Journal of Web Engineering, Vol. 14, No. 1&2 (2015) 063–079

MULTILEVEL ANALYSIS FOR AGENT-BASED SERVICE COMPOSITION[a]

ARIF BRAMANTORO
College of Computer and Information Sciences, Al Imam Mohammad Ibn Saud Islamic University (IMSIU)
Riyadh, Saudi Arabia
arifbramantoro@ccis.imamu.edu.sa

AHLEM BEN HASSINE
National School of Computer Science (ENSI), Tunis University
Tunisia
ahlembh@gmail.com

SHIGEO MATSUBARA
Department of Social Informatics, Kyoto University
Yoshida-Honmachi, Sakyo-ku, Kyoto 606-8501, Japan
matsubara@i.kyoto-u.ac.jp

TORU ISHIDA
Department of Social Informatics, Kyoto University
Yoshida-Honmachi, Sakyo-ku, Kyoto 606-8501, Japan
ishida@i.kyoto-u.ac.jp

Received January 22, 2014
Revised September 5, 2014

Agent-based Web service composition has become one of the most challenging research issues. Many composition techniques and formalizations have been proposed, but they are neither mature nor flexible. They assume that each sub-task is an atomic process, hence it cannot be decomposed based on user requirements. Moreover, those techniques and formalizations are not suitable for dynamic environments such as the language service domain. Language service requires a flexible formalization to accommodate the user's language skills in conjunction with QoS. The key contributions of this paper are (*i*) a complete formalization that ideally reflects the nature of real applications and permits extension of the original abstract workflow (in case of failure); (*ii*) a novel agent-based protocol able to find satisfying solutions for this problem in real time to allow restriction and/or relaxation within the original workflow; (*iii*) a hybrid architecture of service-oriented computing and multi-agent systems for implementing Abstract Web service, Information analysis, and User agents. Experiments are presented to find solutions that can be executed within a feasible time and space.

Keywords: Web service composition, constraint optimization, multi-agent

Communicated by: D. Schwabe & O. Pastora

[a]Some parts of this work were done while the two authors were at Kyoto University and National Institute of Information and Communications Technology (NICT), Kyoto, Japan.

1 Introduction

Web service composition has become a complicated task due to the wide proliferation of the World Wide Web (WWW) and the consequent emergence of a large number of functionally equivalent Web services. Thus, determining and using an appropriate Web service is very hard, especially when dealing with composite Web services to suit a user's preferences and skills. Solving this problem involves features from the service provider such as service profile description and quality of service (QoS), as well as certain features related to the user, such as user's preferences and skills.

Current techniques in Web service composition (*i*) deal only with the agglomeration of abstract Web services (i.e. finding the best abstract workflow) despite the growing number of functionally equivalent Web services making the composition of concrete Web services an NP-hard task as is detailed in [1]; (*ii*) try to find the first feasible solution, even though better ones can be found as shown in [2]; (*iii*) do not deal with many crucial and natural features of Web services, such as the dynamic and distributed environment of Web service information during the composition and execution processes as evidenced in [3]; (*iv*) are not focused on the user's interaction with the services, which involves human skills and preferences as is elucidated in [4, 5].

Our main objective is to build an agent-based system to streamline the generation of a multilevel workflow according to some analyzed criteria. Previous formalizations of Web service composition assume, however, that each subtask is an atomic process and cannot be analyzed for decomposition. We found these shortcomings when we tried to implement the formalizations in a sophisticated domain such as the language service. For example, assume that we have an abstract workflow (built manually or taken from a workflow repository that stores the best practices) including translation service. This workflow cannot be directly concretized when dealing with the Indonesian-Japanese pair of languages, so the translation task should be decomposed into two sub-tasks, such as translating from Indonesian to English and then translating from English to Japanese. In this paper, we propose:

- A multilevel analysis formalization that enables decomposition (to handle failure) of tasks/subtasks using workflow control constructs to guarantee interoperability among selected services.

- A user-centered, agent-based protocol able to find optimal solutions to a Web service composition problem in real time, while allowing restriction and/or relaxation within original workflow.

- An architecture that incorporates service-oriented computing and multi-agent system enabling the implementation of abstract Web service agents, information analysis agents and a user agent.

The rest of this paper is organized as follows. Section 2 provides related work. Section 3 presents a multilevel analysis for service composition. Section 4 describes the formalization into the user-centered environment of the language service domain. Section 5 describes the proposed architecture and algorithm for service composition. Section 6 discusses our experiments and their results, and Section 7 concludes the paper.

2 Related Work

Web service composition has attracted the interest of many researchers. The most recent comparative study on existing techniques in Web service composition is presented by Mahboobeh and Davis [6]. Although there is no performance evaluation of techniques in their study, it is interesting to note that the authors assign the techniques into two categories, optimization-based and automatic negotiation-based. We prefer not to get involved in the debate of this categorization. We believe that the agent architecture makes it possible to combine optimization-based and automatic negotiation-based techniques. Hence, we focus our literature review on the adoption of multilevel analysis, user preference, and agent architecture to solve the problem of Web service composition.

The initial work on Web service composition is performed by Zeng and others [2] who used integer programming to compose services. Their proposed functions for five QoS attributes are our main reference in calculating QoS attributes from network domain, although there are domain-specific attributes that we have to deal with. Their approach on two service selections, local optimization and global composition, is similar to our multilevel analysis of service composition. However, we prefer to use optimization techniques in all level of composition because we need to get the best candidates at each level.

A similar model of multilevel service composition is proposed in [7]. The model is different from our framework, since they propose four levels in multilevel composition (syntactic, static semantic, dynamic semantic, and qualitative composability), not the levels determined by the workflow abstraction as is commonly encountered in real applications. In addition, an automatic composition technique to solve the model remains undefined as future work.

