# Improving Web Service Composition with User Requirement Transformation and Capability Model

Wenbin Li, Youakim Badr, and Frédérique Biennier

INSA-Lyon
Université de Lyon
Villeurbanne, France
{wenbin.li,youakim.badr,frederique.biennier}@insa-lyon.fr

**Abstract.** In order to discover and compose relevant Web services, most Web service composition approaches require users to describe composition requirements and constraints in formal expressions. However, requirements still focus on the technical level as they require domain-specific knowledge on functional and non-functional properties. As a matter of fact, the gap between users' *high-level requirements* describing business objectives and *composition requirements* remains a challenge in Web service composition. In this paper, we propose an end-to-end transformation approach to build a set of technical composition requirements from user high-level requirements, which are specified by an English-structured language to capture business objectives, and to specify actions that Web services can achieve.

**Keywords:** Web services, requirement transformation, service capability.

## 1 Introduction

Web service composition is the process of reusing existing Web services and logically recombining them into composite services in order to provide new functionalities that existing web services cannot provide alone. Composition requirements refer to the constraints imposed on the Web service composition process. Generally, they include various constraints on and among Web services (e.g., control flow constraints, functional and non-functional properties constraints, and dependency constraints, etc.).

However, existing Web service composition approaches [1] focus on how to compose Web services together and highly rely on formal specification of users' composition requirements. Nevertheless, these approaches fail to provide casual users with a natural-like means to express their high-level requirements and to describe their business objectives to be achieved. The gap between user's *high-level requirements* describing business objectives and *composition requirements* specifying Web services details raises a challenge as to how transform high-level requirements in natural-like languages to formalized composition requirements that can be directly used in Web service composition. In order to solve this challenge, we introduce a transformation method between high-level requirements and composition requirements via a Web service capability model. The capability model denotes what an action does in terms

R. Meersman et al. (Eds.): OTM 2013, LNCS 8185, pp. 300–307, 2013.

of changes in the state of Web services that are involved in compositions [2]. It describes what Web services can do without needing to know all the technical details. The composition requirement transformation method comprises three pillars: a composition requirement meta-model; a three-layer capability model; a transformation process for requirements at different levels.

The remainder of this paper is organized as follows: related works are presented in section 2. In section 3, we introduce the composition requirement meta-model, and then in section 4, we demonstrate how we achieve the automated transformation between different composition requirement models. Section 5 concludes our work and sheds light on future trends.

## 2 Related Work

Requirement transformation is the process of generation of target requirements from source requirements, according to transformation rules. Requirement transformation methods are generally divided into three categories [3]: transformations based on traceability, transformations based on behavior trees, and transformations based on model transformation. Nevertheless, most of transformation methods are designated to transform formalized requirements. They lack of formalisms to express nature-likes requirements easy to be understood by casual users. The work in [4] examines existing approaches that transform textual requirements into analysis models. As direct automatic transformations of natural language requirements are very difficult due to the inherent ambiguities of natural language, manual or semi-automated transformation are proposed based on user interventions [5][6] or natural requirements defined by given patterns [7][8]. Moreover, Web service discovery and selection either rely on syntactical keywords or semantic relationships based on domain ontology [9]. However, composition requirement does not only apply on functional and non-functional properties of Web services to be composed, but they also concern the composition process (e.g., control flow constraints, dependency constraints, etc.). By discovering Web services prior to the composition, users often are involved in manually specifying constraints on and among Web services. In conclusion, the gap between requirements specified by causal users and technical requirements on Web services persists and requires automated transformation solution to derive from requirements in natural-like languages, describing business objectives, multiple constraints on Web services and their composition process. In addition, some research efforts attempt to provide an abstraction view of Web services by introducing the Web service capability model such as IOPE (e.g., WSMO [10] and OWL-S [11]) or case-frames to model Web services as action-verbs and informational attributes, describing their behaviors [12]. However, we argue that describing service capabilities only based on action-verb and attributes are oversimplified and causing ambiguity without explicit descriptions of the domain by which service capabilities are manipulated.

# 3 Composition Requirement Meta-model

In order to allow either developers or business people expressing their composition requirements, we model composition requirements in three levels. namely High-level Requirement; Formalized Business Requirements; Web Service Requirements.

