A Survey of Automated Web Service Composition Methods

Jinghai Rao and Xiaomeng Su

Norwegian University of Science and Technology,
Department of Computer and Information Science,
N-7491, Trondheim, Norway
{jinghai, xiaomeng}@idi.ntnu.no

Abstract. In today's Web, Web services are created and updated on the fly. It's already beyond the human ability to analysis them and generate the composition plan manually. A number of approaches have been proposed to tackle that problem. Most of them are inspired by the researches in cross-enterprise workflow and AI planning. This paper gives an overview of recent research efforts of automatic Web service composition both from the workflow and AI planning research community.

1 Introduction

Web services are considered as self-contained, self-describing, modular applications that can be published, located, and invoked across the Web. Nowadays, an increasing amount of companies and organizations only implement their core business and outsource other application services over Internet. Thus, the ability to efficiently and effectively select and integrate inter-organizational and heterogeneous services on the Web at runtime is an important step towards the development of the Web service applications. In particular, if no single Web service can satisfy the functionality required by the user, there should be a possibility to combine existing services together in order to fulfill the request. This trend has triggered a considerable number of research efforts on the composition of Web services both in academia and in industry.

In the research related to Web services, several initiatives have been conducted with the intention to provide platforms and languages that will allow easy integration of heterogeneous systems. In particular, such languages as Universal Description, Discovery, and Integration (UDDI) [4], Web Services Description Language (WSDL) [9], Simple Object Access Protocol (SOAP) [6] and part of DAML-S [14] ontology (ServiceProfile and ServiceGrounding), define standard ways for service discovery, description and invocation (message passing). Some other initiatives such as Business Process Execution Language for Web Service (BPEL4WS) [2] and DAML-S ServiceModel, are focused on representing service compositions where flow of a process and bindings between services are known a priori.

J. Cardoso and A. Sheth (Eds.): SWSWPC 2004, LNCS 3387, pp. 43–54, 2005.

Despite all these efforts, the Web service composition still is a highly complex task, and it is already beyond the human capability to deal with the whole process manually. The complexity, in general, comes from the following sources. First, the number of services available over the Web increases dramatically during the recent years, and one can expect to have a huge Web service repository to be searched. Second, Web services can be created and updated on the fly, thus the composition system needs to detect the updating at runtime and the decision should be made based on the up to date information. Third, Web services can be developed by different organizations, which use different concept models to describe the services, however, there does not exist a unique language to define and evaluate the Web services in an identical means.

Therefore, building composite Web services with an automated or semi-automated tool is critical. To that end, several methods for this purpose have been proposed. In particular, most researches conducted fall in the realm of workflow composition or AI planning.

For the former, one can argue that, in many ways, a composite service is similar to a workflow [8]. The definition of a composite service includes a set of atomic services together with the control and data flow among the services. Similarly, a workflow has to specify the flow of work items. The current achievements on flexible workflow, automatic process adaption and cross-enterprise integration provide the means for automated Web services composition as well. In addition, the dynamic workflow methods provide the means to bind the abstract nodes with the concrete resources or services automatically.

On the other hand, dynamic composition methods are required to generate the plan automatically. Most methods in such category are related to AI planning and deductive theorem proving. The general assumption of such kind of methods is that each Web service can be specified by its preconditions and effects in the planning context. Firstly, a Web service is a software component that takes the input data and produces the output data. Thus the preconditions and effects are the input and the output parameters of the service respectively. Secondly, the Web service also alters the states of the world after its execution. So the world state pre-required for the service execution is the precondition, and new states generated after the execution is the effect. A typical example is a service for logging into a Web site. The input information is the username and password, and the output is a confirmation message. After the execution, the world state changes from "not_logged_in" to "logged_in". The "logged_in" state will be keeping until the "log_out" service is invoked. If the user can specify the preconditions and effects required by the composite service, a plan or process is generated automatically by logical theorem prover or AI planners without knowledge of predefined workflow. During the planning, the business logic can provide constraints in the planning setting.

