is complete: each instance of the CORE concept has a corresponding instance in the WSLA and/or WSDL concepts. The sign $\supseteq \rightarrow$ indicates that an evaluation can be lost because the scope of CORE is wider than the scope of WSLA/WSDL (see §4.2). We refine the mapping by comparing the definition of the subclasses of the four main CORE concepts with the WSLA/WSDL concepts.

Table 4: The mapping between CORE and the WSLA/WSDL taxonomy formalized with DDL

<table>
<thead>
<tr>
<th>Table 4: The mapping between CORE and the WSLA/WSDL taxonomy formalized with DDL</th>
</tr>
</thead>
<tbody>
<tr>
<td>59: CORE:Functional goal $\equiv \rightarrow$ WSLA:Metric $\sqcup$ WSLA:Action guarantee $\sqcup$ WSDL:Operation</td>
</tr>
<tr>
<td>60: CORE:Quality constraint $\equiv \rightarrow$ WSLA:Service level objective $\sqcup$ WSDL:Binding</td>
</tr>
<tr>
<td>61: CORE:Plan $\equiv \rightarrow$ WSLA:Metric $\sqcup$ WSLA:Action guarantee $\sqcup$ WSDL:Operation $\sqcup$ WSDL:Binding</td>
</tr>
<tr>
<td>62: CORE:Functional domain assumption $\equiv \rightarrow$ WSLA:Metric</td>
</tr>
<tr>
<td>63: CORE:Individual evaluation $\supseteq \rightarrow$ WSLA:Obligations</td>
</tr>
<tr>
<td>64: CORE:Comparative evaluation $\supseteq \rightarrow$ WSLA:Obligations</td>
</tr>
</tbody>
</table>
Table 3 indicates that Goal is bridged to all WSLA/WSDL concepts. Lines 59 and 60 from Table 4 specialize it.

Line 59: Functional goal is linked to Metric, Action guarantee and Operation. A metric specifies how the measurement of a QoS property is achieved. The ws consumer can desire the presence or absence of a specific metric. This desire is not the representation of a quality, i.e., its evaluation is binary. An action guarantee or an operation are the representation of a process to perform, and they are not the representation of a quality. Requirements 10 (refined from Requirement 1) and 11 (refined from Requirement 3) are functional goals. They respectively correspond to a metric (see Specification 1) and an operation (see Specification 2).

Requirement 10. functional goal: The owner of the truck company wants that a third company, EvalCompany, measures the average availability rate of the service.

Specification 1.

<ServiceDefinition>
  <Operation>
    <SLAParameter name="AvgAvailability" type="float" unit="percent"/>
    <Metric>AverageAvailabilityMetric</Metric>
  </SLAParameter>
  <Metric name="AvgAvailabilityMetric" type="float" unit="percent"/>
  <Source>EvalCompany</Source>
  <MeasurementDirective xsi:type="Availability" resultType="float">
    <MeasurementURI>http://www.eval.com/availability</MeasurementURI>
  </MeasurementDirective>
  <Metric>
  </Metric>
</Operation>
</ServiceDefinition>

Requirement 11. functional goal: The service must return the geographic coordinates –longitude and latitude– when it receives a postal address.

Specification 2. "AddressTransmissionType" and "CoordinatesTransmissionType" are defined in Appendix B.

<inteface>
  <operation name="CoordinatesTranslator"
    pattern="http://www.w3.org/ns/wsdl/in-out"
Line 60: **Quality constraint** is linked to Service level objective and Binding. Seeing that the observable parameters are described into a metric, the Service level objective's quality space is common to the parties. The descriptions of the communication protocol and of the message format are two qualities of, respectively, the communication process and of the structure of the data exchanged. Their respective quality spaces are shared among the parties. They can easily notice the use of one or another protocol/data structure. Requirement 12 refines Requirement 2. It corresponds to a service level objective which is specified in Specification 3. Note Requirement 2 is actually a softgoal; it is thus approximated by Requirement 12 in which the measurement scale is shared among the involved parties.