## 3.1 High-Level Requirement (HLR)

High-level requirements refer to constraints that are expressed in a natural-like language to describe user business objectives and desired actions to be performed. In our work, we model user's high-level requirement as a subset of Semantics of Business Vocabulary and Business Rules (SBVR) [13]. SBVR is a formal business rule language with a natural language interface, which implies that either casual users or domain experts can express their requirements with a natural-like language. SBVR provides **facts** to define *noun concepts* (i.e., simply relations among noun concepts) and write **rules** with *modal operators*, *quantifiers*, *qualifiers* and *facts*.

As depicted in Figure 1, we model the high-level requirements **(HLR)** as a set of functional requirements (**FR**), non-functional requirements (**NR**) and a **Context** such as: $HLR = FR \cup NR \cup Context$

1. **FR** consists of two parts: a) **Objective** descriptions based on SBVR facts to define a list of business objectives to be achieved; b) **Information** descriptions based on SBVR rules to describe a list of actions to be performed, where :

$$FR.Objective = \{obj_1, obj_2, \ldots, obj_n\},\ FR.Information = \{i_1, i_2, \ldots, i_n\}$$

2. **NR** is defined based on SBVR rules to specify a list of constraints on actions to be performed, where $NR = \{nf_1, nf_2, \ldots, nf_n\}$

3. **Context** is defined based on SBVR facts, describing generic domain states (e.g., date, location, relation between concepts, etc.), where $Context = \{ctt_1, ctt_2, \ldots, ctt_n\}$

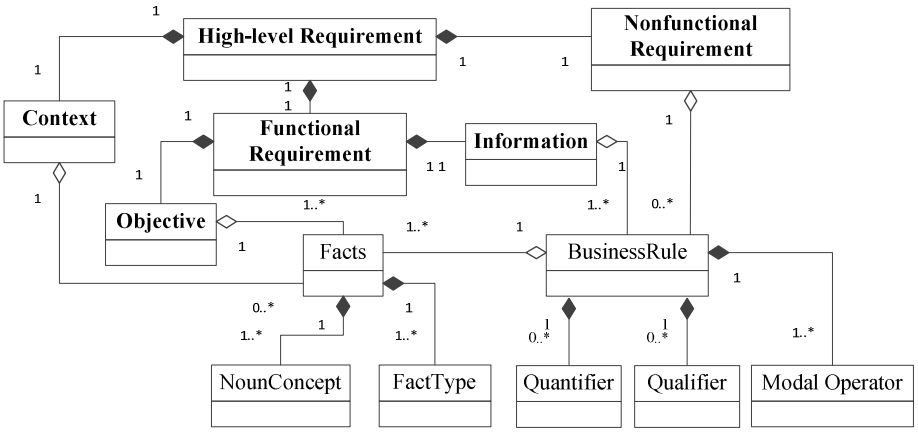


**Fig. 1.** High-level Requirement Model

In order to illustrate our composition requirement models and its transformation method as described later, we introduce a crisis scenario of two trains' crash in Paris with a list of negative facts describes damages, such as caused fire, interrupted electricity supply, etc. We assume that different actions, managing the crisis, are regarded as Web services while the whole crisis response process is regarded as the execution of composite Web services. Table 1 illustrates High-level Requirements for the train crisis scenario.

**Table 1.** High-level Requirement Example

| | |
|---|---|
| **FR.Objective** | **obj$_1$**: *Manage* train crisis |
| **FR.Information** | **i$_1$:** Fire should *be extinguished.*<br>**i$_2$:** Victims should *be assisted.*<br>**I$_3$:** Electricity should *be recovered.* |
| **NF** | **nf$_1$:** It is obligatory that at least 10 firemen *extinguish* fire.<br>**nf$_3$:** It is obligatory that the electricity *is recovered after* the fire *is extinguished.* |
| **Context** | ctt$_1$: Crisis place *is* Pairs.<br>ctt$_2$: Crisis date *is* 2013/03/01. |

### 3.2 Formalized Business Requirement (FBR)

Formalized Business Requirements describe users' business objectives and desired actions to be performed. In our work, we model formalized business requirements based on the Web service capability model, by which users describe requirements in terms of web service capability profiles. We also propose a three-layer capability model, describing Web service capabilities from different perspectives; request, offer, and composition perspectives. The three-layer capability model consists of: 1) the *objective layer* (**OL**) which describes the business objectives that the capabilities can achieve; 2) the *profile layer* (**PL**) which describes capabilities as action verb and noun pairs, and attributes, representing the real abilities that Web services perform; 3) the *composition layer* (**CL**) which creates relationships among capabilities. A Web service capability profile, denoted by **cap**, is defined as a tuple: *cap* = <*id, OL, PL, CL*>, where *id* is the identifier of this capability file. The capability files are mainly created by service providers or domain experts, and are associated with Web services. One capability file can be associated with one or more Web services. Fig. 2 shows a detailed UML class diagram of the capability model.