In this paper we will present an overview of recent methods that provide automation to Web service composition. The automation means that either the method can generate the process model automatically, or the method can locate the correct services if an abstract process model is given. Some methods based on

workflow have been reported in the work by Benatallah [5], but to our knowledge, no overview on service composition methods related to AI planning has been published yet. As a result, in the paper, we will have more focus on the AI planning methods than the workflow based methods.

This paper is organized as follows. Section 2 presents an abstract framework for Web service composition. Section 3 is the introduction of automatic Web service composition based on workflow methods. Section 4 provides an overview and comparison for the selected composition methods based on AI planning. The last section concludes the paper.

2 Web Services Composition Framework

Here, we propose a general framework for automatic Web services composition. This framework is in high-level abstraction, without considering a particular language, platform or algorithm used in composition process. The aim of the framework is to give the basis to discuss similarities and differences of the available service composition methods. In addition, we also use the framework to unify the terms used in the paper.

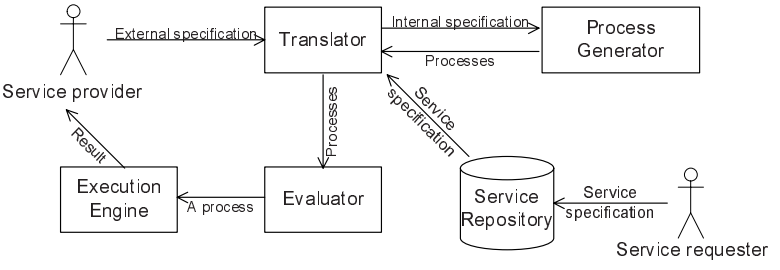

Fig. 1. The framework of the service composition system

A general framework of the service composition system is illustrated in Fig. 1. The composition system has two kinds of participants, service provider and service requester. The service providers propose Web services for use. The service requesters consume information or services offered by service providers. The system also contains the following components: translator, process generator, evaluator, execution engine and service repository. The translator translates between the external languages used by the participants and the internal languages used by the process generator. For each request, the process generator tries to generate a plan that composes the available services in the service repository to fulfill the request. If more than one plan is found, the evaluator evaluates all plans and proposes the best one for execution. The execution engine executes the plan and returns the result to the service provider.

Most precisely, the process of automatic service composition includes the following phases:

Presentation of Single Service: firstly, the service providers will advertise their atomic services at a global market place. There are several languages available for advertising, for example, UDDI [4] or DAML-S ServiceProfile [14]. The essential attributes to describe a Web service include the signature, states and the non-functional values. The signature is represented by the service's inputs, outputs and exceptions. It provides information about the data transformation during the execution of a Web service. The states are specified by precondition and postcondition. We model it as the transformation from one set of states to another in the world. Non-functionality values are those attributes that are used for evaluating the services, such as the cost, service quality and security issues.

Translation of the Languages: most service composition systems distinguish between the external and internal service specification languages. The external languages are used by the service users to enhance accessibility of the users in the sense that the users can express what they can offer or what they want in a relatively easy manner. They are usually different from the internal ones that are used by the composition process generator, because the process generator requires more formal and precise languages, for example, the logical programming languages. So far, the users have already get used to the standard Web service languages, such as WSDL and DAML-S. Thus the translation components between the standard Web service languages and the internal languages have to be developed.

Generation of Composition Process Model: in the meantime, the service requester can also express the requirement in a service specification language. A process generator then tries to solve the requirement by composing the atomic services advertised by the service providers. The process generator usually takes the functionalities of services as input, and outputs process model that describes the composite service. The process model contains a set of selected atomic services and the control flow and data flow among these services.

Evaluation of Composite Service: it is quite common that many services have the same or similar functionalities. So it is possible that the planer generates more than one composite service fulfilling the requirement. In that case, the composite services are evaluated by their overall utilities using the information provided from the non-functional attributes. The most commonly used method is utility functions. The requester should specify weights to each non-functionality attributes and the best composite service is the one who is ranked on top.