**Requirement 12. quality constraint:** The average availability rate of the service should be at least 97%.

**Specification 3.**

```xml
<Obligations>
  <ServiceLevelObjective name="Availability">
    <Obliged>Provider</Obliged>
    <Validity>...</Validity>
    <Expression>
      <Predicate xsi:type="Greater">
        <SLAParameter>Availiability</SLAParameter>
        <Value>0.97</Value>
      </Predicate>
    </Expression>
    <EvaluationEvent>NewValue</EvaluationEvent>
  </ServiceLevelObjective>
</Obligations>
```

Line 61 does not add any information compared with Table 3 because Plan has not subclasses in the CORE ontology. Requirement 13 refines Requirement 4;
its specification captured inside a binding is proposed in Specification 4.

**Requirement 13. plan:** The IS will communicate based on the SOAP-over-HTTP middleware.

**Specification 4.**

```
<binding name="SOAPBinding" interface="tns:InterfaceName"
    type="http://www.w3.org/ns/wsdl/soap"
    wssoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/">
    <operation ref="tns:CoordinatesTranslator"
        wssoap:mep="http://www.w3.org/2003/05/soap/mep/soap-response"/>
</binding>
```

Line 62: For the same reason as the refinement of the Goal concept –i.e., a metric is not the representation of a quality–, Functional domain assumption is mapped to Metric. Requirement 5 is refined by Requirement 14; the latter is specified in Specification 5.

**Requirement 14. functional domain assumption:** The truck company owner intends to compute himself –thanks to its own IS– the average response time based on the 50 last service uses.

**Specification 5.**

```
<ServiceDefinition>
    <Operation>
        <Metric name="AverageResponseTime" type="float" unit="milliseconds">
            <Source>Customer</Source>
            <Function xsi:type="Divide" resultType="float">
                <Operand>
                    <Metric>SumResponseTime</Metric>
                </Operand>
                <Operand>
                    <Metric>Transactions</Metric>
                </Operand>
            </Function>
        </Metric>
        <Metric name="Transactions" type="Q" unit="transactions">
            <Source>Customer</Source>
            <Function xsi:type="QConstructor" resultType="Q">
                <Metric>SumTransactions</Metric>
                <Window>50</Window>
            </Function>
        </Metric>
    </Operation>
</ServiceDefinition>
```
Note there is no mapping link between the Quality domain assumption concept and an WSLA/WSDL concept. Since “ [...] domain assumptions concern what is true [in the future IS and its environment]” [26], we expected to have only a few mapping links for this class. Our application domain –the ws use process and its environment– is specific because many characteristics are negotiable between the involved parties. The few non-negotiable elements mainly concern the unreliable network infrastructure used to exchange the data.

Lines 63 and 64 (of Table 4) refine the mapping between an evaluation and an obligations. The use of a measurement scale based on the money allows the ws consumer to express his emotions and feelings captured by evaluations. An
**Action guarantee** can be tied to the respect of one or more determined *service level objective(s)*. Through those *action guarantees*, *service level objectives* can be linked to financial penalties and rewards [37]. A positive compensation reflects his favour toward a *service level objective*; a negative one reflects his disfavour. If the rewards (penalties) of two *service level objectives* are different, then the ws consumer expresses a preference for the more expensive one; if he agrees to pay more for a specific level of a QoS characteristic, that means he prefers this characteristic in comparison with other (cheaper) ones. Then, the ws discovery tool has to find the accurate service which respects this SLO for the price set by the consumer.

Requirements 6 and 7 are refined as *individual evaluation* (see respectively Requirements 15 and 16). Requirement 8 is refined as a *comparative evaluation* (see Requirement 17). Requirements 16 and 17 are respectively reproduced in Specifications 6 and 7.

**Requirement 15. Individual evaluation:** A response time of 600ms is the maximum accepted.

**Requirement 16. Individual evaluation:** A response time of 400ms is evaluated to 0.02 monetary unit per use.

**Specification 6. “PaymentType”** is defined in Appendix A.

```xml
<Obligations>
  <ServiceLevelObjective name="RP400ms">
    <Obliged>Provider</Obliged>
    <Expression>
      <Predicate xsi:type="wsla:Less">
        <SLAParameter>ResponseTime</SLAParameter>
        <Value>400</Value>
      </Predicate>
    </Expression>
    <EvaluationEvent>NewValue</EvaluationEvent>
  </ServiceLevelObjective>
  <ActionGuarantee name="RewardRP400ms">
    <Obliged>consumer</Obliged>
    <Not>
      <Expression>
        <Predicate xsi:type="wsla:Violation">
          <ServiceLevelObjective>RP400ms</ServiceLevelObjective>
        </Predicate>
      </Expression>
    </Not>
  </ActionGuarantee>
</Obligations>
```
Requirement 17. **comparative evaluation**: a response time of 400 ms is preferred to a response time of 600 ms.