Canfora and others [1] propose a flexible binding in service composition when there is a gap between the initial QoS estimation and the run time realization. An aggregation technique during composition was clearly presented along with two case studies of travel planner and image processing. A genetic algorithm is used to solve the composition problem. It differs from our work which uses constraint optimization because of its flexibility in defining user requirements.

The authors in [3] accommodate user's QoS requirements written in extended WS-Policy [8]. They propose an interesting approach to matching user requirement and provider capability that uses not only syntactical QoS attributes but also semantic reasoning through QoS ontology. However, their model uses UDDI [9], an almost-extinct service directory, so is difficult to implement.

A recent work is done by Mobedpour among others [5] in the context of centering user support in the composition. They provide a complete mechanism to assist users in formulating their QoS needs by presenting a query formulation and user interface design. By using their user interface design, we can focus on the internal process of QoS calculation. Our work might be seen as complementary to their work in that it accommodates more user aspects of QoS and supports composite services.

The use of the constraint optimization problem in composing Web services was initially proposed in [4]. However, they avoided discussion of QoS and decomposition of abstract workflow, both are required in modern service composition. The attempt to implement this work in a dynamic environment, such as the language service domain, always failed to accommodate user requirements analyzed and decomposed in multilevel way. Hence, the fundamental exten-

Table 1. Comparison with existing techniques.

	Customized attributes	Interaction	Composite service	Multi-agent
[2]	None	Weak	Support	None
[7]	None	Support	Support	None
[1]	Support	Weak	Support	None
[3]	Support	Support	None	None
[5]	Support	Support	None	None
[4]	None	None	Support	Support
Ours	Support	Support	Support	Support

sion includes further decomposition of subtasks based on service workflow control constructs for guaranteeing interoperability among selected services dealing with a dynamic environment.

Table 1 compares the most important composition techniques covered in our literature review and our proposed platform in terms of how well the approaches accommodate QoS attribute-driven customization for each service domain, degree of interaction between users and services, the availability of multi-user support in the same composition, support of composite services, and whether the approach supports autonomous agents during service composition.

3 Multilevel Formalization

A constraint-based formalization of the Web service composition problem has already been proposed [4, 10]. This formalization assumes, however, that each subtask is an atomic process and hence cannot be analyzed for decomposition. Nevertheless, in real cases some required services might be unavailable. Such cases require greater formalization flexibility.

In this section, we present a more generic, flexible formalization of the multilevel analysis for the Web service composition problem. In this new formalization, each subtask t^1 is represented as a multilevel constraint optimization problem (COP [11]) by a sequence of COPs, P^1_1,P^1_2,...,$P^1_{\alpha-1}$,P^1_α, ..., where P^1_1 initially involves only *one* variable, with its domain, its set of related constraints, and its objective function. Each P^1_α is derived from relaxation and/or restriction of the previous problem, i.e., $P^1_{\alpha-1}$.

The use of a multilevel COP to analyze each subtask allows us to represent a component service within composition. Hence, each subtask is represented by a single variable as an atomic process, or by a set of variables linked together, a composite process. Four types of control constructs are used: *Sequence*, *Concurrent*, *Choice*, and *Loop*. A *Loop* is represented by a set of sequences with a specific number of iterations. A *Concurrent* control construct is represented by a set of sequences with universal constraints among them.

A Web service composition problem is defined by a set of multilevel COPs, $\{P^1,...,P^n\}$, where n is the number of initial subtasks $\{t^1,...,t^n\}$ involved in the user's query. Each P^i is represented by a sequence of COPs $(\mathrm{X}^i, \mathrm{D}^i, \mathrm{C}^i, f(sl_{\models\{X^i\}}))$, as follows:

- $\mathrm{X}^i=\{x^i_1,x^i_2,...\}$ is the set of possible *abstract* Web services able to solve t^i. Initially $\mathrm{X}^i=\{x^i_1\}$. Each x^i_k is represented by $(x^i_k.in, x^i_k.out)$, standing for sets of inputs/outputs of the concrete Web services.

- $\mathrm{D}^i=\{\mathrm{D}^i_1,\mathrm{D}^i_2,...\}$ is the set of domains for each x^i_k. Initially $\mathrm{D}^i=\{\mathrm{D}^i_1\}$. Each D^i_k rep-

resents possible *concrete* Web services, whose service profile descriptions semantically match the subtask specification [12]. The input of the Web service (*resp.* the output of the subtask) is semantically included in the input of the subtask (*resp.* output of the Web service). For each concrete Web service $s_{kj} \in \mathrm{D}_k^i$, we assign both a weight to express the degree of user preference, $w_{s_{kj}} \in [0, 1]$, and the validity time $v_{s_{kj}}$ of the information maintained by s_{kj}. The information from user is uncertain, so we assign accuracy probability p_{kj} to each bit of received information and required time $e_{s_{kj}}$ to execute s_{kj}.

- We define two types of constraints: intra-problem constraints and inter-problem constraints. A constraint is considered to be an intra-problem constraint if and only if all variables involved in the constraint belong to the same problem P^i. Otherwise, the constraint is taken to be an inter-problem constraint. In the following, C^i refers to the set of intra-problem constraints related to P^i, while C refers to the set of inter-problem constraints. Both types of constraints is defined by $\mathrm{C}=\{\mathrm{C}_S \cup \mathrm{C}_H\}$(*resp.* $\mathrm{C}^i=\{\mathrm{C}_S^i \cup \mathrm{C}_H^i\}$).
  - $\mathrm{C}_S=\{\mathrm{C}_{userQuality}, \mathrm{C}_{QoS}\}$, where $\mathrm{C}_{userQuality}$ represents the soft constraints related to the information from the user that influence QoS such as user preference for particular dictionary service based on her language skill; and C_{QoS} represents constraints related to QoS attributes both from network service domain, such as cost, execution time and reliability; and other service domains such as fluency and adequacy in language service domain. For each soft constraint $\mathrm{c}_l \in \mathrm{C}_S$, we assign a penalty, $\rho_{c_l} \in [0, 1]$ reflecting the degree of unsatisfiability of c_l.
  - $\mathrm{C}_H=\{\mathrm{C}_{controlConstructs}, \mathrm{C}_{preCondition}, \mathrm{C}_{user}\}$, where $\mathrm{C}_{controlConstructs}$ represents the hard constraints related to the defined control constructs, $\mathrm{C}_{preCondition}$ represents the preconditions of each *concrete* Web service, and C_{user} represents the set of hard constraints imposed by the user. For each hard constraint $\mathrm{c}_f \in \mathrm{C}_H$ we assign a weight=1. Only C_{user} can be transformed into soft constraints and so can be relaxed upon request when no solution is found.
- $f(sl_{\models\{X^i\}})$ is the objective function to optimize while projecting the solution to set X^i, i.e., sl is a solution of the problem defined by instantiating of all variables of the problem.

