# Web Services Customization : A Composition Based Approach

Yacine Sam, Omar Boucelma *
{yacine.sam, omar.boucelma}@lsis.org

Mohand-Saïd Hacid **
mshacid@liris.cnrs.fr

**Abstract** In order to fulfill current customers requirements, companies and services providers need to supply a large panel of their products and services. This situation has led recently to the Mass Customizing Paradigm, meaning that products and services should be designed in such a way that makes it possible to deliver and adapt different configurations. The increasing number of services available on the Web, together with the heterogeneity of Web audiences, are among the main reasons that motivate the adoption of this paradigm to Web services technology.
In this paper we describe a solution that allows automatic customization of Web services: a supplier configuration, published in a services repository, is automatically translated into another configuration that is better suitable for fulfilling customers' needs.

**Keywords:** Web service, semantic Web, Web services composition, Mass Customizing Paradigm.

## 1 Introduction

The Web is not only an enormous warehouse of text and images, its evolution made it also a services provider [13]. The *Web service* concept refers to an application advertised over the Internet and made accessible to services requesters through standard Internet protocols. Currently available examples of Web services are weather forecasting, online tickets reservation, banking services, etc. Globally, Web services are defined as *well defined, loosely coupled software components* and constitute therefore a new paradigm for applications integration [4].

Web services are currently implemented through three standard technologies: WSDL [21], UDDI [19] and SOAP [17]. These standards provide only syntactical interoperability that prevents agents for automating services discovery, selection and composition tasks. The semantic Web provides standards for representing and processing computer-interpretable information [5]. Semantic Web services are then a synthesis of these two standards and constitutes therefore a good proposal for the automation of the various tasks of Web services life cycle.

Semantic Web services descriptions relay on Web services annotations with terms having a formal description in structured dictionaries called ontologies. DAML+OIL [6] is a logic based language intended to the description of the most known Web services. It is used directly or through DAML-S [1]. DAML-S is a DAML+OIL concepts ontology describing technical aspects of Web services (Inputs/Outputs parameters, data types, etc). OWL [14], an evolution of DAML+OIL, was recently standardized by the W3C. OWL is now the main standard for Web ontologies descriptions and OWL-S is the corresponding evolution of DAML-S.

Other languages represent interesting solutions for automatic Web services discovery, selection and composition. One can quote GOLOG [9] and LARKS (Language for Advertising

* LSIS–CNRS Université Aix-Marseille 3. 13397 Marseille Cedex 20, France.
** LIRIS–CNRS Université Claude Bernard Lyon 1. 69622 Villeurbanne cedex, France.

and Requesting for Knowledge Sharing) [18]. The latter is a frame based language for semantic Web services discovery and selection. The former is a Situation Calculus Language and was adapted in [12] for Web services composition.

In this article, we will adapt LARKS in order to elaborate a Web services customization framework. Mass Customization Paradigm [8] is a principle which considers that the products must be conceived in such a way that make it possible to satisfy a maximum of different clients' needs. Exponential proliferation of services provided through the Web and the cosmopolitan aspect of the Web users justify the Mass Customization Paradigm for Web services.

We propose a framework that allows the automatic customization of Web services. The basic idea is to automatically transform services published in the services directories in order to generate suitable configurations for answering client needs. The goal of the automatic transformation, based mainly on the dynamic composition of Web services, is twofold. On one hand, the construction task of a given service becomes easier for the providers. On the other hand, the requesters will be able to obtain services in the alternatives that can satisfy their preferences, even if those are not explicitly present in the services directory. Thus, services providers publish explicitly only one configuration of a given service, the others are deduced, from the services directory, dynamically and in an automatic way at the requesting time.

The rest of this article is organized as follows: we provide, in Section 2, a motivating example that illustrates customer requirements for customizable Web services. Section 3 presents LARKS and its use for advertising and requesting for Web services. In Section 4, we develop our approach to Web Services customization. We initially propose a new structure for semantic Web services which allows, contrary to LARKS, to take account the services customization idea. We describe then the Web Services matchmaking process, and finally the automatic Web services customization algorithm. We conclude and give future directions in Section 5.

## 2 Motivating Example

Each year, "La Fête de la lumière" (Light Celebrates) is the most important traditional popular demonstration in Lyon (a French town). Before traveling to Lyon, a Japanese tourist wants to obtain information about the available Hotels there and their fees. Thus, he sends his request to a Web directory which stores this information in a Web services form, expecting for Web services that can satisfy his informational needs.

