A Method of E-Service Workflow Composition Based on Linear Logic Inference Rules

Shan Zhou
School of Electrical Engineering and Information
ChangChun institute of Technology
ChangChun, China
e-mail: 715711990@qq.com

Fangyu Zhang
College of Information Sciences and Technology,
jilin University
changchun, China
zhangfangyu11@163.com

***Abstract*—The paper proposes a method for semantic message matching in automatic service composition. It develops a framework in which the exported message description and behavior description of a service, and represents the behavior of a service with a finite state machine. Since the service interface definition can be represented by ontology concepts, the internal representation language enables us to define some issues required by service composition formally, qualitative and quantitative constraints plus reasoning on concepts, and the service behavior can be represented using linear logic formulas, so the inference rules of linear logic can check the match-ability and satisfy-ability of service message.**

Keywords- E-service; composition; linear logic; Workflow

I. INTRODUCTION

With the explosive growth of Internet, more enterprises are providing various E-services for collaborative commerce online to achieve competitive advantages. Services are self-contained, modular applications that can be described, published, located, and accessed over network by using open standards. The functionality of the individual service is limited and cannot satisfy some practical requirements. The potential of services can only be achieved if they are used to dynamically compose some new services that provide more sophisticated functionalities compared to existing ones [1].

The service composition is a highly complex task, and it is already beyond the human capability to deal with the whole process manually. Rao et al.[2] proposed a method for automatic composition of Semantic Web services using Linear Logic (LL) [3] theorem proving. The services are presented by extralogical axioms and proofs in LL. A process calculus to present the process model of the composite service is used. The above protocols are based on a centralized broker that manages the service composition process. The drawback is that if a huge number of users attempt to access an increasing number of various services distributed over the network, the broker becomes quickly a bottleneck.

Also, agent-based techniques[4] have been proved to be feasible to realize the automatic systems of Web services. Agents are envisioned for automatic discovery, execution, and integration of Web services. However, no mature method has been proposed to manage birth, death, migration, stability, and communication processes of enormous agents. They can not meet with autonomous management, evolution, and adaptation of the next-generation Web service [5].

We describe a method for automated E-service composition which is based on the proof search in (propositional) multiplicative intuitionistic fragment of Linear Logic. Given a set of existing web services and a set of functionality and non-functionality attributes, the method finds a composition of atomic services that satisfies the user requirements. a description of existing web services (written in WSDL) is translated into extralogical axioms in Linear Logic, and the requirements to the composite services are specified in form of a LL sequent to be proven. Second, we use a MILL theorem prover to determine whether the requirements can be fulfilled by the composition of existing atomic service. If it is possible then the last step is to construct flow models (written in BPEL4WS) from the generated proofs. We assume that the composite service is ready to be executed when the flow model and description of each atomic services are given. Because of soundness of MILL correctness of composite services is guaranteed with respect to initial specification. Completeness of MILL ensures that all composable solutions would be found. Our method can be regarded as an extension of the service composition method using Structural Synthesis of Programs proposed. The difference is that the method used in this Web Service Description Process Description Proof Axioms in LL.

The process of service composition paper considers also the quantitative and qualitative constraints in addition to the structural information of services. This paper is focusing on the second step of the composition process, namely presentation and proof using LL. We assume that WSDL presentation of services has been translated into a set of LL axioms by a compiler.

II. LINEAR LOGIC-BASED IMPLEMENTATION OF E-SERVICE COMPOSITION

An e-service is a software artifact that interacts with its clients in order to perform a specified task. In abstract model, the client can interact with the e-service instance by repeatedly choosing an action and waiting for either the fulfillment of the specific task, or the return of some

Published by Atlantis Press, Paris, France.

information. On the basis of the information returned, the client chooses the next action to invoke. The workflow can be used to realize a target e-service requested by the client, not simply by selecting a member of the community to which delegate the target e-service actions, but more generally by suitably "composing" parts of e-service instances in the workflow in order to obtain a virtual e-Service which "is coherent" with the target one..

E-service composition is the cooperation among service resources. Each agent should negotiate with others based on the capabilities that can be executed. We take advantage of full intuitionist LL. To use LL theorem proving as service composition negotiation is that LL is resource conscious logic. We can distinguish the information transformation and the state change produced by the service. Meanwhile, we can perform planning by using both qualitative and quantitative non-functional attributes. Because of soundness of the logic fragment, the correctness of composite services is guaranteed with respective to the initial specifications. Completeness of the logic fragment ensures that all compassable solutions can be found.

The service profile can be translated into LL axioms and LL sequences. In OWL-S, the information about Web services is presented by OWL-S classes and properties. They are translated into LL propositions referring to the specific classes and properties. The meaning of the propositions and the semantic relationships among the propositions are defined by the ontology relationships. After the service profile is translated into LL axioms and LL sequences, the next step is the negotiation among the agents in LL.