In the following, we present one possible way to compute optimal assignment sl^*. For simplicity, we introduce $f(sl)$, same function applicable to $f(sl_{\models\{X^i\}})$. Hence, as indicated in Eq. 1, $f(sl)$ is defined as the summation of the user preferences $Preference(sl)$, QoS attributes $QoS(sl)$, time availability $availableTime(sl)$, and penalty $\psi(sl)$ for all *concrete* Web services involved in solution sl.

$$f(sl) = \begin{array}{l} \alpha * Preference(sl) + \beta * QoS(sl) + \\ \gamma * availableTime(sl) - \delta * \psi(sl) \end{array} \quad (1)$$

with α, β, γ, and $\delta \in [0, 1]$ as adjustable weights that depend on the service domain. For example, if the user request deals with planning a trip overseas, the weight associated with the user's preferences (i.e. α) should be greater than that associated with the QoS (i.e. β) because

a user may prefer a service with less service quality. However, in the case of text translation the weights associated with the QoS and Web service availability, should dominate.

Solving a multilevel analysis for a Web service composition problem consists of finding the best assignment of variables sl^* such that all hard constraints are satisfied while optimizing the following function:

$$sl^* = arg \max_{sl \in Solution} f(sl) \tag{2}$$

such that

$$Preference(sl) = \sum_{s_{kj} \in sl} w_{s_{kj}} \tag{3}$$

$$QoS(sl) = \begin{array}{l} \sum_{s_{kj} \in sl} networkAttributes(s_{kj}) + \\ \sum_{s_{kj} \in sl} otherAttributes(s_{kj}) \end{array} \tag{4}$$

$$availableTime(sl) = \min_{s_{kj} \in sl} (p_{kj} v_{s_{kj}} - e_{s_{kj}}) \tag{5}$$

$$\psi(sl) = \sum_{c_k \in C_S} \rho_{C_k} \tag{6}$$

In addition to a series of *concrete* Web services in the solution, we use "," to indicate sequential execution and "||" to indicate concurrent execution. An example of an obtained solution is sl=$\{s_{1i}$,$\{s_{2j} \| s_{3k}\}$,s_{4h},. . .,$s_{nm}\}$.

Note that this formalization allows us to render a multilevel analysis of the Web service composition problem more flexibly while attempting to generate a possible simple abstract workflow for each subtask detected as insoluble. Especially when the underlying required abstract workflow is complicated, this generation cannot be completed without the help of the user. Therefore, we propose to integrate user interaction into the generation of an appropriate abstract workflow for each subtask, when needed.

4 User-centered Composition

Language services are found on all levels of human-language-related applications and business processes developed with Web services technology. They have a great potential for being run jointly. There have been many efforts in aiming at finding the best combination of language services delivered to not only linguistic community but also end users such as in [13]. Each language service is wrapped by using a WSDL-based standard interface of natural language processing bundled with other descriptions such as QoS attributes, user access, and penalties.

A unique characteristic of the language service is the complexity of user preferences [14]. The attributes of the user determines the language service quality. Assume that there is a Japanese user who wants to use dictionary service. Since there are two dictionary services available, English-to-English and English-to-Japanese dictionary service, the composition should consider the QoS-related state of the user. User's mother tongue and other language capability given by the language certificate are used as QoS-related information from the user. This information can be easily acquired in the language service since all users are registered. This characteristic of language service is clearly shown in Fig. 1.

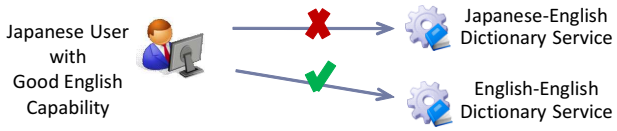

Fig. 1. The sophisticated user preference.

Another characteristic of the language service is multiuser composition. More than one user can utilize the same service at the same time such as a multilingual chat service[b]. Since we need to accommodate interaction between all users and the services, we combine all necessary information from services and users related to the services. In assessing the quality of a translation service, for example, we combine the language skills of each user and the translation accuracy into single QoS attribute used to compose services. We call this attribute the quality of message (QoM) attribute as described in Fig. 2.

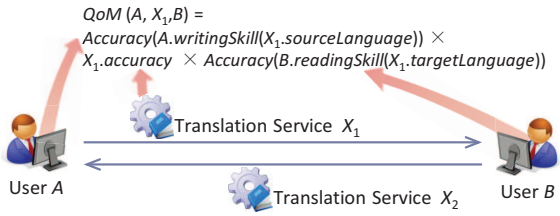

Fig. 2. Multiuser support in QoS calculation.

The last characteristic in the language service is multilevel analysis during composition. This is shown by the possibility of tailoring some atomic language services represented in an abstract workflow as illustrated in Fig. 3. One of the tailored language service workflows is the back translation service; it is used to check the quality of the machine translation service. This service can be decomposed into a community based translation service whenever the user finds that the quality of each atomic service in the back translation service is not enough. However, since service availability may not support all user decompositions, this service is decomposed again into a multi-hop translation service; the setting of a pivot language yields a new translation service between two languages that was previously unavailable.