#### 1. The Objective Layer

The objective layer consists of one or more objectives (**obj**) described by a pair of an *ActionVerb* and a *Noun* [12]. We associate action-verbs and nouns to domain specific ontologies that establish shared agreements on their semantics. The ontology is used to semantically match different concepts.

$OL = \{obj_1, obj_2, \ldots, obj_n\}$ where $obj_i = \langle action\_verb_i, noun_i \rangle$ and $obj_i \in OL$

It is worth noting that an objective can be associated with different capabilities while a capability can have different objectives.

## 2. The Profile Layer

The capability profile layer is defined in terms of a capability *name* (**cname**) and a list of *attributes* (**attr**)**,** describing the real ability and characteristics that a Web service performs. In a similar way to the objective layer, the capability name is defined as a pair of an ActionVerb and a Noun. However, the capability name describes the real ability that a Web service can perform whereas the capability objective describes the objective that the capability can satisfy. Each attribute $attr_i$ is defined in terms of attribute name (**attr_name**) and attribute value (**attr_value**) as follows:

$$PL = \{cname, attr_1, \ldots, attr_n\}, \text{ where}$$

$$cname = <action\_verb_i, noun_i>, attr_i = <attr\_name, attr\_value>, \text{ and } attr_i \in PL$$

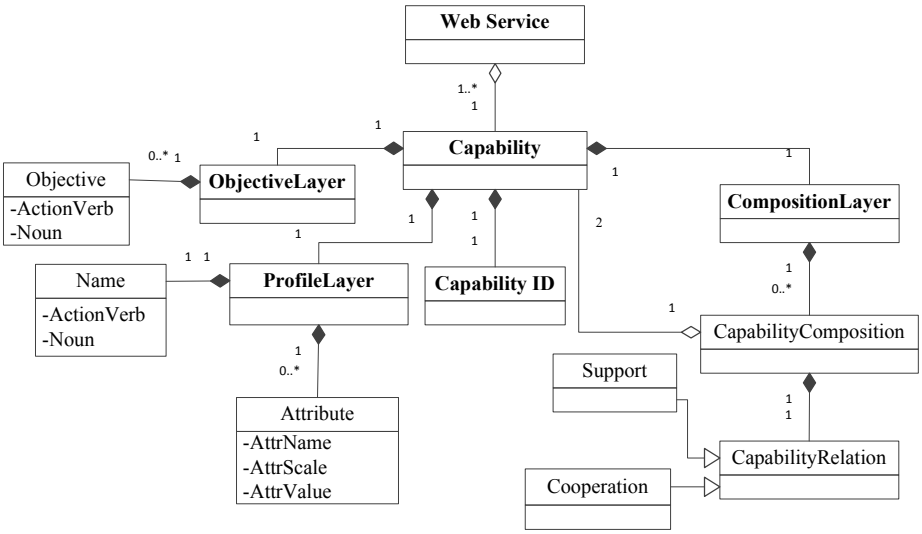


**Fig. 2.** The Capability Model

## 3. Composition Layer

The composition level consists of different capability compositions (**cc**) describing relations between capabilities. The composition layer is defined as

$CL = \{cc_i, cc_2, \ldots, cc_n\}$, $cc_i = <cap_p, cap_q, rel>$ and $cc_i \in CL$ such as $cap_p$ and $cap_q$ are two Web service capability profiles; *rel* is the relation between the two capabilities. We identify two kinds of composition relations between capabilities:

1) **Cooperation Relationship**: two capabities should cooperate with each other.

2) **Support Relationship**: the execution of one capability depends on the successful execution of its previous capabilty.