Negotiation is an interactive process involving partial deduction and LL theorem proving [4]. Partial deduction is applied as a method of deducing sub problems. Generally, a request to a composite service (including functionalities and non-functional attributes) can be expressed by the following LL formula:

$$\Gamma_a, \Gamma_b;\ \Delta_c \ |-\ ((I \otimes P)\ -\circ\ (O \otimes F) \otimes E)) \otimes \Delta_n \quad (1)$$

Where, Γ_a and Γ_b are sets of extra logical axioms representing available value-added Web services and core services, respectively. Δ_c is a conjunction of non-functional constraints. Δ_n is a conjunction of non-functional results. $\otimes$ is multiplicative conjunction. For example, $A \otimes B$ denotes that the literals A and B are consumed or achieved simultaneously. $-\circ$ is linear implication. For example, $A -\circ B$ means that the goal B is achievable only when resource A is available. A LL sequence is divided into two parts by symbol $|-$. For example, $A|-B$ means that the goal B can be achieved by consuming the resource A. $(I \otimes P)\ -\circ\ (O \otimes F) \otimes E)$ is a functionality description of the required service. I represent a set of input parameters for the service and O represents a set of output parameters produced by the service. P and F are multiplicative conjunctions of preconditions and effects, respectively. E presents an exception. Intuitively, the formula can be explained as follows: Given a set of available services and non-functional attributes, we try to find a combination of services that computes O from I as well as that changes the world state from P to F.

Partial deduction steps as inference figures are defined in LL. While using these inference figures instead of basic LL rules, we can achieve more efficient proof search and higher efficiency. Partial deduction is known as one of optimization techniques in logic programming. Its basic idea is as follows: Given a specification, partial deduction derives a new (more specific) specification while preserving the meaning of the original one. We can use partial deduction to extract the maximum information from incomplete knowledge in the sense of the following specialization inference rule. Following is the corresponding functional specification of what a rule specialization process is. The extension to specialization of the agent's bases is straight forward formula (2).

$$S:\ B \to B$$

$$S(a) = \begin{cases} S((R - \{r\} + \{r'\}, P + \{p'\}) & (*)) \\ a & otherwise \end{cases} \quad (2)$$

$$(*)\ \ if\ \ P \neq \phi\ \ and\ \ \exists p \in P\ \ and\ \ \exists r \in R$$
$$such\ \ that\ \ S_R(r, p) = (r', p')\ \ and\ \ r' \neq r.$$

where, B is a set of agents. We note agents as pairs a=(R, P), where R is a set of LL inference rules and P is a set of literals (agent states). In other words, the specialization of a agent's rule base consists of the exhaustive specialization of its rules. Rules that only have one condition appearing in the set of literals will be eliminated and a new literal will be added. This new literal will be used again to specialize the agent. The process will finish when the agent has no rule containing on its conditions.

The following LL inference rules [6], $R_b(L_i)$ and $R_f(L_i)$, are defined for partial deduction back (3) and forward (4) chaining steps, respectively.

$$\frac{S\ |-B \otimes C}{S\ |-A \otimes C} R_b(L_i(\underline{\lambda})) \quad (3)$$

$$\frac{A \otimes C\ |-G}{B \otimes C\ |-G} R_f(L_i(\underline{\lambda})) \quad (4)$$

where A, B, and C are LL formulae. L_i is a labeling of a particular LL axiom representing a agent's capability. $R_b(L_i)$ and $R_f(L_i)$ apply clause L_i to move the initial state towards the goal state or the other way around. In formulae (4), $A \otimes C$ and $B \otimes C$ denote goals G and G', respectively. It encodes that, if there is an extralogical axiom $|-B -\circ A$,

then goal *G* can be changed to *G'*. In formulae (5), $B \otimes C$ and $A \otimes C$ denote state *S* and *S'*, respectively. It encodes that, if there is an extralogical axiom $|-B-\circ A$, then initial state *S* can be changed to *S'*.

Additionally, we assume that $\underline{a} \stackrel{def}{=} a_1, a_2 \ldots\ldots$ is an ordered set of constants, $\underline{\lambda} \stackrel{def}{=} \lambda_1, \lambda_2 \ldots\ldots$ is an ordered set of variables, $[\underline{a}/\underline{\lambda}]$ denotes substitution, and $\lambda = \lambda'[\underline{a}/\underline{\lambda}]$. When substitution is applied, elements in $\underline{a}$ and $\underline{\lambda}$ are mapped in the order they appear in the ordered sets. These sets must have the same number of elements.

The implementation of LL negotiation among the agents is based on Lygon[7]. Lygon is a LL-based logic programming language, and can be viewed as Prolog extended with features derived from LL. These features include a clean declarative notion of state and the ability to express problems involving concurrency. As an abstracted language framework, Lygon can be implemented and expended in Java.

The application of Lygon to agent-oriented system is a new aspect. Since Lygon is suitable for concurrent programming, modeling actions, representing states, and searching, it is natural to use Lygon for working with the WSES. All LL inference rules can be implemented in Lygon. Lygon uses top-down computation, that is, a computation begins with a goal and seeks to prove it using the program. In our work, Lygon (version 0.7) written in fairly standard Prolog is used and should be easy to port to other Prolog systems. Lygon syntax is described in Table 1.

Table 1. Grammar for the Lygon.