Execution of Composite Service: after a unique composite process is selected, the composite service is ready to be executed. Execution of a composite Web service can be thought as a sequence of message passing according to the process model. The dataflow of the composite service is defined as the actions that the output data of a former executed service transfers to the input of a later executed atomic service.

In the following we will give a survey on the methods used for the process generator to generate the process. The methods can be either fully automated or semi-automated.

3 Web Service Composition Using Workflow Technique

In the workflow-based composition methods, we should distinguish the static and dynamic workflow generation. The static one means that the requester should build an abstract process model before the composition planning starts. The abstract process model includes a set of tasks and their data dependency. Each task contains a query clause that is used to search the real atomic Web service to fulfill the task. In that case, only the selection and binding of atomic Web service is done automatically by program. The most commonly used static method is to specify the process model in graph. On the other hand, the dynamic composition both creates process model and selects atomic services automatically. This requires the requester to specify several constraints, including the dependency of atomic, the user's preference and so on.

EFlow[7] is a platform for the specification, enactment and management of composite services. EFlow uses a static workflow generation method. A composite service is modeled by a graph that defines the order of execution among the nodes in the process. The graph is created manually but it can be updated dynamically. The graph may include service, decision and event nodes. Service nodes represent the invocation of an atomic or composite service, decision nodes specify the alternatives and rules controlling the execution flow, and event nodes enable service processes to send and receive several types of events. Arcs in the graph denote the execution dependency among the nodes. Although the graph should be specified manually, EFlow provides the automation to bind the nodes with concrete services. The definition of a service node contains a search recipe that can be used to query actual service either at process instantiation time or at runtime. As the service node is started, the search recipe is executed, returning a reference to a specific service. In particular, the search recipe is resolved each time when a service node is activated. They do so because the availability of services may change very frequently in a highly dynamic environment. In [8], the authors further refine the service composition platform and propose a prototype of composite service definition language(CSDL). An interesting feature of CSDL is that it distinguishes between invocation of services and operations within a service. It provides the adaptive and dynamic features to cope with the rapidly evolving business and IT environment in which Web services are executed.

Polymorphic Process Model (PPM)[23] uses a method that combines the static and dynamic service composition. The static setting is supported by reference process-based multi-enterprise processes, the processes that consist of abstract subprocesses, i.e., subprocesses that have functionality description but lack implementation. The abstract subprocesses are implemented by service and bined at runtime. This is similar to the service binding in EFlow. The dynamic part of PPM is supported by service-based processes. Here, a service is modeled by a state machine that specifies that possible states of a service and their transitions. Transitions are caused by service operation(also called service activity) invocations or internal service transitions. In the setting, the dynamic service composition is enabled by the reasoning based on state machine.

4 Web Service Composition Using AI Planning

Many research efforts tackling Web service composition problem via AI planning have been reported. In general, a planning problem can be described as a five-tuple $\langle S, S_0, G, A, \Gamma \rangle$, where S is the set of all possible states of the world, $S_0 \subset S$ denotes the initial state of the world, $G \subset S$ denotes the goal state of the world the planning system attempts to reach, A is the set of actions the planner can perform in attempting to change one state to another state in the world, and the translation relation $\Gamma \subseteq S \times A \times S$ defines the precondition and effects for the execution of each action.

In the terms of Web services, S_0 and G are the initial states and the goal states specified in the requirement of Web service requesters. A is a set of available services. Γ further denotes the state change function of each service.

DAML-S (also called OWL-S in the most recent versions) is the only Web service language that announces the directly connection with AI planning. The state change produced by the execution of the service is specified through the precondition and effect properties of the ServiceProfile in DAML-S. Precondition presents logical conditions that should be satisfied prior to the service being requested. Effects are the result of the successful execution of a service. Since DAML+OIL, the language used to build DAML-S, uses Description Logics [10] as its logical foundation, DAML+OIL has the express power allowing for logical expressions. The majority of the methods reported in this survey use DAML-S as the external Web service description language. There are also a couple of methods that use WSDL or their own languages.