Based on the capability model, the formalized business requirements (FBR) for Web service composition are modeled as a list of capability profiles.

$$FBR = \{cap_1, cap_2, cap_3, \ldots, cap_n\}$$

**Table 2.** Capability Profiles Examples

| ID | Objective Layer | Profile Layer | | Composition Layer |
|---|---|---|---|---|
| | | cname | attr | |
| **cap$_{1.1}$** | {<manage, crisis>} | <evacuate, population> | {(AvailableRegion, France)} | |
| **cap$_{1.2}$** | {<manage, crisis>} | <evacuate, population> | {(AvailableRegion, Marseille)} | |
| **cap$_2$** | {<manage, crisis>, < rescue, people>} | <transport, victim> | {(MaxTransportNumber, 100)} | {cap$_2$, cap$_3$, Support} |
| **cap$_3$** | {<manage, crisis>, <rescue, people>} | <assist, victim> | {(MaxAssistNumber, 300)} | {cap$_2$, cap$_3$, Support} |

### 3.3 Web Service Requirement (WSR)

Web service requirements refer to constraints related multiple constraints on/among functional and non-functional Web service properties. They consists of desired Web services to be executed and constraints on/among them. We structure multiple constraints on/among Web services with Structure Rules and QoS Rules.

1) **Structure Rules** indicate composition patterns between Web services. A composition rule $\mathbf{sr}_m$ is defined as a tuple: $sr_m = <s_l, s_r, cp>$, where $s_l$, $s_r$ are the two sub services, *cp* is the composition pattern, and *cp is* sequence, parallel or branch operator, describing the control flow in the composed Web service.

2) **QoS Rules** represents constraints on Web services QoS attributes. The QoS of a service $s_i$ is represented as $\mathbf{s_i.QoS} = \{q_1, q_2, \ldots, q_n\}$, where $q_j$ represents the QoS attribute *j* of service $s_i$. A QoS rule $\mathbf{qr_n}$ is defined as: $qr_n = <s_i, q_j, expression>$ where $expression = <operator, value>$, and $s_i$, and $q_j$ are respectively the Web service and its QoS attribute to be constrained. The *expression* describes constraints whereas $operator \in \{<, \leq, >, \geq, =, \supseteq, \ldots\}$ and *value* indicates QoS attribute values.

The Web service requirement (WSR) for composition is modeled as a list of Web services to be executed and a list of composition rules:

$$WSR = \{s_1, s_2, \ldots, s_l, sr_1, sr_2, \ldots, sr_m, qr_1, qr_2, \ldots, qr_n\}$$

## 4 Composition Requirement Transformation Process

In order to achieve requirement transformations between different levels, we propose **Capability Matching**, and **Association Discovery** algorithms as illustrated in Fig. 3.


**Fig. 3.** General Methodology of Composition Requirements Transformation

These algorithms are based on semantic matching between concepts related to different requirement levels by using ontologies. We apply the concept matching algorithm proposed by [14] to cover three cases of matching between two concepts: 1) two concepts are identical, synonyms or one concept is generalization of another

concept with reference to common ontology. Due to page size limit, we only describe these two algorhtms as follows:

**Capability Matching Algorithm**: transforms high-level requirements to formalized business requirements by matching concepts from the high-level requirement model with concepts from the capabilty model following two principles:
*Principle 1.* Functional reuqirements in the HLR are matched against objective layer and profile layer of capability profiles. Capabilities satisfying objectives and information defined in functional reuqirements are discovered and added to a primary set;
*Principle 2.* Non-functional requirements and context of HLR are matched against capability profiles layer. Capabilities dissatisfying rules specified in the NR are deleted from the primary result.

**Association Discovery Algorithm**: Given discovered capabilities from the capability-matching algorithm, the association discovery algorithm transforms capabilities' descriptions into composition rules among web services following three principles:
*Principle 1.* For each discovered capability, all Web services that are associated with the capability are discovered and added to the Web service discovery set.
*Principle 2.* If NR describes a constraint on and among certain capabilities, then a composition rule will be created on and among all Web services that are associated with these capabilities.
*Principle 3.* If a composition relationship between two capabilities is defined in the compositon layer of the capability profiles, a structure rule is created between Web services that are associated with the two capabilities.