A language service composition problem is defined by a set of multilevel COPs, $P^1,\ldots,P^6$. Each P^i is a sequence of $P^i_1,P^i_2,\ldots$, where initially each P^i_1 is defined by (X^i, D^i, C^i, $f(sl_{\models X^i})$) as shown in Table 2 (note that the D^i entries are based on available services as August 2014).

Due to the space limits, we start the formalization from level 2 as shown in Fig. 3 (b). No intra-problem constraints exist for all the above problems and for all X^i, i ∈ [1, 6], $f(sl_{\models\{X^i\}})$ as defined in Eq. 1. Some inter-problem constraints are defined as follows (note that QoS values and dimensions can be normalized [15]):

- Soft constraints c_1: $Latency(\mathrm{X}^5_1) \leq 0.7$ with ρ_{c_1}=0.6; c_2: $Cost(\mathrm{X}^2_1) + Cost(\mathrm{X}^4_1) \leq y$ with ρ_{c_2}=0.3; c_3: $QoM(\mathrm{A}, \mathrm{X}^5_1, \mathrm{B}) \geq 0.4$;
- Hard constraints c_3: X^1_1.originalSentence $\neq$ *null*; c_4: X^2_1.morphemes = X^1_1.morphemes;

[b] http://langrid.org/playground/chat/ChattingMain.html

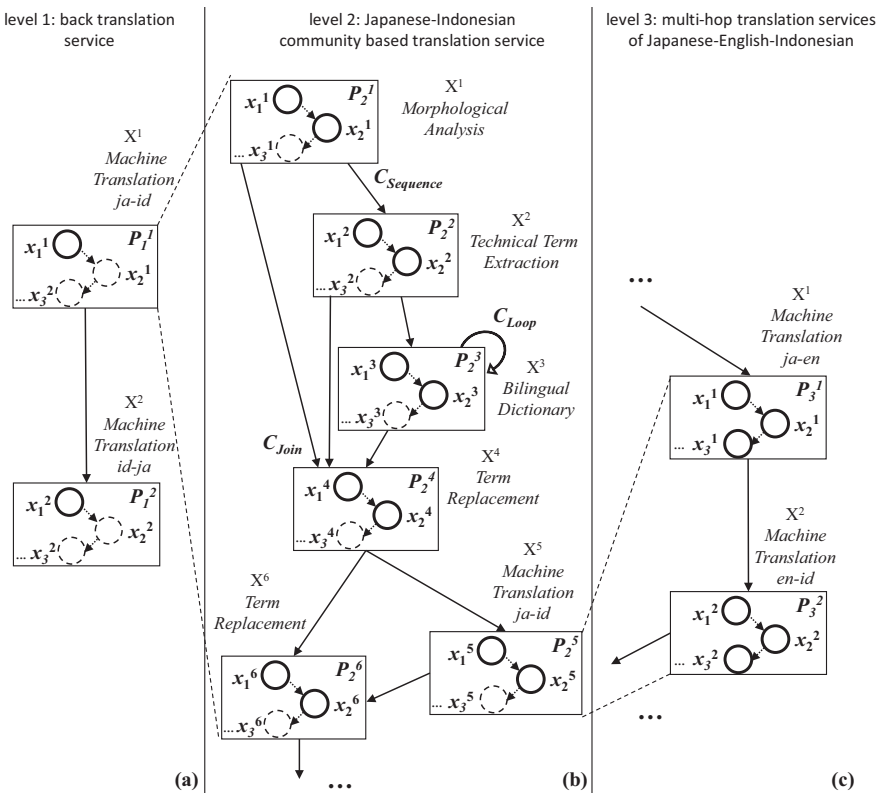

Fig. 3. Constraint graph of the abstract workflow of the language service composition.

c_5: X_1^3.technicalTerms=X_1^2.technicalTerms;
c_{Loop}: X_1^3.technicalTerms $\neq$ *null*;

The above workflow is the generic, abstract one used for language services. Suppose that a user wants to translate a sentence including some technical terms from Japanese to Indonesian which is currently not available unless a multi-hop translation service with English as the pivot language is used (see Fig. 3 (c)). Subtask X^5 cannot be fulfilled. We have to enable the composition process to add x_3^1 to X^1 and x_3^2 to X^5 as shown in Table 3 with their domain and related intra-problem constraints. We try to solve the two subtasks by using two variables, if possible; otherwise another variable is added and the same process continued until it becomes possible to solve t^i or to its insolubility is confirmed.

5 Architecture and Algorithm

In this section, we provide an architecture that accommodates the use of multi-agents, and an algorithm that is implemented inside the architecture to allow the agents to solve the problem of Web services composition.

5.1 Multi-agent based architecture

We propose a hybrid architecture of multi-agents and services to solve the problem of multilevel analysis of service composition. This multi-agent architecture provides enough

Table 2. The set of variables and their domains defined for each P^i.

P_1^1={P_2^1}	x_1^1: *morphological analysis*; $x_1^1.in$={language, text}; $x_1^1.out$={analyzeReturn}; D_1^1={Chasen, Juman, Mecab, TreeTagger,...}
P_1^1={P_2^2}	x_1^2: *technical term extraction service*; $x_1^2.in$={morphemes}; $x_1^2.out$={technicalTerms}; D_1^2={CaboCha}
P_1^1={P_2^3}	x_1^3: *bilingual dictionary service*; $x_1^3.in$={headLang, targetLang, headWord, matchingMethod}; $x_1^3.out$={searchReturn}; D_1^3={Lextron-dict-public, Life Science Dictionary, Glossary on Natural Disasters, Wikipedia,... }
P_1^1={P_2^4}	x_1^4: *term replacement service*; $x_1^4.in$={originalSentence, technicalTermsTranslated, technicalTermsIntermediateCode}; $x_1^4.out$={intermediateCodeSentence}; D_1^4={TermReplacementService,...}
P_1^1={P_2^5}	x_1^5: *machine translation service*; $x_1^5.in$={sourceLang, targetLang, source}; $x_1^5.out$={translateReturn}; D_1^5={Google translate, Parsit, Translution, Toshiba, YakushiteNet, J-server, Web-Transer,...}
P_1^1={P_2^6}	x_1^6: *term replacement service*; $x_1^6.in$={intermediateCodeSentenceTranslated, technicalTermsIntermediateCode, technicalTermsTranslated}; $x_1^6.out$={originalSentenceTranslated}; D_1^6={TermReplacementService}

autonomy for service users and providers in decomposing Web services without neglecting the interaction between user and services. Service-oriented architecture lacks this capability and, so forces service users and providers to decide the decomposition manually.

