Chapter 4
Horizontal Service Composition for Language Services

Ahlem Ben Hassine¹, Shigeo Matsubara², and Toru Ishida²

¹ National School of Computer Science (ENSI), Tunis University, Tunisia, e-mail: ahlembh@gmail.com
² Department of Social Informatics, Kyoto University, Yoshida-honmachi, Sakyo-ku, Kyoto 606-8501, Japan, e-mail: {matsubara, ishida}@i.kyoto-u.ac.jp

Abstract In the Language Grid, automatically composing Web services is a crucial task. This task involves vertical and horizontal composition. Vertical composition consists of defining an appropriate combination of simple processes to perform a composition task. Horizontal composition consists of determining the most appropriate Web service from among a set of functionally equivalent ones for each component process. The latter is important in language services. For the horizontal composition of Web services, we propose a generic formalization of any Web service composition problem based on a constraint optimization problem (COP) and then propose an incremental user-intervention-based protocol to find the optimal composite Web service according to some predefined criteria at run-time.

4.1 Introduction

The Language Grid provides users with functions to combine language resources (e.g., bilingual dictionaries) or language processing functions (e.g., machine translators) and to add their own language resources to create new language services for their own intercultural activities (Ishida 2006). That is, combining a variety of language services allows users to make better use of the large quantity of language resources that have accumulated on the Internet. It will enable a language service to be built that is optimal for the actual field of activity performed by the intercultural collaboration.

Consider a specialized translation service with back translations. This service can be achieved by using a composite service. Several atomic services, such as machine translations, morphological analyzers, and specialized dictionaries can be combined to create the specialized translation service. However, this composition task might be difficult to realize because so many services with the same or similar ability exist. To overcome this difficulty and properly support users, we have developed a constraint-based Web service composition technique.
Our technique is based on the technologies of Web services. The great success of Web services, due especially to their rich applications made possible by open common standards, has led to their wide proliferation and a tremendous variety of Web services are now available. However, this proliferation has rendered the discovery and use of the most appropriate Web service arduous. These tasks are increasingly complicated, especially if the target is a composite Web service that must satisfy a user’s long-term complex goal. The automatic Web service composition task consists of finding an appropriate combination of existing Web services to achieve a global goal.

We focus on the fact that many available Web services can fulfill the same task and we refer to these Web services as functionally equivalent Web services. In the sequel of this chapter, as is generally done in the literature, we refer to each of the subtasks making up the main goal as an abstract Web service and to each Web service able to perform a subtask as a concrete Web service. Solving the Web service composition problem involves two types of composition:

- **Vertical** composition is aimed at finding the “best” combination of abstract Web services, i.e., abstract workflow, in terms of achieving the main goal while satisfying all existing interdependent restrictions.

- **Horizontal** composition is aimed at finding the “best” concrete Web service, from among a set of available functionally equivalent Web services, i.e., executable workflow, to perform each abstract Web service. The quality of the response to the user’s query (the composition task) strongly depends on the selected concrete Web services. The choice of a concrete Web service is dictated by functional (i.e., related to the inputs) and/or non-functional attributes (i.e., related to the quality of service attributes).

The main benefits gained by differentiating these two composition processes are: i) the Web service composition problem is simplified with reduced computational complexity, ii) avoiding any horizontal composition redundancy that may appear while searching for the “best” combination of abstract Web services, and mainly iii) ensuring more flexibility for user intervention, i.e., the user is able to modify/adjust the abstract workflow when needed.

This chapter consists of two main parts. The first is a generic formalization of any Web service composition problem as a constraint optimization problem (COP) in which we try to express most of the Web service composition problem features in a simple and natural way. Our main purpose is to develop a common and robust means of expressing any Web service composition problem that ideally reflects realistic domains. The second contribution is a real-time interactive protocol to solve any Web service composition problem by overcoming most of the limitations encountered above. Although there are various techniques for solving COPs, none of these consider the user interaction issue. The constraint optimization problem formalism is especially promising for ideally describing any realistic Web service composition problem, because this problem is a combinatorial problem that can be represented by a set of variables connected by constraints. Two approaches are proposed in this chapter, a centralized approach and a distributed approach.
This chapter is organized as follows. In Section 4.2, we explain why horizontal composition is needed in the language services domain. In Section 4.3, we present the proposed formalization. In Section 4.4, we describe a real-world scenario. In Section 4.5, we describe the proposed algorithm. In Section 4.6, we discuss possibilities of an extension of the previous algorithm. In Section 4.7, we compare the proposed techniques to the existing techniques. In Section 4.8, we conclude the chapter.

