Semantic Web Services Composition for the Mass Customization Paradigm

Yacine Sam[1], Omar Boucelma[1] and Mohand-Saïd Hacid[2]

[1] LSIS–CNRS, Université Aix-Marseille 3, Domaine Universitaire de Saint-Jérôme.
Avenue Escadrille Normandie-Niemen, 13397 Marseille Cedex 20, France

[2] Université Claude Bernard Lyon 1.
43, boulevard du 11 Novembre 1918, 69622 Villeurbanne cedex, France

Abstract. In order to fulfill current customers requirements, companies and service providers need to supply a large panel of their products and services. Recently, this situation has led 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 an approach 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.

1 Introduction

The Web is not only an enormous warehouse of text and images, its evolution made it also a services provider [14]. 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. Roughly speaking, Web services are defined as *well defined, loosely coupled software components* and constitute therefore a new paradigm for applications integration [5].

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 [1, 6]. Semantic Web services are then a synthesis of these two standards and constitute 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 [7] is a logic-based language intended to the description of Web services. It is used directly

Sam Y., Boucelma O. and Hacid M. (2006).
Semantic Web Services Composition for the Mass Customization Paradigm.
In *Proceedings of the 1st International Workshop on Technologies for Collaborative Business Process Management*, pages 52-61

or through DAML-S [2]. DAML-S is a DAML+OIL concepts ontology describing the technical aspects of a Web service (Inputs/Outputs parameters, data types, etc). OWL [15], an evolution of DAML+OIL, was recently standardized by the W3C [20]. OWL is now the main standard language for Web ontologies description 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 [10] and LARKS (Language for Advertising 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 [13] for Web services composition.

In this paper, we build on LARKS to elaborate a Web services customization framework. Mass Customization Paradigm [9] is a principle which considers that products must be conceived in such a way that makes it possible to satisfy various needs. Exponential proliferation of services provided through the Web and the cosmopolitan aspect of 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 they are not explicitly present in the services directory. Thus, service providers publish only one explicit configuration of a given service, the others are dynamically and automatically infered from the services directory.

The rest of this paper 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 first propose a new structure for semantic Web services that allows, in contrast to LARKS, customization of services customization. We then describe the matchmaking process, and finally the automatic Web services customization algorithm. We conclude in Section 5.

2 Motivating Example

Each year, ”La Fête de la lumière” (Light Celebrates) is the most important traditional popular celebration in Lyon (a French city). Before traveling to Lyon, a Japanese tourist wants to obtain information about the available Hotels in Lyon and their fees. Thus, he sends his request to a Web directory which stores this information in the form of Web services, expecting to find a Web service execution that can satisfy the request.

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. This scenario shows that the request cannot be satisfied, though the Web service exists in the directory, but in an incompatible configuration.

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 request will then be built by the coordination of these two intermediate services and the service initially available in the directory.

3 Larks Language

LARKS is an advertising and requesting language for Web services [18]. 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 Web services. Ontologies can be described formally with concept languages like ITL [4], LOOM [12] or KIF [3].
- **TextDescription** : textual description of the service requester needs or what a service provider can offer.

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) will fail. Indeed, no knowledge is available to capture the fact that these two concepts can be made comparable (providing a conversion). Consequently, no Web services transformation is possible in this language.

In the following we propose a new Web services structure which constitutes the foundation for the customization process. It allows a service directory to detect, during the matchmaking process, that two concepts (measuring units) can be convertible by another service. Doing so, we avoid immediate failure of the matchmaking process. From now, the term service is used to refer to a service offered by a service provider, and the term request refers to a service requested by a client.

4 Web Services Customization

This section introduces a new semantic Web services specification structure, the matchmaking process associated to it and the Web services customization.

4.1 A New Structure for Web Services

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

The Structural System (*SS*). The *SS* is defined by the triple $(\mathcal{C}, \mathcal{I}, \mathcal{O})$. $\mathcal{C}$ is the context of the specification, it is defined by a keyword related to the specified service. $\mathcal{I}$ and $\mathcal{O}$ 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 triple $(\mathcal{C}, \mathcal{I}, \mathcal{O})$ can be annotated by formal concepts defined in an ontology, which we consider to be shared between all the users of a specific domain.

Example 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.*

$\mathcal{C}$	Book
$\mathcal{I}$	Your-Book:String
$\mathcal{O}$	Price***Price**:Real, Presentation***Description**:String

Fig. 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.