The hybrid architecture consists of three kinds of agents, Abstract Web service agents, one or more Information analysis agents, and a User agent. The Abstract Web service agents have to detect the global termination when either a solution is found or an inconsistency exists during the composition process. These agents also store already processed services and workflows in a service repository in order to enable further offline processing. The User agent is required to create Abstract Web service agents and the Information analysis agents and, most importantly, to inform the user of the result.

Information analysis agents process the language services provided by service providers. In this paper, we use the language services provided by the Language Grid, a project that is collecting and sharing 142 atomic language services from different institutions world wide.[c] The Language Grid project eases the effort needed to use a lot of language services without

[c] http://langrid.org/service_manager/language-services

Table 3. Variables of the decomposed workflow.

P_2^5=$\{P_3^1\}$	x_3^1: *machine translation service of Japanese-English*; $x_3^1.in$={Japanese, English, source}; $x_3^1.out$={translateReturn}; D_3^1={Google translate(Kyoto), J-Server(NICT), YakushiteNet,...}
P_2^5=$\{P_3^2\}$	x_3^2: *machine translation service of English-Indonesian*; $x_3^2.in$={English, Indonesian, source}; $x_3^2.out$={translateReturn}; D_3^2= {Google translate (Bangkok)}; intra-problem $C_{Sequence}$: $x_3^2.in \subseteq x_3^1.out$ and C_{Loop}: $x_k^2 \neq x_{k-2}^2$, $\forall\ k$

dealing with a specific service level agreement (SLA) [16] for each service provider. These atomic language services are further combined as composite services by utilizing a particular workflow.

Another task of Information analysis agents is to receive a request from other agents for analyzing information such as the maximum time needed to run particular Web service, its validity, and quality of service. These agents analyze the information received from a service provider by using some information analysis tools wrapped as services in our project, a credibility analysis provided by WISDOM [17] and information relation analysis provided by Torishiki-kai [18]. The proposed architecture of these agents described in Fig. 4 is inspired from Singh and Huhns's book [19].

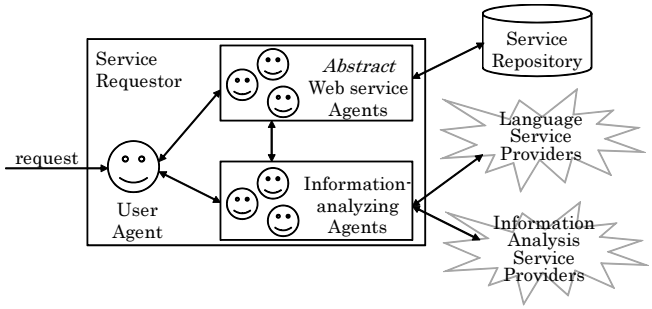

Fig. 4. A hybrid service composition architecture.

5.2 Global dynamic algorithm

Each agent has static and dynamic knowledge, a local behavior, a set of acquaintances and a *mailbox*. Each agent A_i is responsible for a multilevel COP and has its own local knowledge (static and dynamic) and a reasoning engine. The local goal of each agent is to solve its local problem (initially includes only one variable) while optimizing the objective function. The negotiation process consists of two main overlapping steps:

- Solve each local multilevel COP P^i (X^i, D^i, C^i, $f(sl_{\models\{X^i\}})$) in order to find the local set of best solutions, while enforcing local bidirectional arc-consistency [20] between each pair of connected agents.

- Propagate consistency throughout the network until the best subset of global solutions is found.

Each agent A_i begins solving its local problem P^i, which initially includes *only one* variable $X^i=\{x_1^i\}$. It first reduces its set of *concrete* Web services D^i by eliminating all non-viable values according to its unary constraints (Algorithm 1, line 1). If D_1^i becomes empty, then the user is asked to relax some of her constraints that directly or indirectly involve x_1^i (Algorithm 1, line 3). We propose to relax user's hard constraint in case of failure before proceeding to local decomposition in an attempt to reduce the total cost. If no possible constraint can be relaxed for X^i, then the agent A_i proceeds by decomposing subtask $X^i=\{x_1^i\}$ into two sub-subtasks $X^i=\{x_1^i, x_2^i\}$. A new variable is added to the local problem P^i (Algorithm 1, line 5). The domains of the two variables are updated to include not only functionally equivalent Web services that exactly match semantically with the subtask description, but also all concrete Web services whose descriptions are semantically included in the subtask description. Also, a new *Sequence* intra-problem constraint is added. A^i tries to find consistent bindings for these two variables while optimizing the local objective function $f(sl_{\models\{X^i\}})$. If no sub-solution can be found, a third variable, x_3^i, is added to X^i and the same process reiterated until a solution is found or the insolubility of underlying subtask X^i is determined.