### 4.2 Why Horizontal Composition?

The language services domain has different characteristics from other domains such as supply-chain management. In the language services domain, the number of services included in a composite service is at most six or seven. On the other hand, a lot of functionally equivalent services exist that can be used to realize a subtask. For example, more than one hundred parallel dictionaries are available. This suggests that the challenge is not how to find the “best” combination of the abstract Web services but how to find the “best” concrete Web service from among a set of available functionally equivalent Web services.

Fig. 4.1 shows an example of vertical composition and horizontal composition. In vertical composition, a task of tailored-translation is given. First, the task is decomposed into the two subtasks of looking-up-dictionary and tailored-machine-translation and then a sequence of services accomplishing each subtask is searched for. The result is the best combination of the abstract Web services such as morphological-analysis, technical-term-extraction, technical-term-bilingual-dictionary, term-replacement, and machine-translation. Here, note that each service is an abstract Web service, that is, it is not bound to any concrete service.

In horizontal composition, on the other hand, a workflow of abstract Web services is given and the goal is to find a best combination of concrete Web services. For example, an abstract service of morphological-analysis can be bound to a concrete Web service such as LX-Suite, POSTAGE/K, FreeLing, or HAM.

Solving the horizontal Web service composition problem is not easy. The method of selecting the best service for each subtask and combining them to form a composite service does not work well because it does not guarantee to satisfy the constraints such as the user’s budget constraint. The task of combining Web services has attracted the interest of many researchers, (McIlraith and Son 2002), (Sirin et al. 2004), (Lin et al. 2005), and several approaches have been reported. Most of these deal only with vertical composition, where only a single concrete Web service is available for each abstract one. Thus, their techniques cannot be applied to horizontal Web service composition.
To solve the problem of mixing and matching component Web services we have to consider various features. These features can be divided into two main groups:

- Features related to the user, including the user’s constraints and preferences. For example, the user prefers J-Server over Google Translate as a machine translation service, while another user has the reverse preference.
- Features related to Web services; these can be divided into two subgroups, internal and external features. Internal features include quality of service (QoS) attributes, and external features include existing restrictions on the connection of Web services, (e.g., a hotel room should be reserved for the ISWC conference usually after booking the flight). External features are specified in the Web service ontology language, OWL-S (OWL Services Coalition 2003), through a set of control constructs such as Sequence, Unordered, Choice, etc.

As mentioned above, the tremendous number of functionally equivalent concrete Web services makes the search for an appropriate one, i.e., horizontal composition of concrete Web services, an NP-hard task (Canfora et al. 2005). This composition process also has the following characteristics:

- Information is often incomplete and uncertain.
- The environment is naturally distributed and dynamic.
- Many (non)-functional features, inter-related restrictions and especially user preferences may affect the quality of the response to a user’s query.

Existing research efforts have tackled only some parts of the natural features of the Web service composition problem (Au et al. 2005), (Kuter et al. 2004), none
have tried to deal with all of them. Moreover, some complex real-world problems require some level of abstract interactions with the user to refine the search for a valid composite Web service. Finally, very few studies have considered the validity of the information concerning a concrete Web service during the composition process and none have dealt with this question of validity during the execution process. We have learned from all these works and focused our research on the requirements of the Web service composition problem that are derived from the natural features of the problem, search-based user intervention and the information validity during the composition and execution processes. Our main goal is to provide a means by which an optimal composite executable workflow can be created for a given set of sub-tasks with their inter-relation restrictions, i.e., an abstract workflow.

4.3 Constraint-based Formalization of Horizontal Web Service Composition

The constraint satisfaction problem (CSP) framework is a key formalism for many combinatorial problems. The great success of this paradigm is due to its simplicity, its natural expressiveness of several real-world applications, and especially the efficiency of the existing underlying solvers. We therefore believe that the CSP formalism allows a better and more generic representation of any Web service composition problem. Hence, we formalize the Web service composition problem as a constraint optimization problem (COP) in which we have two kinds of constraints: hard and soft constraints.