After the request processing, the services directory seems to be unable to satisfy the information required by the japanese customer. Indeed, the services turned over describe Hotels in French with their fees in Euro. This makes them useless for the customer, which understands only the Japanese language and uses the local currency : Yen. The fundamental report through this scenario is that the customer is not satisfied because the requested Web service exists in the directory in an incompatible configuration compared to its needs.

The framework we propose allows transformation of Web services. The basic principle of the Web service transformation consists in dynamically calling two intermediate Web services. The first will translate the Hotels descriptions from French to Japanese and the second will transform the fees from Euro to Yen. The answer to the Japanese customer request will then be built by the coordination of these two intermediate services and the service initially available in the directory. The following section introduces the LARKS language.

## 3 Larks Language

LARKS is an advertising and requesting language for Web services. In LARKS, services and requests are both specified in the form of a frame. The frame's attributes are described hereafter:

- **Context** : it represents a keyword describing what the service does.
- **Types** : definition of the abstract data types used in the specification.
- **Input/Output** : declaration of the Input/Output variables of the service.
  *Context* and *Input/Output* attributes can be annotated by machine-interpretable concepts stored in the attribute *ConcDescription*.
- **InConstraints/OutConstraints** : logical constraints on the Inputs/Outputs. These constraints can be restrictions on the Inputs/Outputs values or logical constraints between the service Inputs/Outputs.
- **ConcDescripton** : formal description of the concepts being used for the semantic annotation of the context and the Inputs/Outputs of the services. The association of a concept $C$ to a word (Context or Input/Output) $w$ is noted $w*C$, which means that the concept $C$ is the formal description of the word $w$. The use of formal ontologies in LARKS makes it possible to semantically describe the Web services. Ontologies can be described formally with concept languages like ITL [3], LOOM [11], or KIF [2].
- **TextDescription** : textual description of the service requester needs or what can offer a service provider.

In LARKS, the constraints are used to restrict the values of an Input/Output. However, the assignment of a measuring unit to an Input/Output can only be specified by semantic annotations using extensional formal concepts. Extensional concepts are sets of instances (objects) used, in this case, to capture the set of Input/Output's measuring units. During the Web Services matchmaking process in LARKS, the comparison of the two different concepts EURO and YEN (See Figure 2) leads directly to failure. Indeed, no knowledge is available to capture that these two concepts can be convertible. Consequently, no Web services transformation is possible in this language. In what follows, we propose a new Web services structure which constitutes the foundation of their customization process. It allows in particular the services directory to detect, during the matchmaking process, that two concepts (measuring units) can be convertible by another service. This fact avoids the immediate failure of the matchmaking process as it is in LARKS. In the rest of this article, the term service if used solely means a service offered by a service provider. The service requested by the customer is designed by the term request.

## 4 Our Approach to Web Services Customization

### 4.1 A New Structure for Web Services

The structure suggested in this article is used to specify both the services and the requests. It is made up of two subsystems : the Structural System, and the Constraints System.

#### 4.1.1 The Structural System (*SS*)

The *SS* is defined by the triplet $(C, I, O)$. $C$ is the context of the specification, it is defined by a keyword related to the specified service. $I$ and $O$ are respectively the description of the Input/Output variables and their abstract data types in a service or in a request. In Figure 1, the abstract data type of the attribute *price*, that represents the price of a book, is *Real* in the

Output of the service specification. The keywords of the triplet $(C, I, O)$ can be annotated by formal concepts defined in an ontology, which we consider to be shared between all the users of a specific domain.

**Exemple 1** *Figure 1 illustrates the SS of a books-sale service. It is described by its context "Book" and its Inputs/Outputs "your-book"/("Price", "presentation"). The Output parameters "Price" and "Presentation" are annotated by the concepts* **Price** *and* **Description** *respectively.*

| | |
|---|---|
| $C$ | Book |
| $I$ | Your-Book:String |
| $O$ | Price*__Price__:Real, Presentation*__Description__:String |

**Figure 1:** A Structural System Example.

The formal concepts **Price** and **Description** – see Figure 2 – are used to assign types to the Web service Inputs/Outputs Price and Presentation respectively. By the Inputs/Outputs' types we do not mean the abstract data types (Integer, Real, etc), but the measuring units used to express the values of the Inputs/Outputs in the domain ontology. In Figure 1, the measuring unit of the Output Price is defined by the concept **Price** which corresponds to a set of currencies : Dollar(USD), Euro(EUR) and Yen(YEN) in the ontology.