In the following we introduces a list of Web service composition methods based on AI planning. This kind of methods have been reported frequently in recent years, so we can not claim that we have an exhaustive list of the methods. We further classify the methods into five categories, namely, the situation calculus, the Planning Domain Definition Language (PDDL), rule-based planning, the theorem proving and others.

4.1 Situation Calculus

McIlraith et. al. [17, 19, 16] adapt and extend the Golog language for automatic construction of Web services. Golog is a logic programming language built on top of the situation calculus. The authors address the Web service composition problem through the provision of high-level generic procedures and customizing constraints. Golog is adopted as a natural formalism for representing and reasoning about this problem.

The general idea of this method is that software agents could reason about Web services to perform automatic Web service discovery, execution, composition and inter-operation. The user's request (generic procedure) and constraints can be presented by the first-order language of the situation calculus(a logical language for reasoning about action and change). The authors conceive each Web service as an action - either a *PrimitiveAction* or a *ComplexAction*. Primitive actions are conceived as either world-altering actions that change the state

of the world or information-gathering actions that change the agent's state of knowledge. Complex actions are compositions of individual actions. The agent knowledge base provides a logical encoding of the preconditions and effects of the Web service actions in the language of the situation calculus. The agents use procedural programming language constructs composed with concepts defined for the services and constraints using deductive machinery. A composite service is a set of atomic services which connected by procedural programming language constructs(if-then-else, while and so forth).

The authors also propose a way to customize Golog programs by incorporating the service requester's constraints. For example, the service requester can use the nondeterministic choice to present which action is selected in a given situation, or use the sequence construct to enforce the execution order between two action. The generation of the plan have to obey the predefined constraint.

4.2 PDDL

A strong interest to Web service composition from AI planning community could be explained roughly by similarity between DAML-S and PDDL representations. PDDL is widely recognized as a standardized input for state-of-the-art planners. Moreover, since DAML-S has been strongly influenced by PDDL language, mapping from one representation to another is straightforward (as long as only declarative information is considered). When planning for service composition is needed, DAML-S descriptions could be translated to PDDL format. Then different planners could be exploited for further service synthesis.

In presenting the Web service composition method based on PDDL, McDermott [15] introduces a new type of knowledge, called *value of an action*, which persists and which is not treated as a truth literal. From Web service construction perspective, the feature enables us to distinguish the information transformation and the state change produced by the execution of the service. The information, which is presented by the input/output parameters are thought to be reusable, thus the data values can be duplicated for the execution of multiple services. Contrarily, the states of the world are changed by the service execution. We interpret the change as that the old states disappear and the new states are produced.

To deal with this issue is critical for Web service composition using AI planning because usually in AI planning, closed world assumption is made, meaning that if a literal does not exist in the current world, its truth value is considered to be *false*. In logic programming this approach is called *negation as failure*. The main trouble with the closed world assumption, from Web service construction perspectives, is that merely with truth literals we cannot express that new information has been acquired. For instance, one service requester might want to describe that after sending a message to a Web service, an identity number to the message will be generated. Thus during later communication the ID could be used.

4.3 Rule-Based Planning

Medjahed [18] present a technique to generate composite services from high-level declarative description. The method uses composability rules to determine whether two services are composable. The composition approach consists of four phases. First, the specification phase enables high-level description of the desired compositions using a language called Composite Service Specification Language(CSSL). Second, the matchmaking phase uses composability rules to generate composition plans that conform to service requester's specifications. The third phase is selection phase. If more than one plan is generated, in the selection phase, the service requester selects a plan based on quality of composition (QoC) parameters (e.g. rank, cost, etc.). The final phase is the generation phase. A detailed description of the composite service is automatically generated and presented to the service requester.