A static CSP is a triplet \((X, D, C)\) composed of a finite set \(X\) of \(n\) variables, each of which takes a value in an associated finite domain \(D\) and a set \(C\) of \(e\) constraints between these \(n\) variables (Montanari 1974). Solving a CSP consists of finding one or all complete assignments of values to variables that satisfy all the constraints. This formalism was extended to the COP to deal with applications where we need to optimize an objective function. A constraint optimization problem is a CSP that includes an objective function. The goal is to choose values for variables such that the given objective function is minimized or maximized.

We define a Web service composition problem as a COP by \((X, D, C, f(sl))\) where:

- \(X = \{X_1, \ldots, X_n\}\) is the set of abstract Web services, each \(X_i\) being a complex variable represented by a pair \((X_i.in, X_i.out)\) where
  - \(X_i.in = \{in_{i1}, in_{i2}, \ldots, in_{ip}\}\) represents the set of \(p\) inputs of the concrete Web service, and
  - \(X_i.out = \{out_{i1}, out_{i2}, \ldots, out_{iq}\}\) represents the set of \(q\) outputs of the concrete Web service.
• \( D = \{ D_1, \ldots, D_n \} \) is the set of domains, each \( D_i \) representing possible concrete Web services that fulfill the task of the corresponding abstract Web service. \( D_i = \{ s_j | s_j.\text{in} \subseteq X_i.\text{in} \text{ AND } X_i.\text{out} \subseteq s_j.\text{out} \} \)

• \( C = C_S \cup C_H \)
  - \( C_S \) represents the soft constraints related to the preferences of the user and to some Quality of Service attributes. For each soft constraint \( C_S \in C_S \) we assign the penalty \( \rho_{C_S} \in [0, 1] \). This penalty reflects the degree to which the soft constraint \( C_S \) is not satisfied.
  - \( C_H \) represents the hard constraints related to the inter-abstract Web services relations, the OWL-S defined control constructs, and the preconditions of each concrete Web service. For each hard constraint, \( C_H \in C_H \), we assign a weight \( \perp \) (i.e. satisfaction is an imperative). It is noteworthy that \( C_H \) may also include some hard constraints specified by the user. These hard constraints can be relaxed upon request if no solution to the problem is found.

• For each concrete Web service we assign a weight to express the degree of user preference, \( w_{s_j} \in [0,1] \). Weights are automatically accorded to the values of variables in a dynamic way with respect to the goal.

• \( f(sl) \) is the objective function to optimize, \( f(sl) = \otimes_{s_j \in sl} \) (user’s preferences, penalty over soft constraints, Quality of Service attributes, probability of information expiration), and \( sl \) is a solution of the problem defined by the instantiation of all the variables of the problem. In this work, we focus on optimizing both \( i) \) the user’s preferences toward selected concrete Web services denoted by \( \phi(sl) \) and \( ii) \) the penalty over soft constraints denoted by \( \psi(sl) \). The Quality of Service attributes and the probability of information expiration will be tackled in our future work.

Solving a Web service composition problem consists of finding a “good” assignment \( sl^* \in Sol := D_1 \times \ldots \times D_n \) of the variables in \( X \) such that all the hard constraints are satisfied while objective function \( f(sl) \) is optimized according to Eq. 4.1.

\[
f(sl^*) = \arg\max_{sl \in Sol} \otimes(\phi(sl), \psi(sl))
\]  

In this chapter, we maximize the summation of the user preferences for all concrete Web services involved in solution \( sl \) and minimize the summation of the penalties associated to all soft constraints according to Eq. 4.2.

\[
f(sl^*) = \arg\max_{sl \in Sol} (\sum_{s_j \in sl} w_{s_j} - \sum_{C_S \in C_S} \rho_{C_S})
\]  

Since the solution might be more than a sequence of concrete Web services, i.e., it may include concurrent concrete Web services, we use “,” to indicate sequential execution and “||” to indicate concurrent execution. This information is useful in the execution process. The obtained solution will have a structure such as
This problem is considered to be a dynamic problem since the set of abstract Web services (the set of variables) is not fixed; i.e., an abstract Web service can be divided into other abstract Web services if no available concrete Web service can perform the required task. In addition, the set of values in the domain of each variable (the set of possible concrete Web services) is not fixed. Concrete Web services can be added to/removed from the system.