**Specification 7.**

```
<QualifiedAction>
  <Party>customer</Party>
  <Action actionName="RewardPayment" xsd:type="PaymentType">
    <Debtor>Customer</Debtor>
    <Amount>0.002</Amount>
    <CausingGuarantee>RP400ms</CausingGuarantee>
    <Currency>USD</Currency>
  </Action>
</QualifiedAction>
<ExecutionModality>Always</ExecutionModality>
</ActionGuarantee>
</Obligations>

<ServiceLevelObjective name="RP600ms">
  <Obliged>Provider</Obliged>
  <Expression>
    <Predicate xsi:type="wsla:Less">
      <SLAParameter>ResponseTime</SLAParameter>
      <Value>600</Value>
    </Predicate>
  </Expression>
</ServiceLevelObjective>
<EvaluationEvent>NewValue</EvaluationEvent>
</ServiceLevelObjective>
<ServiceLevelObjective name="RP400ms"> ... [See previous specification] ... </ServiceLevelObjective>
<ActionGuarantee name="RewardRP400ms">
  <Obliged>consumer</Obliged>
  <Not>
    <Expression>
      <Predicate xsi:type="wsla:Violation">
        <ServiceLevelObjective>RP400ms</ServiceLevelObjective>
      </Predicate>
    </Expression>
  </Not>
</ActionGuarantee>
<EvaluationEvent>NewValue</EvaluationEvent>
```
<Party>customer</Party>
<Action actionName="RewardPayment" xsd:type="PaymentType">
  <Debtor>Customer</Debtor>
  <Amount>0.002</Amount>
  <CausingGuarantee>R400ms</CausingGuarantee>
  <Currency>USD</Currency>
</Action>
</QualifiedAction>
<ExecutionModality>Always</ExecutionModality>
</ActionGuarantee>
</Obligations>

5 A tool operating thanks to the proposed mappings: STR@WS

A tool, named STR@WS for Specifications Transcribed from Requirements in a ws environment (hence the @ws in the name), has been implemented. It employs the mappings developed in §4.3. In this section, we present STR@WS. First, we briefly state the technologies used to implement the tool (§5.1) followed by a description of the tool architecture (§5.2). In §5.3, we illustrate how to use STR@WS. In order to refine the one-to-many mappings, we build decision trees which are developed in §5.4.

5.1 The technologies used

Our tool is developed with the language Java O.O. We also use the JAXB API\(^7\) which allows us to translate XML document into Java object as well as marshalling, unmarshalling and validating XML documents based on XSD or DTD documents.

5.2 The functionalities of STR@WS

STR@WS is compounded of the five following modules:

1. **RequirementsEditor** allows a ws consumer to add and remove requirements about a service he is describing.

\(^7\)https://jaxb.dev.java.net/
2. **Translator** bridges the requirements expressed by the WS consumer with the WSLA/WSDL concepts based on the mapping between CORE and the WSLA/WSDL specifications.

3. **MappingRefinement** helps refining the one-to-many mappings –see Lines 59, 60 and 61 of Table 4. We build three decision trees, which are used by the requirement engineering to refine the problematic mappings (See 5.4 for the development of these decision trees). STR@WS supports this process.

4. **OpenFile** enables to open a specification file or a requirements file which has been saved with STR@WS. The file format chosen is XML.

5. **SaveFile** enables to save a specification file or requirements file.

Fig. 1 shows the main window of STR@WS as well as the tool menu.

### 5.3 The use of STR@WS through our scenario

We now go back to the scenario explained in §4.1 and discussed in §4.2 and in §4.3. In this section, we illustrate how our tool uses the mappings between CORE and the WSLA/WSDL taxonomy and can help requirements engineers during the development of a service-based system.