A_i ranks D_i according to user information, i.e., $w_{s_{ij}}$, QoS, and the availability of information. If the local problem includes only one variable, then A_i selects the best k viable candidates, $Candidate_{X^i}$. Otherwise, A_i tries first to generate k local best sub-solutions (Algorithm 1, lines 10 to 14), then determines the set of $Candidate_{X^i}$, which includes the values taken by the first and last variables in the generated local abstract workflow. A_i communicates its set of selected concrete Web services to its directly linked lower priority agents together with the minimum time for which the information about these concrete Web service candidates remains valid (Algorithm 1, line 24). Each agent A_i receiving a message to process concrete Web services $Candidate_{X^j}$ from the agent A_j proceeds by first enforcing *lazy* arc-consistency on its $Candidate_{X^i}$ (Algorithm 2, lines 1 to 5). For each Web service $s_{ik} \in Candidate_{X^i}$, A_i seeks, for the first compatible Web service $s_{jl} \in Candidate_{X^j}$ that satisfies C_H and minimizes the penalty $\psi(sl)$ for all violated C_S. Each s_{ik} that has no support in $Candidate_{X^j}$, is *temporarily* discarded from $Candidate_{X^i}$ and rechecked whenever any modification is performed on the set of received candidates. Once all received concrete Web services are checked, and all of them are discarded, A_i selects the next set of best k candidates from D_i, if possible. Otherwise, a *backtrack* message is sent to the "nearest" parent to ask for more concrete Web services (Algorithm 2, line 7). The order of the asked parents is maintained in order to ensure the completeness of this protocol.

If not all concrete Web services are discarded from $Candidate_{X^i}$, the agent enforces *lazy* arc-consistency on the received candidates $Candidate_{X^j}$. For each received s_{jl}, that was unchecked in the previous step, A_i seeks, for the "first" support in D_i. This step (i) allows us to detect inconsistent concrete Web services (that cannot belong to any solution), in order to eliminate them from the domain and consequently reduce the size of the problem; and (ii) aims mainly to reduce both the number of backtracks and messages. Note that if any $s_{jl} \in Candidate_{X^j}$ is detected as inconsistent, then the parent A_j should be informed to remove s_{jl} from its domain (Algorithm 2, line 24), since s_{jl} cannot be involved in any solution. Finally, after performing bidirectional arc-consistency on both sets of candidates (local and

Algorithm 1 *Begin* message in each agent A_i.

BeginWith:*j*

1: Filter updated D^i such that all local hard intra-agent constraints are satisfied;
2: **if** $\exists\ D^i_k \in D^i$, $k \leq j$ such that $D^i_k = \emptyset$ **then**
3: Request a relaxation of hard constraints to user;
4: **if** *No more constraints to relax* **then**
5: Add x^i_{j+1} to X^i; Update D^i and C^i; *BeginWith*:(*j+1*);
6: **end if**
7: **end if**
8: Rank all $D^i_k \in D^i$ according to $(w_{s_{ij}} + p_{ij} * v_{s_{ij}} - e_{s_{ij}})$;
9: **if** $\|X^i\| > 1$ **then**
10: Generate k possible sub-solutions;
11: **if** *no possible sub-solutions* **then**
12: Add x^i_{j+1} to X^i; Update D^i and C^i; *BeginWith*:(*j+1*);
13: **else**
14: $Candidate_{X^i} \leftarrow \{Candidate_{x^i_1} \cup Candidate_{x^i_h}\}$ for the best k candidates;
15: **end if**
16: **else**
17: $Candidate_{X^i} \leftarrow \{\forall\ x^i_k,\ k \leq j \mid$ "*best*" $\boldsymbol{k}$ candidates$\}$;
18: **end if**
19: **if** any information is required for any concrete Web service s_{il} **then**
20: *send*(*Information analysis*, *self*, *RequestInformationFor*:s_{il});
21: **end if**
22: $availableTime^{A_i} \leftarrow \min_{s_{il} \in Candidate_{X_j}} (ps_{il} v_{s_{il}} - e_{s_{il}})$;
23: **for all** $A_j \in Children^{A_i}$ **do**
24: *send*(A_j, *self*, *process*:$Candidate_{X^i}$ within:$availableTime^{A_i}$);
25: **end for**

received), A_i generates its set of sub-solutions and sends it with its (new) set of candidates to all $Children^{A_i}$ (Algorithm 2, line 29).

Any agent receiving a backtrack message looks first for new, possible candidates. If the local problem involves more than one variable and no more sub-solutions can be locally found, then the agent tries to relax and/or restrict its local workflow by removing and/or adding new variables to increase sub-solutions size. User can join this step to help generate a new local abstract workflow.

We note that performing only bidirectional lazy arc-consistency on the set of concrete Web services and the received services is not sufficient to ensure resolution of the problem. Therefore, each agent A_i receiving a message to process alterations first updates its set of candidates according to the new candidates. If $Candidate_{X^i} \neq \emptyset$, then A_i detects the set of shared abstract Web services among its higher priority agents and checks these services' consistency. This latter step allows us to enforce consistency among the paths of linked abstract Web services, known as path consistency. If any conflict occurs, and no consistency is detected between at least two sets of candidates, then a *backtrack* message is sent to the "nearest" parent for propagation to the concerned agent maintaining the shared variable source

Algorithm 2 *Process* messege within each agent A_i.