In the Web service composition problem, several control constructs connecting Web services can be used. The main ones, defined in the OWL-S description, can be divided into four groups and we describe our formalization of these four groups below.

- **Ordered**, which involves the SEQUENCE control construct, can be expressed by using a hard constraint. Each pair of abstract Web services linked by a sequence control construct are involved in the same $C_{\text{Sequence}}$ constraint.

- **Concurrency** involves the SPLIT, SPLIT+JOIN, and UNORDERED control constructs. The inherent aspect of the following proposed agent-based approach (Section 4.5) allows the formalization of this control construct in a natural way. Note that only “JOIN” will be associated with a $C_{\text{Join}}$ constraint. SPLIT and UNORDERED will be modeled using an “empty” constraint, $C_{\text{empty}}$, that represents a universal constraint. This constraint will be used to propagate information about parallel execution to concerned variables in the following proposed protocol.

- **Choice** involves IF-THEN-ELSE and CHOICE control constructs. For each set of abstract Web services (two or more) related by the IF-THEN-ELSE or CHOICE control construct, the corresponding variables are merged into the same global variable ($X_j$ for example), and their domains are combined and ranked according to the preference of the user. For example, given a set of $m$ abstract Web services ($\{t_1, t_2, ..., t_m\}$) related by the “CHOICE” control construct, we combine them into a global variable ($X_i$ for example) and rank their domains. For their preconditions, we assign a sub-constraint to each condition $\{C_{\text{cond}1}, C_{\text{cond}2}, ..., C_{\text{cond}m}\}$ and create a global constraint $C_{\text{Choice}} = \bigcup_i C_{\text{cond}i}$. At any time we are sure that only one condition will be satisfied since $\cap_i C_{\text{cond}i} = \emptyset$.

- **LOOP**, neither the CSP formalism nor any of its extensions can handle iterative processing. This will be considered in our future work.

### 4.4 Real-world Scenario

The main objective of the Language Grid project (Ishida 2006) is to enhance intercultural collaboration by increasing the accessibility and usability of existing language resources on the Web. Murakami et al. (Murakami et al. 2006) proposed an abstract workflow for a tailored translation, see Fig.4.2. In the following, we present our proposed formalization of this workflow.
\[ X = \{X_1, X_2, X_3, X_4, X_5, X_6\} \], where each \( X_i = (X_i.in, X_i.out) \) corresponds to one of the atomic services.

- \( X_1 \): corresponds to the atomic service of morphological analysis; \( X_1.in = \{\text{originalSentence, sourceLang}\}; X_1.out = \{\text{morphemes}\} \)
- \( X_2 \): corresponds to the atomic service of technical term extraction service; \( X_2.in = \{\text{morphemes}\}; X_2.out = \{\text{technicalTerms}\} \)
- \( X_3 \): corresponds to the atomic service of technical term bilingual dictionary; \( X_3.in = \{\text{technicalTerms}\}; X_3.out = \{\text{technicalTermTranslated, technicalTermsIntermediateCode}\} \)
- \( X_4 \): corresponds to the atomic service of term replacement service; \( X_4.in = \{\text{originalSentence, technicalTermTranslated, technicalTermsIntermediateCode}\}; X_4.out = \{\text{intermediateCodeSentence}\} \)
- \( X_5 \): corresponds to the atomic service of machine translation service; \( X_5.in = \{\text{intermediateCodeSentence}\}; X_5.out = \{\text{intermediateCodeSentenceTranslated}\} \)
- \( X_6 \): corresponds to the atomic service of term replacement service; \( X_6.in = \{\text{intermediateCodeSentenceTranslated, technicalTermsIntermediateCode, technicalTermsTranslated}\}; X_6.out = \{\text{originalSentenceTranslated}\} \)

\[ D = \{D_1, D_2, D_3, D_4, D_5, D_6\} \], where

- \( D_1 = \{\text{LX-Suite, POSTAGE/K, FreeLing, TreeTagger, Morpha, HAM}\} \)
- \( D_2 = \{\text{CaboCha}\} \)
- \( D_3 = \{\text{Sztaki szotar, CambridgeDict, ENAMDICT, UrduWord, KamusJot}\} \)
- \( D_4 = \{\text{Kura}\} \)
- \( D_5 = \{\text{Kataku, SYSTRAN5.0, Reverso, Free Translation, J-Server, PeTra, Web-Transer}\} \)
- \( D_6 = \{\text{Kura}\} \)

\[ C = C_S \cup C_H \], where

- \( C_S \) including
  \( X_1.\text{originalSentence} \neq \text{nil}; \quad X_2.\text{morphemes} = X_1.\text{morphems}; \quad X_3.\text{technicalTerms} = X_2.\text{technicalTerms} \)