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 service matchmaking that determinates the services *likely* able to satisfy the request. There is a second service matchmaking stage where only the services being, effectively, able to satisfy it will be selected.

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)

Fig. 2. Examples of formal concepts defined in ITL language.

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 typesin the domain ontology (typing constraints). The CS is defined by the quadruplet $(\mathcal{I}_{ct}, \mathcal{O}_{ct}, \mathcal{I}_{cv}, \mathcal{O}_{cv})$ where the elements are sets of constraints on the Input types, Output types, Input values and Output values respectively.

In this paper, we focus on typing constraints that allow to specify the measuring units of Input/Output values in the the specifications of services. 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 issue.

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 her/his service fees in a particular currency unit, an additional knowledge must be added to the service specification. Thus, she/he must annotate the service in the *CS* with a more specialized concept than **Price**. It can be, for example, the concept **Yen** if the user wants the fees in Yen (or the concept **Euro** if the provider can offer the service in Euro).

price = **EURO**
Description = **French**

Fig. 3. Typing Constraints.

Context	Hôtel
$\mathcal{I}$	Location : String
$\mathcal{O}$	Price∗**Price** : Real, Presentation∗**Description** : String
$\mathcal{I}_{ct}$	**Price**=**Euro**
$\mathcal{I}_{cv}$	
$\mathcal{O}_{ct}$	**Description**=**French**
$\mathcal{O}_{cv}$	

Fig. 4. A motivating example in our new structure for Web services.

Figure 3 shows two typing constraints corresponding to the *SS* in Figure 1 that a requester/provider can specify in the *CS*. 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 shows a fragment of a service description corresponding to our motivating example. It is specified using our structure for Web services description.

4.2 Web Services Matchmaking Process

In our service structure, the Web services matchmaking process involves 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 component of the directory in charge of the matchmaking process performs a syntactic comparison of the keywords describing the request triplet $(\mathcal{C}, \mathcal{I}, \mathcal{O})$ with the triplets $(\mathcal{C}', \mathcal{I}', \mathcal{O}')_s$ of each available service in the directory, and performs semantic comparison of the associated concepts.

A service is considered as an 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 : (1) the comparison of the Input/Output typing constraints, and (2) the comparison of their (value) constraints. We will illustrate in what follows the matchmaking process of Inputs/Outputs typing constraints. The matchmaking of Inputs/Outputs value constraints can be performed by constraint satisfaction algorithms [11].

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 (a priori) satisfy the request. However, this does not draw aside this service definitively since it may happen that the conflict can be solved.

Definition 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 all sub-concepts of the concept A.*

The cover axiom represents knowledge allowing the distinction between the concepts being able to lead to 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** $:=$ $E\text{URO} \vee Y\text{EN} \vee \ldots \vee D\text{OLLAR}$.

Definition 2 ***(Typing conflict)*** *Let $\mathcal{C}$ be a set of concepts described in an ontology, x, y, z three extensional concepts defined in $\mathcal{C}$, and the two constraints:*

$x = y$ *(Request typing constraint)*
$x = z$ *(Service typing constraint)*

If $y \neq z \wedge (\exists\, c \in \mathcal{C} \mid y \sqsubseteq c \wedge z \sqsubseteq c)$*, 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 revealed, 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 into the configuration required by the service requester.

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 conflict resolution. In other words, we propose a mechanism that allows to transform advertised services in order to be compatible with the requirements of the service requester.

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.

Definition 3 ***(Context Association Rule)*** *A Context Association Rule is a binary predicate* ***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 (keyword) 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 conflicting concepts are recovered from the annotation concepts of the same Input/Output in a request and in a service.

Example 2 *Figure 5 shows 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.*

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

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

Fig. 5. Context Association rules.

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 [16].

The Inputs/Outputs of the conflict 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 Input 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 the Output 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 passed to the conflicts resolution services in the service invocation phase. Then, the values of the Inputs/Outputs of the request and those of the service are made comparable (based on a same measuring unit). This allows to pursue the matchmaking process with the verification of the non-contradiction of the Input/Output value constraints of the request and the service causing the conflict. The Input/Output value constraints matchmaking makes it possible to select the services able to answer the request. All these matchmaking steps that allow Web services customization by dynamic composition of other services are illustrated in Algorithm 1.

5 Conclusion

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 paradigm from products intended to a large audience of customers to customizable products. This new paradigm is called *Mass Customizing Paradigm* [8]. We adapted this paradigm to Web services in order to provide a framework allowing to automatically satisfy the customer requirements in terms of customizable Web services, while avoiding to the service providers the heavy task of building specific configuration for each customer.