process:*Candidate*$_{X^j}$ within:*t*

1: **for all** $s_{il} \in Candidate_{X^i}$ **do**
2: **if** $\exists\ s_{jk} \in Candidate_{X^i}$ such that s_{il} compatible with s_{jk} **then**
3: $Candidate_{X^i} \leftarrow Candidate_{X^i}/s_{il}$; update $availableTime^{A_i}$; add($checkedValue$[X_j], s_{il});
4: **end if**
5: **end for**
6: **if** $Candidate^{X^i} = \emptyset$ **then**
7: **if** Possible *backtrack* **then**
8: *send Backtrack* message to $Parents^{A_i}$ to ask for more candidates;
9: **else**
10: Request user for a relaxation of hard constraints related in/directly to X_i;
11: **if** *No more constraints to relax* **then**
12: Inform user of inconsistency of the problem; Exit;
13: **end if**
14: **end if**
15: **end if**
16: **for all** $s_{jh} \in Candidate_{X^j}$ **do**
17: **if** $\neg(s_{jh} \in checkedValue)$ **then**
18: **if** $\exists\ s_{im} \in D_i$ such that s_{im} is compatible with s_{jh} **then**
19: add($inconsistentValue$, s_{jh});
20: **end if**
21: **end if**
22: **end for**
23: **if** $inconsistentValue \neq \emptyset$ **then**
24: *send*(A_j, *self*, *removeFromYourDomain*:$inconsistentValue$);
25: **end if**
26: Generate $subSolution^{A_i}$;
27: Rank $subSolution^{A_i}$ according to $f(solution)$;
28: **for all** $A_j \in Children^{A_i}$ **do**
29: *send*(A_j, *self*, *process*:$Candidate_{X^i}$ and:$subSolution^{A_i}$ within:$availableTime^{A_j}$);
30: **end for**

of conflict. If the set of received candidates is consistent, A_i updates its set of sub-solutions, i.e., $subSolution^{A_i} = \bowtie subSolution^{A_i} \cup_{j \in Parents^{A_i}} subSolution^{A_j}$. The obtained $subSolution^{A_i}$ is then ranked again according to $f(subSolution^{A_i})$ (Eq. 1), and finally sent to $Children^{A_i}$ with the corresponding set of *maxTime* values, i.e., estimations of the maximum time intervals for the validity of $subSolution^{A_i}$ and the (new) set of $Candidate_{A^i}$.

Termination should be detected locally (one or several times) by each agent and globally by all agents in progressive manner. Local termination can be detected when either of the following applies:

- There is no concrete Web service for the sub-subtask x^i_k such that $x^i_{k-1}.output = x^i_k.input$, AND there is no concrete Web service for sub-subtask x^i_k such

Table 4. Experimental results obtained.

	CPU Time (in Sec)	Number of Checks	Number of Messages	Inconsistency (in %)
10/0.40	4.29	1527.67	192.21	40
20/0.30	8.58	7504.68	386.18	60
30/0.25	4.32	9409.36	279.99	70
40/0.25	4.51	6941.07	211.09	30
50/0.20	10.67	21693.05	347.20	50

that $x^i_{k+1}.input{=}x^i_k.output$.

- $x^i_k.output$ is semantically equivalent to $x^i_h.input$ for any $h{<}k$, i.e., for any previous sub-subtask.

For the first case, local termination is detected whenever no possible path can be found for at least one extremity of the sequential local abstract workflow. In contrast, termination in the second case is related to detecting an infinite Loop process. For the global termination, the stable state (i.e., the state where a solution is found or the problem is declared inconsistent) is progressively detected by all the Abstract Web service agents.

6 Experimental Evaluation

Our aim was to evaluate the performance of the proposed approach in the case when the number of existing Web services was increased. We tested our approach in two ways. First, we developed an agent-based test scenario by using Actalk under the Smalltalk-80 environment. Second, we conducted an experiment on a service-oriented platform to evaluate the service response time given different levels of composition and QoS values.

In the first experiment, we used the following parameters: n agents, d concrete Web services per agent, p percentage of possible basic constraints, q percentage of allowed pairs of concrete Web services per constraint, p_{Info} probability of requesting an additional information for s_{ij}, and $w_{s_{ij}}$ weights for the soft constraints expressing user's preferences. We generated instances with $n{=}10$ agents, $d{=}\{10; 20; 30; 40; 50\}$, and $p_{Info}{=}0.20$. We set $p{=}0.15$ for the basic/user hard constraints and $p{=}0.15$ for the soft constraints, $q{=}\{0.4; 0.3; 0.25; 0.2\}$ to deal with the hardest problems (i.e., problems belonging to the phase transition), and $w_{s_{ij}} \in [0, 1]$. For each pair d/q, 10 instances were generated and each was executed 30 times. Table 4 shows the averages of the obtained results in terms of three criteria: the CPU time in seconds, the number of checked constraints, and the number of exchanged messages.

In terms of CPU time, our approach was able to solve the composition problem for typical cases (10/0.40 and 20/0.30) in a short time, with the minimum number of constraint checks and exchanged messages. For the case of 40/0.25, however, we noticed that our approach required less CPU time and fewer constraint checks and exchanged messages than for the case of 20/0.30. This is explained by the fact that the 20/0.30 case had a higher percentage of inconsistent problems than the 40/0.25 case. To prove the inconsistency of the problem, the approach requires an exhaustive search to check all possible values for all variables. The same applied to the 50/0.20 case, in which the proposed method performed an exhaustive

search to prove the insolubility of the underlying instance 60% of the time. Hence, we can say that the proposed protocol in agent-based system is scalable.

In the second experiment, we examined four scenarios with two levels of composition together with their corresponding QoS values. We chose two levels for evaluation efficiency. The first level of composition follows the number of users in each scenario. The scenarios are two service instances for two users, six service instances for three users, 12 service instances for four users and so forth. Detailed evaluation results are shown in Fig. 5.