- \( C_S \) including
  \( \text{Acc}(X_5) \geq 0.7 \) with \( \rho_{C_S} = 0.6 \), where the function of \( \text{Acc} \) returns the Web service result accuracy;
  \( \text{Cost}(X_3) + \text{Cost}(X_4) \leq 1 \) cent with \( \rho_{C_S} = 0.3 \), where the function of \( \text{Cost} \) returns the Web service cost.

- The main objective is to find the best combination, \( sl \), of the above abstract Web services and assign the most appropriate concrete Web services such that \( sl \) maximizes objective function \( f(sl) \) defined in Section 4.3 Eq. 4.2. Note that, for simplicity, we assume pairwise independence between the values of the different domains. We will consider dependence issues in future work.
4.5 Constraint Optimization Problem Interactive Algorithm for Solving the Web Service Composition Problem

The overall objective of our approach is to generate the best executable workflow (according to the aforementioned criteria) within a feasible time. Several constraint optimization problem algorithms can be applied to solve this problem, but none allows the intervention of the human user during the search process. In the following, we propose an algorithm (Algorithm 4.1) that allows human interaction with the system to enhance the solving process.
For each variable $X_j$ we first determine a set of candidate concrete Web services, $\text{Cand}_{X_j}$, for its abstract Web service that satisfies all the hard constraints, $C_{\text{Hil}} \in C_{\text{Hil}}$ (Algorithm 4.1 line 4), and then we rank $\text{Cand}_{X_j}$ according to the objective function defined in Section 4.3. This ranked set is used to guide the selection of the next variable $X_{j+1}$ in the search process. For $X_{j+1}$ we start by applying the join operation to the received list $\text{Cand}_{X_j}$ and the current one $\text{Cand}_{X_{j+1}}$, i.e., join of $\text{Cand}_{X_j}$ and $\text{Cand}_{X_{j+1}}$ (Algorithm 4.1 line 12). The obtained sub-solutions are then filtered (Algorithm 4.1 line 12) according to the set of existing hard constraints. Finally, the resulting set of sub-solutions is ranked according to the objective function for optimization. If the set of candidates $\text{Cand}_{X_j}$ is large, to avoid explosion in the join operation, we select a fixed number of the most preferred concrete Web services for each variable, (i.e., a subset of candidates), and try to propagate these to the next variable. Whenever this subset does not lead to a complete solution, we backtrack and then seek a solution using the remaining candidates. The order of the values in the candidate set is established to avoid missing any solution. The obtained sets of sub-solutions are propagated to the next variable (Algorithm 4.1 line 16) and the same dynamic continues until all the abstract Web services are instantiated. If the set of candidate Web services becomes empty (i.e., none of the available Web services satisfies the hard constraints), or the set of sub-solutions resulting from the join and filter operations becomes empty and no more backtracking can be performed, the user is asked to relax some of his/her constraints (Algorithm 4.1 line 23). However, if the user’s relaxed constraints involve the first instantiated variable in the search tree then the search process is reentered from scratch. It is noteworthy that three issues are possible in this algorithm, i) ask for user intervention whenever a local failure is detected, which may reduce the number of backtracks, ii) ask for user intervention only when a global failure is detected, no more backtracks can be performed, iii) store a trace of the explored search tree to be able to point directly to the variable involved in user relaxation and pursue the solving process and avoid some computational redundancy.