References

1. T. Berners-Lee, J. Hendler, and O. Lassila, editors. *The Semantic Web*. May, 2001.

[3] Concept for the semantic annotation of request Input
[4] Concept for the semantic annotation of service Input
[5] Concept for the semantic annotation of request Output
[6] Concept for the semantic annotation of service Output

2. 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 [6], pages 348–363.
3. M. R. Genesereth. Knowledge interchange format. In *KR*, pages 599–600, 1991.
4. N. Guarino. A concise presentation of itl. In *PDK*, pages 141–160, 1991.
5. G. Hohpe. *Web services: Pathway to a Service Oriented Architecture*. Thought Works, Inc., 2002.
6. 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.
7. 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.
8. B. J. P. II and S. Davis. *Mass Customization: The New Frontier in Business Competition*. Harvard Business School Press.
9. J. Kovse, T. Harder, and N. Ritter. Supporting mass customomization by generating adjusted repositories for product configuration. In *CAD*, pages 17–26, 2002.
10. 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.
11. C. Liu and I. T. Foster. A constraint language approach to matchmaking. In *RIDE*, pages 7–14, 2004.
12. R. M. MacGregor. Inside the loom description classifier. *SIGART Bulletin*, 2(3):88–92, 1991.
13. S. A. McIlraith and T. C. Son. Adapting golog for composition of semantic web services. In *KR*, pages 482–496, 2002.
14. S. A. McIlraith, T. C. Son, and H. Zeng. Semantic web services. *IEEE Intelligent Systems*, 16(2):46–53, 2001.
15. OWL. http://www.w3.org/TR/owl-features/.
16. J. A. Robinson. *A machine oriented logic based on the resolution principle*. J.ACM, 12(1):23-41, 1965.
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. W3C. http://www.w3.org/.
21. WSDL. http://www.w3.org/TR/wsdl/.

Algorithm 1 Web Services Customization.

Step 1 : ***Structural Systems matchmaking***

1. Seek a set $\mathcal{S}$ of services whose context and Inputs/Outputs are similar to those of the request $\mathcal{Q}$.
2. If the set $\mathcal{S}$ is empty then the matchmaking process fails, and no result can be turned over for the request. Else pass to step 2.

Step 2 : ***Constraints Systems matchmaking***

For each service s in the set $\mathcal{S}$ do:

A. Detect the Inputs/Outputs whose types are in conflict with the Inputs/Outputs of the request. If no conflict is detected then pass to step C, else pass to step B.

B. Seek a conflict resolution service for each typing conflict of an Input/Output in a request and a service specifications. Two cases are to be distinguished, according to whether the conflict relates to an Input or to an Output.

a. **If**(ConceptAnnotate(I_Q)[3]=C1
$\wedge$ ConceptAnnotate$(I_s)^4 = C2$ // C1 $\neq$ C2
$\wedge$ $(\exists C \mid C1 \sqsubseteq C \wedge C2 \sqsubseteq C)$// C: immediate subsumer of C1 and C2
$\wedge$ $C := C_1 \vee C_2 \vee \ldots$) // Cover Axiom
then Find the service whose structure is :

Context	ConflictResolution(C, context)
$\mathcal{I}$	$\mathcal{I}_s$
$\mathcal{O}$	$\mathcal{I}_Q$
$\mathcal{I}_{ct}$	$\mathcal{I}_{ct(s)}$
$\mathcal{O}_{ct}$	$\mathcal{I}_{ct(Q)}$

b. **If**(ConceptAnnotate(O_Q)[5] = C1
$\wedge$ ConceptAnnotate$(O_s)^6 = C2$ // C1 $\neq$ C2
$\wedge$ $(\exists C \mid C1 \sqsubseteq C \wedge C2 \sqsubseteq C)$// C: immediate subsumer of C1 and C2
$\wedge$ $C := C_1 \vee C_2 \vee \ldots$) // Cover Axiom
Then Find the service whose structure is :

Context	ConflictResolution(C, context)
$\mathcal{I}$	$\mathcal{O}_s$
$\mathcal{O}$	$\mathcal{O}_Q$
$\mathcal{I}_{ct}$	$\mathcal{O}_{ct(s)}$
$\mathcal{O}_{ct}$	$\mathcal{O}_{ct(Q)}$

C. Evaluate the Inputs/Outputs values constraints. If they are not in contradiction then the service can answer the request, else the service is not suitable like response to the request.