In Fig. 2, Requirements 10 to 15 are entered in the RequirementsEditor of STR@WS. Once the nature of the requirement is selected by the user –i.e., the requirement is a functional goal for instance–, STR@WS gives the corresponding
concept of the WSLA/WSL taxonomy. This information is displayed in green at the very right of the window. If the core concept has several corresponding service concepts, then the message displayed in red is “One to Many” and the Refine button is clickable. By clicking on it, a new window is opened. It allows the user refining the one-to-many mappings according to the decision trees described in §5.4. This window is shown in Fig. 3; the refinement of the first functional goal is the example shown (the decision path followed is surrounded).

At the end of the refinement process, the right service concept is displayed in green on the main window and that information is saved in the tool database –see the first requirement of Fig. 2 which is the only one to have been refined. STR@WS allows the user to enter a requirement having its category set at “Raw” if he does not yet know the right nature of this requirement.

The lower part of Fig. 2 depicts the translated file in which the requirements are mapped to their corresponding concept in the service taxonomy. The meaning of the tags used is as follows: <metric/> for metrics, <ag/> for action guarantees, <op/> for operations, <slo/> for service level objectives, <bind/> for bindings, <oblig/> for obligations and <unkw/> for unlinked requirements.

---

8This last tag is used when a one-to-many mapping has not been refined, or if the requirement engineer uses the “Raw” category.
Figure 3: Illustration of the use of the decision trees through the refinement of a functional goal as example

Fig. 4 shows a WSLA extract of the individual evaluation entered in the main window of STR@WS which has been translated into a WSLA/WSDL extract.

5.4 The decision trees for one-to-many mappings

The mappings formalized by Lines 59, 60 and 61 (Table 4) are one-to-many relationships. For each of them, we build a decision tree in order to refine their categorization in the accurate WSLA/WSDL class. For each one-to-many mapping, some questions related to the content of the involved requirement are asked to the tool user; she only has to answer by ‘Yes’ or ‘No’. At the end of each decision tree, the right category is proposed. Figs. 5(a), 5(b) and 5(c) illustrate the decision trees developed below.

For the Functional goal requirements (Line 59), there are three possible corresponding classes: Metric, Action guarantee and Operation. The structure of the decision tree is shown in Fig. 5(a). Its content is as follows:
Figure 4: Corresponding result of the mapping of the individual evaluation

1: Does the functional goal describe interaction(s) between the parties involved in the service use?
   
   If Yes, then link the requirement to the Operation class.
   
   If No, then go to Question 1.1.

1.1: Does the functional goal describe how a QoS property is measured?
   
   If Yes, then link the requirement to the Metric class.
   
   If No, then link the requirement to the Action guarantee class.

Concerning the Quality constraint (Line 60), there are two possibilities in the mapping: Service level objective and Binding. The decision tree, illustrated in Fig. 5(b), is as follows:

2: Does the quality constraint capture the needs about the format or the technologies used to exchange data with the service provider?
Does the **functional goal** describe interaction(s) between the parties involved in the service use?

- **Yes**
- **Operation**
- **No**

Does the **functional goal** describe how a QoS property is measured?

- **Yes**
- **Metric**
- **No**
- **Action guarantee**

(a) The decision tree for the one-to-many relationship implying the **Functional goal** class

Does the **plan** describe a process to follow?

- **Yes**
- **No**

Does the **plan** describe how a QoS property is measured?

- **Yes**
- **Metric**
- **No**
- **Operation**

Does the **plan** state a commitment of a party involved in the service use?

- **Yes**
- **Action guarantee**
- **No**
- **Binding**

(b) The decision tree for the one-to-many relationship implying the **Quality constraint** class

(c) The decision tree for the one-to-many relationship implying the **Plan** class

Figure 5: Decisions trees for the one-to-many mappings

If **Yes**, then link the requirement to the **Binding** class.

If **No**, then link the requirement to the **Service level objective** class.

The last one-to-many relationship implies the **Plan** concept (Line 61) with
four possible corresponding concepts: Binding, Metric, Action guarantee and Operation. The decision tree, illustrated by Fig. 5(c), is as follows:

3: Does the plan describe a process to follow?
   If Yes, then go to Question 3.1.
   If No, then go to Question 3.2.