In addition, whenever we need any information concerning any concrete Web services, a request-message is sent to an information-providing Web service to get the necessary information along with both its validity duration and the maximum time required to execute the underlying Web service. The agent should maintain this time so that it can detect information expiration and perform the right decision (Algorithm 4.1 line 20). To deal with the main characteristic of this real-world problem, the dynamic environment, we maintain the validity of necessary information during the solving and execution processes, $\text{totalTime}$. $\text{totalTime}$ should be less than the minimum validity time required for any Web service information. We use the following notations:

- $T_{\text{plan}}(s_l)$: time needed to provide plan $s_l$,
- $t_{\text{exe}}(s)$: time needed to execute one concrete Web service,
- $t_{\text{val}}(\text{inf}_j)$: estimated time before the expiration of solicited information $\text{inf}_j$. 
Naturally, the validity of information is usually considered as uncertain. Hence, for each validity time a probability of information alteration $p_{alt}(inf)$ can be associated with underlying information $inf$. We will consider this probability of information alteration in our future work. The maximal time required to provide a solution, $T_{plan}$, is defined by Eq. 4.3.

$$T_{plan}(st) < \min_{s_j \in st} t_{val}(inf_j) - \sum_{s_j \in st} t_{exe}(s_j)$$  \hspace{1cm} (4.3)

Algorithm 4.1 User-intervention-based algorithm for Web service composition.

$WSCSolver(i, setSubSol, totalTime, checkedValues)$

1: if $i>|X|$ then
2: return setSubSol;
3: end if
4: $Cand_i[i] \leftarrow \{s_i \in D_i \mid s_i \text{satisfies all the } C_{ih}\} \setminus$ checkedValues[$i$];
5: if information required for any $s_i \in Cand_i[i]$ then
6: Collect necessary information; update $t_{val}$, $t_{exe}$ and $totalTime$;
7: end if
8: Rank $Cand_i[i]$ according to $w_{stj}$ and $\rho_{cst}$ and while checking $t_{val}$, $t_{exe}$ and $totalTime$;
9: subSol $\leftarrow \emptyset$;
10: while subSol $= \emptyset$ do
11: subCand $\leftarrow$ subset of the $Cand_i[i]$; add(checkedValues[$i$], subCand);
12: subSol $\leftarrow$ join of setSubSol and subCand; Filter and Rank subSol according to $f$ (subSol);
13: end while
14: if subSol $\neq \emptyset$ then
15: add(setSubSol, subSol);
16: return $WSCSolver(i+1, setSubSol, totalTime, checkedValues)$;
17: else
18: if $i>1$ then
19: reset to $\emptyset$ all checkedValues[$j$] for $j>i$;
20: Update $totalTime$; Update setSubSol;
21: return $WSCSolver(i-1, setSubSol, totalTime, checkedValues)$;
22: else
23: $RelaxedConst \leftarrow$ ask User to relax constraints involving $X_k$ where $k<i$;
24: Update($C_{ih}$, $C_{ih}.RelaxedConst$);
25: $i \leftarrow$ such that $\forall X_j$ involved in $C_i$ and $C_i \in RelaxedConst$, $X_j$ precedes $X_i$;
26: Update setSubSol;
27: return $WSCSolver(i+1, setSubSol, totalTime, checkedValues)$;
28: end if
29: end if

Each sub-solution based on expired information will be temporarily discarded but kept for use in case the agent cannot find any possible solution. This mea-
surement is an efficient way to cope with Web services with effects characterized by information volatility because it allows for the forward estimation of the validity of information during both the composition process and the execution process.

4.6 Extended Algorithm

The main limitation of the previous algorithm is that it cannot be easily adapted to any alteration in the environment. Whenever a user decides to relax some of his/her constraints, and these constraints involve an already invoked variable, especially the first one in the search tree, the search for a solution must be recommenced from scratch. However, distributed approaches well support user intervention. In this solution, the same algorithm will be split among a set of homogeneous entities. Each entity will be responsible for one variable and the same algorithm will be performed in parallel by all entities. In case of conflict, i.e., no solution can be generated and no backtrack can be performed, the system will ask the user to relax some constraints. The concerned entity will update its view, generate new candidates and exchange them with the concerned entities. The process repeats until either a solution for the problem is generated or the lack of a solution is confirmed, even with all possible relaxations. Nevertheless, this distributed solution might be inefficient for some real-world scenarios that demand access to a specialized Web service. A specialized Web service maintains information about a set of Web services. Information on the involved Web services is considered private, which makes it difficult to gather and process at the same site the information needed on Web services. Hence, we believe that extending the above algorithm to a multi-agent system is more effective for realistic domains.