**Price** = **Money**
**Money** = (and Real (all in-currency aset(USD, EUR, YEN)))
**Euro** = (and Real (all in-currency aset(EUR)))
**Yen** = (and Real (all in-currency aset(YEN)))
**Dollar** = (and Real (all in-currency aset(USD)))
**Description** = **Language**
**Language** = (and String (all in-currency aset(English, French, Japanese)))
**Japanese** = (and String (all in-currency aset(japanese)
**French** = (and String (all in-currency aset(French)

**Figure 2:** Examples of formal concepts defined in ITL language

Note that the fact that the concept **Price** contains several measuring units can seem inconsistent since an attribute value can only have one measuring unit at time. However, the annotation with this kind of concepts (sets of measuring units) is used only for one partial services matchmaking that determinates the services *likely* able to satisfy the request. There is a second services matchmaking stage where only services being effectively able to satisfy it will be selected.

### 4.1.2 The Constraints System (*CS*)

The *CS* allows the specification of two kinds of constraints : constraints on the values of the Inputs/Outputs and constraints on their types in the domain ontology (typing constraints). The CS is defined by the quadruplet $(I_{ct}, O_{ct}, I_{cv}, O_{cv})$ where the elements are sets of constraints on the Inputs types, Outputs types, Inputs values and Outputs values respectively.

Our main interest in this article is the Web services customization. Thus, we focus on the typing constraints that make it possible to specify the measuring units of the services specifications Inputs/Outputs values. The typing constraints can be regarded as the specialization of the concepts used at the time of semantic annotation level of the *SS*. The role of the typing constraints is to specify by exactly one type (measuring unit) each Input/Output. The following example illustrates this. According to Figure 2, the concept **Price** is equivalent to the concept **Money** which is an extensional concept containing sets of currencies. If the user(service provider) wants his service fees in a particular currency unit, an additional knowledge must be added to the service specification. Thus, he must annotate his request (service) in the *CS* with a more specialized concept than **Price**. It can be for example the concept **Yen** if the user wants his service fees in Yen (or the concept **Euro** if the provider can offer his service in Euro).

| **price** = EURO |
|---|
| **Description** = **French** |

**Figure 3:** Typing Constraints

| Context | Hôtel |
|---|---|
| $I$ | Location : String |
| $O$ | Price∗**Price** : Real,<br>Presentation∗**Description** : String |
| $I_{ct}$ | **Price=Euro** |
| $I_{cv}$ | |
| $O_{ct}$ | **Description=French** |
| $O_{cv}$ | |

Figure 4: Our Web services motivating example

Figure 3 shows two typing constraints that a requester/provider can specify in the *CS* corresponding to the *SS* in Figure 1. With such constraints, the requester/provider can offer the necessary details on the values of the service Inputs/Outputs, i.e., in only one measuring unit. Figure 4 is an hotel-informational service corresponding to our motivating example. It is illustrated using our new structure for Web services.

### 4.2 Web Services Matchmaking Process

In our service structure, the Web services matchmaking process works in two steps and consists in determining if the customer's request can be satisfied by the services advertised in a services directory. During the first step, the software module of the directory in charge of the matchmaking process compares syntactically the keywords describing the request triplet $(C, I, O)$ with the triplets $(C', I', O')_s$ of each available service in the directory, and semantically the possibly associated concepts.

A service is considered to be able to answer to a request if its context and Inputs/Outputs are similar to those of the request with a similarity threshold that can be defined within the directory. At the end of this step, a set of services *likely* to be able to satisfy the customer request is selected.

The second step dependents on the success of the matchmaking process during the first step, i.e., the first step must return at least one service in order to pass to the second step of the matchmaking process. This second step consists in the comparison of the *CS* of the request and the *CS* of the selected services in the first step. This step also comprises two stages : initially

the comparison of the Inputs/Outputs typing constraints, then the comparison of their values constraints. We will illustrate in what follows the matchmaking process of Inputs/Outputs typing constraints. The matchmaking of Inputs/Outputs values constraints can be performed by constraints satisfaction algorithms [10].