3.1: Does the plan describe how a QoS property is measured?
   If Yes, then link the requirement to the Metric class.
   If No, then link the requirement to the Operation class.

3.2: Does the plan state a commitment of a party involved in the service use?
   If Yes, then link the requirement to the Action guarantee class.
   If No, then link the requirement to the Binding class.

6 Related work

Two tools [67, 68] and a method [69] have been proposed in order to ease the WS discovery process. Based on textual requirements, WS matching the WS consumer needs are suggested. However, these works exclusively focus on functional requirements and the requirements are expressed without any restructure. That makes the discovery task more demanding in methods for extracting accurate information from the various requirements.

Rolland et al. [70] introduce a model for Intentional Service Modelling (ISM): WS providers have to describe their WS and WS consumers use an “intentional matching mechanism” to select potential WSs. This model requires new technologies for publishing, browsing and discovering services in comparison to the most widespread ones, i.e., UDDI and ebXML registries. The QoS characteristics of WSs are not considered in the discussion. Another relevant paper [71] uses the ISM approach. The authors improve the work of Rolland and colleagues by taking into account the QoS levels of WS during the matching and selection step.

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 the service providers learn how this language has to be used.

Regarding the solutions of semantic matching between the WS descriptions
and the needs of the WS consumer, related work is often built on technical
languages and specifications. For instance, [72], [73] and [74] respectively use
USQL (Universal Service Query Language), DAML-S and BPOL (Business Process
Outsourcing Language). The handling of those technologies requires thorough
knowledge of each of them. Works on semantic matching often concentrate on
the WS provider side, e.g., [75, 76, 77, 78]. In order to have a comprehensive
approach of the problem, we also need a user-friendly solution that eases the
requirements elicitation task at the WS consumer side.

In [79], 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 the errors or conflicts discovery.
The authors create their own meta-model for the four elements required in service
consumption (i.e., role, goal, process and service). The method and the tool are
very interesting. However, they are grounded in the WS literature turned towards
the service producer [22]. By grounding the RE for services in a generic ontology
for requirements, we take the point of view of the service consumers. This is very
important to adopt the consumer point of view in order to build a comprehensive
method and/or tool supporting the whole RE process for definition of service
requests.

The work of Zachos et al. [80] shares some similarities with ours. They create
a tool which is able to discover WSs based on requirements expressed by the user
in natural language. The requirements elicitation process depends on use-case
analysis. Requirements related to the use-cases are then added in the system,
UCARE, which follows the VOLERE requirements shell. The scope of our work is
more restricted than theirs: we focus exclusively on the mapping between the
requirements of the WS consumer and WSLA/WSDL. Our approach uses CORE
as the source of requirements concepts, rather than use cases. Moreover, we
formalize the mapping between the requirements, which could be expressed in
natural language, and their specifications. First, it will allow to keep the track of requirements when a WS is selected. If the system-to-be selecting WSs cannot replace a defective WS, it is able to identify too demanding requirements by comparing the characteristics of the best fitted WS and the consumer requirements contained in the service request. Secondly, it enables to directly analyze the consequences of requirements changes in comparison with the (composite) WS chosen. This is very important for requirements monitoring in an SOA, as already noted in [81]. With regards to works related to RE monitoring in a service-oriented environment [81, 82], proposed methods to elicit requirements are based on RE techniques. Our contribution could be complementary to those works in order to improve the RE process.

In [83], the authors propose a WS composition framework based on state machines. Their system iteratively helps WS consumers to elicit their needs. In case of problems during the WS composition, the causes are exposed to the service consumer. Then, the system helps him to reformulate his needs. Seeing that there is not ontological grounding for the requirements expressed by WS consumers, the latter must know both the RE and the service context, and relates himself those two conceptualizations. Our view on the problem allows the WS users and their software/requirements engineers to concentrate only on the RE issue.

The last significant paper [84] related to this work proposes an online monitoring of the WS requirements. The aim of the authors is to make the behaviour of WSs consistent with the requirements of the service consumer. In this way, they design a novel language, the Web Service Constraint Description Language (WSCDL), with which values and events constraints are captured. As in many other works, a new “standard” is once again proposed. Secondly, the content of a WSCDL file obviously comes from an RE work. However, this is not clearly underlined neither explained. Therefore, our work is complementary to their research: we point out the origin of the service request content by bridging the requirement types to the service concepts. It should improve the monitoring of the requirements, and especially the understanding and the forecasting of the
consequences of changes in the service consumer needs.