4.7 Discussion

In this chapter, we propose a generic formalization for a horizontal Web service composition problem based on a constraint optimization problem (COP) and then propose an incremental user-intervention-based protocol to find the optimal composite Web service according to some predefined criteria at run-time. We clarify our contributions by giving an overview of existing research.

Several solutions to the Web service composition problem have been reported including integer programming (IP)-based techniques (Aggarwal et al. 2004), (Zeng et al. 2004), non-classical planning-based techniques and logic-based techniques (McIlraith and Son 2002), (Narayanan and McIlraith 2002). Recently, some researchers have suggested applying existing artificial intelligence (AI) optimization techniques, such as genetic algorithms (GA), mainly to include some Quality of Service attributes in the search process. Regarding IP-based proposed solutions (Aggarwal et al. 2004), (Zeng et al. 2004), the authors assume linearity
of the constraints and of the objective function. As for non-classical planning techniques, Sirin et al. proposed an HTN-planning based approach (Sirin et al. 2004) to solve this problem. Their efforts were directed toward encoding the OWL-S Web service description as a SHOP2 planning problem, so that SHOP2 can be used to automatically generate a composite Web service. As mentioned above, these studies implicitly assume that only a single concrete Web service is available for each abstract one. In addition, we believe that these planning techniques are difficult to scale up compared to constraint-based techniques.

McIlraith and Son (McIlraith and Son 2002) proposed an approach to building agent technology based on the notion of generic procedures and customizing user constraints. The authors claim that an augmented version of the logic programming language Golog provides a natural formalism for automatically composing services on the semantic Web. They suggested that this problem should not be considered as simple planning, but rather as the customization of reusable, high level generic procedures. These logic-based techniques also do not consider horizontal Web service composition and are difficult to scale up if a lot of functionally equivalent services exist.

Canfora et al. in (Canfora et al. 2005) proposed tackling the QoS-aware composition problem with Genetic Algorithms (GA). This approach tackles both vertical and horizontal compositions. However, to accomplish the Web service composition task, the Web service composition procedure may need to retrieve information from Web services while operating. Most studies have assumed that such information is static (McIlraith and Son 2002), (Sirin et al. 2004), (Canfora et al. 2005). The advantage of our proposal, especially the distributed algorithm for horizontal Web service composition problems, is that it complies with inherent characteristics of real-world problems such as the dynamism of the environment.

Other studies have assumed that an interactive process with the user can get all the necessary information as inputs. Nevertheless, the static information assumption is not always valid, and the information on various Web services may change, i.e., it may be “volatile information” (Au et al. 2005) either while the Web service composition procedure is operating or during execution of the composition process. Kuter et al. (Kuter et al. 2004) present an extension of earlier non-classical planning-based research efforts to better cope with volatile information. This arises when the information-providing Web services do not return the needed information immediately after it is requested (or not at all). In addition, Au et al. (Au et al. 2005) proposed two different approaches for translating static information into volatile information. They propose assigning a validity duration to each item of information received from information providing services. Our proposal also complies with the need to deal with such volatile and uncertain information during the composition and execution processes.
4.8 Conclusion

In the Language Grid, automatically composing Web services is a crucial task. The Web service composition problem is a challenging research issue because of the tremendous growth in the number of Web services available, the dynamic environment, and the fluidity of user needs. In this chapter, we have proposed a real-time interactive solution for the Web service composition problem. This problem involves vertical composition and horizontal composition, and we have focused on the latter. This work complements existing techniques dealing with vertical composition in that it exploits their abstract workflows to determine the best executable one according to predefined optimality criteria. We have developed a protocol that overcomes the published limitations of the existing work and complies with the inherent characteristics of real-world Web service composition problems such as the dynamism of the environment and the need to deal with volatile information during the composition and execution processes. Two main approaches were proposed in this chapter, the first is a user-intervention-based centralized approach and the second is a distributed version of the previous one that can easily handle any alteration in the environment.

Acknowledgments
This research was partially supported by National Institute of Information and Communications Technology (NICT) and a Grant-in-Aid for Scientific Research (A) (21240014, 2009-2011) from Japan Society for the Promotion of Science (JSPS).

References