Here, we should pay more emphasis on the composability rules because it is the major issue to define how the plan is generated. The composability rules consider the syntactic and semantic properties of Web services. Syntactic rules include the rules for operation modes and the rules for binding protocols of interacting services. Semantic rules include the following subset: (1) message composability defines that two Web services are composable only if the output message of one service is compatible with the input message of another service; (2) operation semantic composability defines the compatibility between the domains, categories and purposes of two services; (3) qualitative composability defines the requester's preferences regarding the quality of operations for the composite service; and (4) composition soundness considers whether a composition of services is reasonable. To this end, the authors introduce the notion of composition templates that define the dependency between the different kinds of services.

The main contribution of this method is the composability rules, because they define the possible Web service's attributes that could be used in service composition. Those rules can be used as a guideline for other Web service methods based on planning.

SWORD [20] is another developer toolkit for building composite Web services using rule-based plan generation. SWORD does not deploy the emerging service-description standards such as WSDL and DAML-S, instead, it uses Entity-Relation (ER) model to specify the Web services. In SWORD, a service is modeled by its preconditions and postconditions. They are specified in a world model that consists of entities and relationships among entities. A Web service is represented in the form of a Horn rule that denotes the postconditions are achieve if the preconditions are true. To create a composite service, the service requester only needs specify the initial and final states for the composite service, then the plan generation can be achieved using a rule-based expert system. Besides the general composition methods, an interesting work done by SWORD is that the authors give a discussion on that the rule-based chaining can sometimes generate "uncertain" results if a precondition can not uniquely determines a postcondition. The authors argue that the uncertain results can avoid only

when the preconditions are functionally depending on the postconditions inside a service. In fact, it may happen in most service composition methods described in this survey but not all authors explicitly declare it.

4.4 Other AI-Planning Methods

Some other AI planning techniques are proposed for the automatic composition of Web services. In [26] the SHOP2 planner is applied for automatic composition of Web services, which are provided with DAML-S descriptions. SHOP2 is an Hierarchical Task Network(HTN) planner. The authors believe that the concept of task decomposition in HTN planning is very similar to the concept of composite process decomposition in DAML-S process ontology. The authors also claim that the HTN planner is more efficient than other planning language, such as Golog. In their paper, the authors give a very detail description on the process of translating DAML-S to SHOP2. In particular, most control constructs can be expressed by SHOP2 in an explicit way.

Sirin et al [24] present a semi-automatic method for web service composition. Each time when a user has to select a Web service, all possible services, that match with the selected service, are presented to the user. The choice of the possible services is based both on functionalities and non-functional attributes. The functionalities (parameters) are presented by OWL classes and OWL reasoner is applied to match the services. A match is defined between two services that an output parameter of one service is the same OWL class or subclass of an input parameter of another service. The OWL inference engine can order the matched services so that the priority of the matches are lowered when the distance between the two types in the ontology tree increases. If more than one match is found, the system filters the services based on the non-functional attributes that are specified by the user as constraints. Only those services who pass the non-functional constraints can be presented to the service requester. The idea of semi-automatic service composition is quite interesting because it is very difficult to capture behavior in sufficient detail and compose the services in a fully automatic way, especially for the commercial-grade services. Although the proposed method is simple, it indicates the trend that automatic planner and human being can work together to generate the composite service for the user's request.

4.5 Theorem Proving

Waldinger [25] elaborates an idea for service synthesis by theorem proving. The approach is based on automated deduction and program synthesis and has its roots in his earlier work [13]. Initially available services and user requirements are described in a first-order language, related to classical logic, and then constructive proofs are generated with SNARK theorem prover. Finally, service composition descriptions are extracted from particular proofs.