7 Conclusion

Service-oriented computing raises new issues, included the management of the requirements: mainly, their elicitation, their capture, their analysis and their specification into a service request. In the literature, authors often work with pure technical specifications to capture and specify the service consumer’s requirements. Adding a clear link between an ontology for requirements and a service taxonomy allows (i) to move a step closer to the automation of the creation of service requests based on the WS consumers’ requirements, (ii) to help the WS composition system to identify easily non-suitable requirements asked by the WS consumer, (iii) to know which requirements are no longer satisfied when a WS provider fails to comply with the agreement and (iv) to know precisely which part of an WSLA and/or WSDL document must be modified when the WS consumer changes some of his requirements. Creating and keeping this link is enabled by the proposed mappings between the two conceptualization on the problem tackled in this paper. The main original idea is to base the high level representation from an ontology for RE and translate it to WS descriptions.

7.1 Future work

Taking into account the possible faults of the service oriented system in its actual operation is a priority for future work. Reinecke, Wolter and Malek’s contributions [85] appear to be a relevant starting point towards that aim. They propose an overview of the fault-models available both in WS technologies (e.g., WSDL, see §2.3), and in communication technologies (e.g., HTTP).

On the RE side, a requirement modeling language should be created or adapted in order to capture the requirements expressed by the WS consumers. In order to (automatically) reason on the requirements expressed, we have to structure them. The requirements modeling language could be grounded on Techne [86]. This would also ease the translation of a RE solution to a specification of the
service request which is usable by discovery tools.

This paper does not cover the difference between hard and soft slo s. WS consumers often express their minimal requirements regarding the non-functional characteristics of the ws as well as additional (soft) slo s increasing their satisfaction. It also avoids the issue of requirements concerning orchestration and choreography. Before tackling this question, RE for a single ws should be done more suitably.

Taking into account the gaps (see §4.2) between the two levels of requirements representation is also a future task. This can be done within a wider is composed of our tool as well as other computational modules enabling the discovery and the composition of ws s based on the wsla/wsdl specifications.

The last point to improve is the process followed to refine the one-to-many mappings. It could be enhanced by, e.g., adding a syntactic and/or semantic matching based on the requirements content.

References


A Appendix: Definition of PaymentType

Here is the XML Schema of the PaymentType element.

Specification 8.

<xsd:complexType name="PaymentType">
  <xsd:sequence>
    <xsd:element name="Debtor" type="xsd:string"/>
    <xsd:element name="Amount" type="xsd:float"/>
    <xsd:element name="CausingGuarantee" type="xsd:string"/>
    <xsd:element name="Currency" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>

B Appendix: Definition of AddressTransmissionType and CoordinatesTransmissionType

Here are the XML Schema of the AddressTransmissionType element and of the CoordinatesTransmissionType element.


<types>
  <element name="AddressTransmissionType"/>
  <complexType>
    <sequence>
      <element name="Street" type="string"/>
      <element name="Number" type="integer"/>
    </sequence>
  </complexType>
</types>
<element name="Box" type="string"/>
<element name="ZIP" type="integer"/>
<element name="City" type="string"/>
<element name="Country" type="string"/>
</sequence>
</complexType>
</element>
<element name="CoordinatesTransmissionType">
<complexType>
<sequence>
<element name="Latitude" type="OneCoordinateType"/>
<element name="Longitude" type="OneCoordinateType"/>
</sequence>
</complexType>
</element>
<complexType name="OneCoordinateType">
<sequence>
<element name="Degree" type="intDegree"/>
<element name="Minute" type="int2"/>
<element name="Second" type="int2"/>
</sequence>
</complexType>
<simpleType name="intDegree">
<restriction base="integer">
<totalDigits value="3"/>
<minInclusive value="-180"/>
<maxInclusive value="180"/>
</restriction>
</simpleType>
<simpleType name="int2">
<restriction base="integer">
<totalDigits value="2"/>
<minInclusive value="-60"/>
<maxInclusive value="60"/>
</restriction>
</simpleType>
</types>