If all the Inputs/Outputs types of the request and those of a service have similar measuring units, then there is no conflict and the matchmaking process continuous with their Input/Output values constraints. If at least an Input/Output has two different measuring units in the request and in a service, then a typing conflict appears. The typing conflict between a request's Input/Output and a service's Input/Output means that the service cannot answer directly to the request of the customer. However, that does not draw aside definitively this service, because it may happen that the conflict can be solved. We introduce in what follows the definition of a typing conflict after the definition of the Cover Axiom notion.

**Définition 1** ***(Cover axiom)*** *Let $A_1, A_2, ..., A_n$ be a set of formal concepts. A cover axiom is an assertion of the form $A := A_1 \vee A_2 \vee ... \vee A_n$, which means that the concepts $A_1, A_2, \ldots, A_n$ are the all sub-concepts of the concept A.*

The cover axiom represents knowledge allowing the distinction between the concepts being able to cause typing conflicts and the other ones. An axiom is associated with each extensional concept being able to cause a conflict in the domain ontology. In the example of Figure 1, the concept **Price** (equivalent to the concept **Money**) can constitute the head of one cover axiom, because the price of a product can be specified in several different currencies. Thus, we will have the axiom **Money** $:= \textit{EURO} \vee \textit{YEN} \vee \ldots \vee \textit{DOLLAR}$.

**Définition 2** ***(Typing conflict)*** *Let $C$ be a set of concepts described in an ontology, $x, y, z$ three extensional concepts defined in $C$, and the two constraints:*
$x = y$ *(Request typing constraint)*
$x = z$ *(Service typing constraint)*

*If $y \neq z \wedge (\exists\, c \in C \mid y \sqsubseteq c \wedge z \sqsubseteq c)$, and if there is a cover axiom $c := y \vee \ldots \vee z$ then there is a conflict due to the difference between the measuring units of the Input/Output $x$ in the request and in the service.*

When conflicts between a request and a service are detected, the process of retrieving some services able to solve them starts. The semantics of a conflict resolution is the transformation of the Web service configuration available in the directory toward that required by the service requester. It is what we will discuss in the following Section.

### 4.3 Web Services Customization Process

The matchmaking process between a request and a service can reveal several typing conflicts between their Inputs/Outputs. We use the **Web services composition** as a mean for conflicts resolution. In other words, we propose a mechanism for the transformation of the advertised services in order to be compatible with the requirements of the services requesters.

To fulfill the services customization, our approach exploits services able to solve only one conflict. In order to obtain the context (keyword) for the conflict resolution service, we define a set of rules called *Context Association Rules*. Thus, for each domain ontology, a set of rules is defined and stored in the services directory – one rule for each concept that can generate a typing conflict between the various users in the domain.

**Définition 3** *(**Context Association Rule**) A Context Association Rule is a two arguments predicate symbol **ConflictResolution**(Concept, Context). "Concept" is a variable representing extensional concepts defined in a domain ontology and belong to the head of one cover axiom. "Context" is a variable intended to receive the context of the conflict resolution service.*

A typing conflict is induced by the difference between two concepts both subsumed by the same concept appearing in the first argument of one ***ConflictResolution*** predicate. The two concepts in conflict are recovered from the annotation concepts of the same Input/Output in a request and in a service.

**Exemple 2** *Figure 5 illustrates two context association rules in relation to the ontology of Figure 2. The first rule associates the concept **Money** to the keyword (context) of the currency conflict resolution service : ConversionMoney. The second associates the concept **Language** to the keyword of presentation-language conflict resolution service: Translation.*

| **ConflictResolution**(Money, ConversionMoney) |
|---|
| **ConflictResolution**(Language, Translation) |

**Figure 5:** Context Association rules

The context of the service to call in order to solve an Input/Output typing conflict is extracted by exploring the context association rules. Indeed, the predicate *"ConflictResolution"* having as first argument the concept causing the conflict and as second argument a variable indicating the name of the conflict resolution service, will be sent to the set of context association rules. The context of the conflict resolution service is determined by the substitution of the variable *Context* by a keyword (context) appearing in one of the context association rules (see Figure 5). This is done through the terms unification algorithm [15]. The Inputs/Outputs of the conflicts resolution services, are obtained following two cases, whether the conflict relates to an Input or to an Output. When a conflict occurs between the Input of a request and the one of a service $s$, the Input of the conflict resolution service is the Inputs of the service $s$, and its Output is the Inputs of the request. If the conflict is caused by the Output of a request and that of a service s, the Input of the service to be called takes the Outputs of the service $s$, and its Output takes the Outputs of the request.