Lämmermann [12] applies Structural Synthesis of Program (SSP) for automated service composition. SSP is a deductive approach to synthesis of programs

from specifications. The specifications of services only include the structural properties, i.e. the input/output information. SSP uses propositional variables as identifiers for input/output parameters and uses intuitionistic propositional logic for solving the composition problem. The composition is based on the proofs-as-programs property of intuitionistic logic. It equates the program of service composition to the problem of proof search. The author also takes takes advantage of disjunctions in classical logic to describe exceptions, which could be thrown during service invocation.

Rao et. al. [21, 22] introduces a method for automatic composition of semantic Web services using Linear Logic theorem proving. The method uses semantic Web service language (DAML-S) for external presentation of web services. And, internally, the services are presented by extralogical axioms and proofs in Linear Logic. Linear Logic, as a resource conscious logic, enables people to define attributes of Web services formally (including qualitative and quantitative values of non-functional attributes). In addition, Linear Logic has close relationship with π-calculus, which is the formal foundation of many Web service composition languages. The view of a Linear Logic proof as a π-calculus process was firstly taken up formally by Abramsky [1], and further elaborated by Bellin and Scott [3]. The authors attach the π-calculus to the Linear Logic inference rules in the style of type theory, thus the process model for a composite service presented by π-calculus can be generated directly from the proof. The authors also present the subtyping rules that are used for semantic reasoning with LL inference figures. Thus the Linear Logic theorem prover can deal with both the service specification and the semantic Web information. Unlike other methods that use non-functional attributes only to filter the generated plan, the authors consider the non-functional attributes directly in the theorem proving process. Both service functionalities and non-functional attributes are translated into propositions in the logical axioms, but the distinguish between the functionalities and non-functional attributes is enabled by the Linear Logic inference rules.

5 Conclusion

This paper has aimed to give an overview of recent progress in automatic Web services composition. At first, we propose a five-step model for Web services composition process. The composition model consists of service presentation, translation, process generation, evaluation and execution. Each step requires different languages, platforms and methods.

In these five steps, we concentrate on the methods of composite Web services process generation. We give the introduction and comparition of selected methods to support this step. The methods are enabled either by workflow research or AI planning. The workflow methods are mostly used in the situation where the request has already defined the process model, but automatic program is required to find the atomic services to fulfill the requirement. The AI planning methods is used when the requester has no process model but has a

set of constraints and preferences. Hence the process model can be generated automatically by the program.

Although the different methods provide different level of automation in service composition, we can not say the higher automation the better. Because the Web service environment is highly complex and it is not feasible to generate everything in an automatic way. Usually, the highly automated methods is suitable for generating the implementation skeletons that can be refined into formal specification. A discussion on this topic is presented by Hull et. al. [11].

Further work will include a more thorough analysis of the field in addition to practical testing of and experiments with the methods.

References

1. S. Abramsky. Proofs as processes. *Theoretical Computer Science*, 135(1):5–9, 1994.
2. T. Andrews et al. Business Process Execution Language for Web Services (BPEL4WS) 1.1. Online: http://www-106.ibm.com/developerworks/webservices/library/ws-bpel, May 2003.
3. G. Bellin and P. J. Scott. On the pi-calculus and Linear Logic. *Theoretical Computer Science*, 135(1):11–65, 1994.
4. T. Bellwood et al. Universal Description, Discovery and Integration specification (UDDI) 3.0. Online: http://uddi.org/pubs/uddi-v3.00-published-20020719.htm.
5. B. Benatallah, M. Dumas, M.-C. Fauvet, and F. Rabhi. *Patterns and skeletons for parallel and distributed computing*, chapter Towards Patterns of Web Services Composition, pages 265–296. Springer-Verlag, 2003.
6. D. Box et al. Simple Object Access Protocol (SOAP) 1.1. Online: http://www.w3.org/TR/SOAP/, 2001.
7. F. Casati, S. Ilnicki, and L. Jin. Adaptive and dynamic service composition in EFlow. In *Proceedings of 12th International Conference on Advanced Information Systems Engineering(CAiSE)*, Stockholm, Sweden, June 2000. Springer Verlag.
8. F. Casati, M. Sayal, and M.-C. Shan. Developing e-services for composing e-services. In *Proceedings of 13th International Conference on Advanced Information Systems Engineering(CAiSE)*, Interlaken, Switzerland, June 2001. Springer Verlag.
9. R. Chinnici et al. Web Services Description Language (WSDL) 1.2. Online: http://www.w3.org/TR/wsdl/.
10. B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description Logic programs: Combining logic programs with Description Logic. In *Proceedings of the 12th International Conference on the World Wide Web (WWW 2003*, Budapest, Hungary, 2003.
11. R. Hull, M. Benedikt, V. Christophides, and J. Su. E-service: A look behind the curtain. In *Proceedings of the 22nd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, San Diego, USA, June 2003.
12. S. Lämmermann. *Runtime Service Composition via Logic-Based Program Synthesis*. PhD thesis, Department of Microelectronics and Information Technology, Royal Institute of Technology, June 2002.
13. Z. Manna and R. J. Waldinger. A deductive approach to program synthesis. *ACM Transactions on Programming Languages and Systems*, 2(1):90–121, 1980.
14. D. Martin et al. DAML-S(and OWL-S) 0.9 draft release. Online: http://www.daml.org/services/daml-s/0.9/, May 2003.