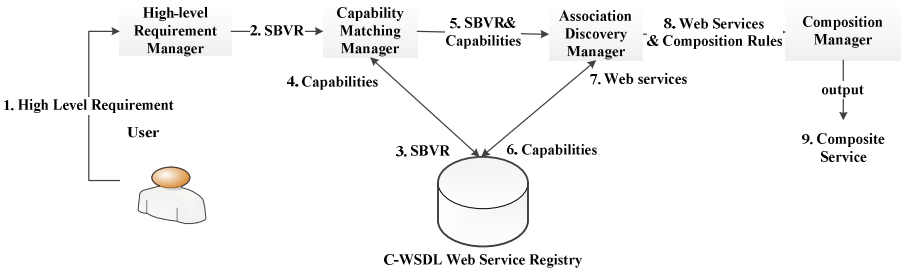


**Fig. 4.** Prototype Architecture for Requirement Transformation

## 5 Conclusion

In this paper, we propose an end-to-end transformation approach between high-level requirements and composition requirements via a Web service capability model to reduce the gap between requirements expressed by casual users in SBVR and technical requirements related to Web service composition process. We have demonstrated the feasibility of our transformation method by developing a prototype comprising different modules implementing the capability matching algorithm and the association discovery algorithm (see Fig. 4). We have also integrated the C-WSDL Web service registry and the rule-driven Web service composition algorithm developed in our previous work [15] i.e., Service Farming. Based on the train's crisis case study,

preliminary results illustrate the transformation process and generate a set of composition rules. As a future work, we are studying the Intuitionist Linear Logic to prove whether a set of capabilities can be found to ensure that high-level requirements can be translated in set of composition rules among a discovered set or Web services.

## References


1. Dustdar, S., Schreiner, W.: A Survey On Web Services Composition. International Journal of Web and Grid Services 1, 1–30 (2005)
2. OASIS Reference Model for Service Oriented Architecture 1.0 (2006), http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf
3. Jazuli Bin Kamarudin, N., Sani, N.F.M., Atan, R.: Transformation from requirement into behavior design: A review. In: 2012 International Conference on Information Retrieval & Knowledge Management (CAMP), pp. 153–157 (2012)
4. Yue, T., Briand, L.C., Labiche, Y.: A Systematic Review of Transformation Approaches Between User Requirements and Analysis Models. Requirements Engineering 16, 75–99 (2011)
5. Grünbacher, P., Egyed, A., Medvidovic, N.: Reconciling Software Requirements and Architectures With Intermediate Models. Software and Systems Modeling, 235–253 (2004)
6. Fliedl, G., Kop, C., Mayr, H.C., Salbrechter, A., Vöhringer, J., Weber, G., Winkler, C.: Deriving Static and Dynamic Concepts from Software Requirements Using Sophisticated Tagging. Data Knowledge Engineeriong 61, 433–448 (2007)
7. Subramaniam, K., Liu, D., Far, B.H., Eberlein, A.: UCDA: Use Case Driven Development Assistant Tool for Class Model Generation. In: 16th International Conference on Software Engineering and Knowledge Engineering (SEKE 2004), Banff, Canada (2004)
8. Samarasinghe, N., Somé, S.: Generating a Domain Model from a Use Case Model. In: Proceedings on Intelligent and Adaptive Systems and Software Engineering (2005)
9. Mukhopadhyay, D., Chougule, A.: A Survey on Web Service Discovery Approaches. In: Wyld, D.C., Zizka, J., Nagamalai, D. (eds.) Advances in Computer Science, Engineering & Applications. AISC, vol. 166, pp. 1001–1012. Springer, Heidelberg (2012)
10. Roman, D., de Bruijn, J., Mocan, A., Lausen, H., Domingue, J., Bussler, C., Fensel, D.: WWW: WSMO, WSML, and WSMX in a nutshell. In: Mizoguchi, R., Shi, Z.-Z., Giunchiglia, F. (eds.) ASWC 2006. LNCS, vol. 4185, pp. 516–522. Springer, Heidelberg (2006)
11. Martin, D., Paolucci, M., Wagner, M.: Towards Semantic Annotations of Web Services: Owl-S from the SAWSDL Perspective. In: Proceedings of the OWL-S Experiences and Future Developments Workshop at ESWC (2007)
12. Bhiri, S., Derguech, W., Zaremba, M.: Web Service Capability Meta Model. In: WebIST 2012, pp. 47–57 (2012)
13. Fayoumi, A., Yang, L.: SBVR: Knowledge Definition, Vocabulary Management, and Rules Integrations. International Journal of E-Business Development (2013)
14. Meditskos, G., Bassiliades, N.: Structural and Role-Oriented Web Service Discovery with Taxonomies in OWL-S. IEEE Transactions on Knowledge and Data Engineering 22, 278–290 (2010)
15. Li, W., Badr, Y., Biennier, F.: Service farming: An Ad-Hoc and Qos-Aware Web Service Composition Approach. In: Proceedings of the 28th Annual ACM Symposium on Applied Computing, pp. 750–756. ACM, New York (2013)