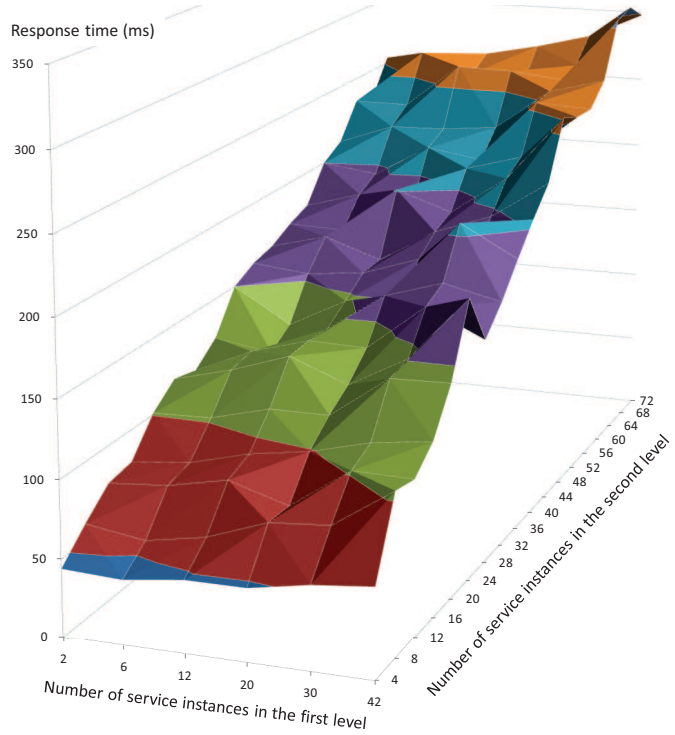

Fig. 5. Two level service composition evaluation

As can be inferred from the evaluation result, the response time required to run our framework generally depends both on the number of service instances on the levels of the composition (two in this example). However, performance depends strongly on the number of service instances in the second level of composition. This can be explained by the fact that our approach calculates QoS values of each service in the decomposed workflow. In the first level composition, we obtain QoS values from the decomposed workflow and use them to find the most optimal combination of services to be delivered to users. The optimization process requires significantly more time when calculating QoS values. The evaluation result shows the advantage of our framework in that it allows the decomposition to be processed offline. Therefore, the response time to process our framework can be shortened significantly by reducing the number of service instances in the decomposed workflow. This can be done when one or more workflows have been decomposed before and stored in service repository.

7 Conclusion

We adopted agent-based constraint optimization as it is a promising technique for solving the problem of composing Web services which exist in multilevel and user-centered environment such as language services. This technique allows us to represent any atomic subtask included in the original abstract workflow as a component process whenever this subtask cannot be concretized. An interactive agent-based protocol was introduced to offset the deficiencies of existing techniques (they deal with only concrete Web service composition) in that it exploits and extends (if possible) their abstract workflows in order to identify satisfactory executable workflows according to predefined optimality criteria. The developed negotiation protocol solves the problems of interoperability among Web services while complying with most natural features of realistic problems such as the dynamism of the environment during the composition and execution processes. In addition, this protocol allows the user to interfere in order to enhance the search for a solution by relaxing her constraints and/or assisting in the generation of the local abstract workflow.

A hybrid multi-agent and service-oriented architecture was implemented to enable automation of the decomposition process and only user-constraint relaxation is left as a manual process. Experiments showed that the proposed architecture is able to solve problems and find solutions within a feasible time, while performing the minimum number of required constraint checks and reducing the number of exchanged messages as much as possible.

Acknowledgements

The work is partially supported by a Grant-in-Aid for Scientific Research (S) (24220002, 2012-2016) from Japan Society for the Promotion of Science (JSPS)

References

1. G. Canfora et al. (2008), *A framework for QoS-aware binding and re-binding of composite web services*, J. Systems and Software, Vol.81, No.10, pp. 1754-1769.
2. L. Zeng et al. (2004), *QoS-aware middleware for web services composition*, IEEE Trans. on Software Engineering, Vol.30, No.5, pp. 311-327.
3. S. Chaari et al. (2008), *Enhancing web service selection by QoS-based ontology and WS-policy*, ACM symposium on Applied computing, pp. 2426-2431.
4. A. Ben Hassine et al. (2006), *A constraint-based approach to horizontal web service composition*, International Conference on Semantic Web, pp. 130-143.
5. D. Mobedpour and C. Ding (2013), *User-centered design of a QoS-based web service selection system*, Service Oriented Computing and Applications, Vol.7, No.2, pp. 117-127.
6. M. Mahboobeh and J.G. Davis (2014), *Service selection in web service composition: a comparative review of existing approaches*, Web Services Foundations, Springer New York, 321-346.
7. B. Medjahed et al. (2014), *On the composability of semantic Web services*, Web Services Foundations, Springer, pp. 137-160.
8. S. Bajaj et al. (2006), *Web services policy framework (WS-Policy)*, http://www-128.ibm.com/developerworks/library/specification/ws-polfram/.
9. UDDI Spec Technical Committee (2003), *UDDI version 3.0.2*, http://uddi.org/pubs/uddi_v3.htm.
10. A.B. Hassine et al. (2007), *Dynamic horizontal composition for semantic Web services: an investigation of real use*, International Conference on Semantic Web, pp. 7-8.
11. R. Dechter (2003), *Constraint processing*, Morgan Kaufmann, pp. 1098-1106.

12. M. Paolucci et al. (2002), *Semantic matching of Web services capabilities*, International Conference on Semantic Web, pp. 333-347.
13. T. Ishida (2011), *The language grid*, Springer.
14. A. Bramantoro and T. Ishida (2009), *User-centered QoS in combining Web services for interactive domain*, International Conference on Semantics, Knowledge and Grid, pp. 41-48.
15. N.B. Mabrouk et al. (2009), *QoS-aware service composition in dynamic service oriented environments*, Middleware 2009, Springer, pp. 123-142.
16. D. Verma (1999), *Supporting service level agreements on IP networks*, Macmillan Technical Publishing.
17. T. Kawada et al. (2011), *Web information analysis for open-domain decision support: system design and user evaluation*, Joint WICOW/AIRWeb Workshop on Web Quality, pp. 13-18.
18. K. Torisawa et al. (2010), *Organizing the Web's information explosion to discover unknown unknowns*, New Generation Computing, Vol.28, No.3, pp. 217-236.
19. M.P. Singh and M.N. Huhns (2005), *Service-oriented computing*, John Wiley & Sons.
20. A.K. Mackworth (1977), *Consistency in networks of relations*, Artificial intelligence, Vol.8, No.1, pp. 99-118.