G::=G⊗G| G⊕G| G&G G⅋G !G |negD|1|⊥|⊤|A| negA
D::=[linear](A1⅋A2⅋…⅋An<-G)
Top 1 One
Bottom 0 Zero
& Provable in any context
⊗ Provable only in empty context
& Cannot be proved, but can be weakened away
⊕ Not provable

When a agent has to compose a new service, sub-services are generated. The sub-services are distributed among the partners and treated as offers to other agents. Semantic descriptions of the existing services are translated into extra logical axioms of LL, by applying partial deduction to find partial solutions. Ontology is used to reason over the Semantics of Web services' inputs and outputs. Partial solutions can be extended through our Web service emergent framework until a complete solution to be found

III. SIMULATION RESULTS AND ANALYSIS

This simulation work implements two different scenarios and comparison between two experiment settings is conducted: WSES (agent) and OWL-S workflow composition with agent (non-agent). Because there is no standard testing data sets, the matching service data and partial deduction are generated randomly, which are assigned to the agents for testing the characteristics of service emergence. We make a set of common web service resources (which contains 500 different resource vectors).

In the simulation, the user request is the abstracted OWL-S information. The simulation is the procedure to generate a composite service with inputs and outputs according to the request. The simulation evaluates the adaptation and evolution from response time. Fig. 1 shows how service request rate changes with simulation time during 240 minutes. The simulation results show that the WSES can adapt and evolve to meet with user's requests, as shown in Fig.2

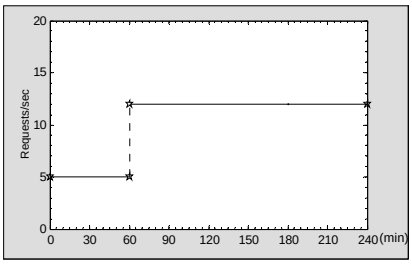

Fig. 1. Change in service request.

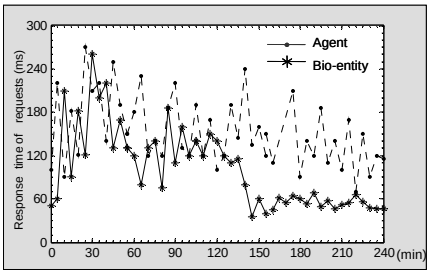

Fig. 2. Response time of requests.

In Fig. 2, response time represents the efficiency of Web service emergence. In all 240 minutes, response time slowly decreases with agent. While with agent, at the beginning, response time becomes very high to be 220ms. When the request of the WSES changes, With the time passing by, the agents incline to high effective Web services. Finally, the response time decreases dramatically, until it reaches the minimum value to be about 50 ms. Web service management will be self-evolution based on the mechanism of selecting the superior and eliminating the inferior.

The above simulation results show that WSES achieves built-in mechanisms to support some key features of

systemic systems such as adaptability, self-evolution, self-organization, and survivability.

IV. Conclusions and Further Work

Web service, as next generation promising infrastructure established in the Internet, has caused extensive attention from industry and academy circles around the world. The emphasis at next generation Web service composition shifts to dynamic self-composition. Requirements for these systems resemble the characteristics of systemic intelligent system. Inspired by this resemblance, we introduce the concept of Web service emergence. Then, the WSES is designed based on the mechanism of systemic intelligence. A novel method for automatic Web service composition and management is developed through Web services emergence. It exhibits desirable system characteristics such as self-organization, evolution, scalability, and adaptability.

The next work is on issues about improving the efficiency of both the agents' negotiation and the Web service emergence. Usually, the amount of the available Web services and the size of ontology models are huge. Therefore, it is necessary to reduce the search space during problem solving. In addition, more approaches will be developed to evaluate the WSES for automatic Web service composition and management..

References

[1] S. McIlraith, T. C. Son, and H. Zeng, Semantic Web services, IEEE Intelligent Systems 16(2) (2001) 46-53.

[2] S. Thakkar, C. Knoblock, and J. L. Ambite, A view integration approach to dynamic composition of Web services, in: Proceedings of ICAPS'03 Workshop on Planning for Web Services (Trento, Italy, 2003) 228-235.

[3] J. S. Hodas and D. Miller, Logic programming in a fragment of intuitionistic Linear Logic, Information and Computation 110(2) (1994) 327-365.

[4] T. J. Norman, A. Preece, S. Chalmers, N. R. Jennings, M. M. Luck, V. D. Dang, T. D. Nguyen, V. Deora, J. Shao, W. A. Gray, and N. J. Fiddian, Agent-based formation of virtual organizations, Knowledge-Based Systems 17(2-4) (2004) 103-111.

[5] Z. G. Hai, The future interconnection environment, IEEE Computer 38(4) (2005) 27-33.

[6] P. Küngas and M. Matskin, Symbolic negotiation with Linear Logic, Lecture Notes in Computer Science 3259 (Springer-Verlag, Berlin, 2004) 71-88.

[7] M. Winikoff and J. Harland, Implementing the Linear Logic programming language Lygon, in: Proceedings of the International Logic Programming Symposium (Portland, USA, 1995) 66-80.

Published by Atlantis Press, Paris, France.