15. D. McDermott. Estimated-regression planning for interactions with Web services. In *Proceedings of the 6th International Conference on AI Planning and Scheduling*, Toulouse, France, 2002. AAAI Press.
16. S. McIlraith and T. C. Son. Adapting Golog for composition of Semantic Web services. In *Proceedings of the 8th International Conference on Knowledge Representation and Reasoning(KR2002)*, Toulouse, France, April 2002.
17. S. McIlraith, T. C. Son, and H. Zeng. Semantic Web services. *IEEE Intelligent Systems*, 16(2):46–53, March/April 2001.
18. B. Medjahed, A. Bouguettaya, and A. K. Elmagarmid. Composing Web services on the Semantic Web. *The VLDB Journal*, 12(4), November 2003.
19. S. Narayanan and S. McIlraith. Simulation, verification and automated composition of Web service. In *Proceedings of the 11th International World Wide Web Conference*, Honolulu, Hawaii, USA, May 2002. ACM. presentation available at http://www2002.org/presentations/narayanan.pdf.
20. S. R. Ponnekanti and A. Fox. SWORD: A developer toolkit for Web service composition. In *Proceedings of the 11th World Wide Web Conference*, Honolulu, HI, USA, 2002.
21. J. Rao, P. Küngas, and M. Matskin. Application of Linear Logic to Web service composition. In *Proceedings of the 1st International Conference on Web Services*, Las Vegas, USA, June 2003.
22. J. Rao, P. Küngas, and M. Matskin. Logic-based Web services composition: from service description to process model. In *Proceedings of the 2004 International Conference on Web Services*, San Diego, USA, July 2004. IEEE.
23. H. Schuster, D. Georgakopoulos, A. Cichocki, and D. Baker. Modeling and composing service-based and reference process-based multi-enterprise processes. In *Proceeding of 12th International Conference on Advanced Information Systems Engineering (CAiSE)*, Stockholm, Sweden, June 2000. Springer Verlag.
24. E. Sirin, J. Hendler, and B. Parsia. Semi-automatic composition of Web services using semantic descriptions. In *Proceedings of Web Services: Modeling, Architecture and Infrastructure workshop in conjunction with ICEIS2003*, 2002.
25. R. Waldinger. Web agents cooperating deductively. In *Proceedings of FAABS 2000, Greenbelt, MD, USA, April 5–7, 2000*, volume 1871 of *Lecture Notes in Computer Science*, pages 250–262. Springer-Verlag, 2001.
26. D. Wu, E. Sirin, J. Hendler, D. Nau, and B. Parsia. Automatic Web services composition using SHOP2. In *Workshop on Planning for Web Services*, Trento, Italy, June 2003.