The service to be called will convert the values of the Inputs/Outputs of the original service in order to make them comparable with those of the request. The values of the Inputs/Outputs to convert will be transmitted to the conflicts resolution services in the service invocation phase. From there, the values of the Inputs/Outputs of the request and that of the service will be comparable (in the same measuring unit). This unables the continuation of the matchmaking process with the verification of the non-contradiction of the Inputs/Outputs values constraints of the request and the service initially in conflict. The Inputs/Outputs values constraints matchmaking makes it possible to select the services, effectively, able to answer the customer request. Because of space constraint, we can't give our Web services customization algorithm in this article. It is detailed in [16] where all its steps are illustrated with the motivating example of Section 2.

## 5 Conclusion and Future Work

In order to be able to provide products in a more and more global market, companies must vary their products according to the customers requirements. To achieve this, they must change their

providing paradigm from products intended for the great mass of customers [20] to customizable products. This new paradigm is called *Mass Customizing Paradigm* [7].

We adopted this paradigm to Web services in order to be able to automatically satisfy the customer requirements in terms of customizable Web services, while avoiding to the services providers the heavy task consisting to build for each customer his own configuration. Our approach, currently under development, will be used for Web services customization in a services directory. Also, it can play the role of a substitution services system if within a combination of services, the Outputs of a service $i$ are not compatible with the Inputs of a service $i + 1$. Thus the services composition will not fail because of a simple incompatibility between two services when one follow the other in a composition.

## Bibliography


1. M. H. Burstein, J. R. Hobbs, O. Lassila, D. L. Martin, D. V. McDermott, S. A. McIlraith, S. Narayanan, M. Paolucci, T. R. Payne, and K. P. Sycara. Daml-s: Web service description for the semantic web. In Horrocks and Hendler [5], pages 348–363.
2. M. R. Genesereth. Knowledge interchange format. In *KR*, pages 599–600, 1991.
3. N. Guarino. A concise presentation of itl. In *PDK*, pages 141–160, 1991.
4. G. Hohpe. *Web services: Pathway to a Service Oriented Architecture*. Thought Works, Inc., 2002.
5. I. Horrocks and J. A. Hendler, editors. *The Semantic Web - ISWC 2002, First International Semantic Web Conference, Sardinia, Italy, June 9-12, 2002, Proceedings*, volume 2342 of *Lecture Notes in Computer Science*. Springer, 2002.
6. I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. Reviewing the design of daml+oil: An ontology language for the semantic web. In *AAAI/IAAI*, pages 792–797, 2002.
7. B. J. P. II and S. Davis. *Mass Customization: The New Frontier in Business Competition*. Harvard Business School Press.
8. J. Kovse, T. Harder, and N. Ritter. Supporting mass customomization by generating adjusted repositories for product configuration. In *CAD*, pages 17–26, 2002.
9. H. J. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. B. Scherl. Golog: A logic programming language for dynamic domains. *J. Log. Program.*, 31(1-3):59–83, 1997.
10. C. Liu and I. T. Foster. A constraint language approach to matchmaking. In *RIDE*, pages 7–14, 2004.
11. R. M. MacGregor. Inside the loom description classifier. *SIGART Bulletin*, 2(3):88–92, 1991.
12. S. A. McIlraith and T. C. Son. Adapting golog for composition of semantic web services. In *KR*, pages 482–496, 2002.
13. S. A. McIlraith, T. C. Son, and H. Zeng. Semantic web services. *IEEE Intelligent Systems*, 16(2):46–53, 2001.
14. OWL. http://www.w3.org/TR/owl-features/.
15. J. A. Robinson. *A machine oriented logic based on the resolution principle*. J.ACM, 12(1):23-41, 1965.
16. Y. SAM. *l'Incohérence dans l'Appariement de Services Web : Détection et Résolution*. Master 2 Recherche. http://www.lsis.org/~samy/master.pdf, 2005.
17. SOAP. http://www.w3.org/TR/soap/.
18. K. P. Sycara, S. Widoff, M. Klusch, and J. Lu. Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace. *Autonomous Agents and Multi-Agent Systems*, 5(2):173–203, 2002.
19. UDDI. http://uddi.org/pubs/uddi_v3.htm.
20. Y. Umeda, A. Nonomura, and T. Tomiyama. Study on life-cycle design for the post mass production paradigm. *AI EDAM*, 14(2):149–161, 2000.
21. WSDL. http://www.w3.org/TR/wsdl/.