AI Planning in Web Services Composition: a review of current approaches and a new solution

Luciano A. Digiampietri[1], José J. Pérez-Alcázar[2] , Claudia Bauzer Medeiros[1]

[1]Institute of Computing, University of Campinas
CP 6176, 13084-971, Campinas, SP, Brazil

[2]EACH, University of São Paulo, 03828-000, São Paulo, SP, Brazil,

{luciano,cmbm}@ic.unicamp.br,jperez@usp.br

Abstract. *Web services represent a relevant technology for interoperability. An important step toward the development of applications based on Web services is the ability of selecting and integrating heterogeneous services from different sites. When there is no single service capable of performing a given task, there must be some way to adequately compose basic services to execute this task. The manual composition of Web services is complex and susceptible to errors because of the dynamic behavior and flexibility of the Web. This paper describes and compares AI planning solutions to Web service automatic composition. As a result of this comparison, it proposes an architecture that supports service composition, and which combines AI planning with workflow mechanisms.*

1. Introduction

A Web Service (WS), according to W3C's work group on Web Services Architecture [Austin et al. 2002], is "a software system designed to support interoperable machine-to-machine interaction over a network". It provides a systematic and extensible framework for application-to-application interaction, built on top of existing Web protocols and based on open XML standards [W3C 2003]. Depending on the kind of service provided, and user needs, one can analyze Web Services (WS) under the software engineering framework of software components. Therefore, a Web service must: possess a description which allows other Web services to understand its functionality and how to use it; allow other Web services to use it whenever necessary; and be easy to invoke. These characteristics and the use of XML based technologies promote the reuse and the interoperability of the applications based in Web services.

An important characteristic required for applications based on Web Services is the ability to efficiently select and integrate, at runtime, heterogeneous services, from different companies. This has led to efforts toward helping service selection and composition. One such effort is the specification of composition mechanisms such as: WSBPEL ("Web Services Business Process Execution Language") [Andrews et al. 2003], WSCI ("Web Service Choreography Interface") and others. Any such mechanism requires manual work to allow process construction and integration. This can be a laborious task and susceptible to mistakes.

The notion of a "Semantic Web" appeared to face some of these problems. The Semantic Web allows data and services to be interpreted automatically by software agents, without ambiguity problems [Berners-Lee et al. 2001]. Current description languages

(such as WSDL or WSBPEL) must be extended, or new languages must be created to allow Web Services to be described in a way that is understandable by computers. Some approaches try to apply semantics to Web Services technology. Examples are the use of OWL-S [Martin et al. 2004] and the work developed by the WSMF group ("Web Service Modeling Framework") [Fensel and Bussler 2002], to develop a flexible and scalable platform for the development of workflows based on Web Services.

The main idea behind our proposal is that the problem of automatic or semi-automatic composition of workflow tasks can be seen as an Artificial Intelligence planning problem. This approach has become interesting due to the maturity that the planning area has achieved in AI [Long and Fox 2003].

We furthermore consider Web Services in the context of workflows (i.e., the problem of service composition is seen as a problem of workflow design, where workflow tasks invoke services). The paper thus attacks the problem of constructing workflows, under the assumption that they are the basis for specifying and executing tasks in a distributed environment. The execution of each activity within such a workflow can be executed either by invocation of a Web service or of another (sub)workflow. Thus we use the terms "service composition" and "workflow composition/construction" interchangeably.

The main contributions are: (i) comparing the state of the art of AI planning solutions applied to Web service composition; (ii) based in this comparison, discussing our solution to the problem of Web service composition combining results from AI and database systems, thereby providing mechanisms for automatic, semi-automatic and manual composition. This solution has been implemented in a prototype, applied to bioinformatics [Digiampietri et al. 2007].

The rest of this paper is organized as follows. Section 2 describes related work and concepts. Section 3 describes AI planning solutions that can be exploited in supporting workflow composition. Section 4 presents the proposed architecture and our prototype. Section 5 contains conclusions and ongoing work.

2. Basic concepts

We consider three main kind of service composition: manual, iterative and automatic. In *manual composition*, the user chooses the services that he/she wants to use and the links between the services interfaces. In the *iterative composition*, the user starts the composition from an abstract activity and, iteratively, decomposes it into more specific activities until only concrete activities (services) are defined. *Automatic composition* is the process where the user's request is a goal and the system automatically composes services to achieve this goal.

We focus, in this paper, on the automatic composition problem. Several solutions have proposed to solve the problem of automatic composition. AI planning is a well known area but there are challenges in applying its planning techniques to Web service composition. Section 3 presents a comparative analysis of mainstream proposals, which we compare under criteria discussed in the next two paragraphs.

According to [Srivastava and Koehler 2003], in order to use planning in the automatic composition of Web services, AI planning concepts must be extended to consider the following characteristics: (i) plans need complex control structures with loops, non-

determinism and conditionals; (ii) the objects managed may have a complex structure and description; and (iii) plans can produce new objects at execution time.

We highlight other important characteristics, not usually found in AI planning. One is the use of non-functional attributes, such as cost or quality, which can facilitate the choice of the plan most adequate to the user's needs when there are several plans with similar functionalities (plan selection). Moreover, plans need to support semantic constructions such as hierarchies (abstractions), as well as compatibility with the different Semantic Web service description standards, such as OWL-S (www.daml.org/services) and WSMO (www.wsmo.org). Furthermore, plans also must consider the definition of extended goals involving complex conditions on process behavior. Still other characteristics needed in service composition but not found in planning include concurrency in service access, use of Web standards and scalability.

3. AI Planning applied to services

This section discusses classes of planning systems and proposals that can be considered to compose Web services. We compare these proposals to justify our workflow construction strategy.

3.1. Overview of proposals

Proposal using Golog. McIlraith and Son [McIlraith and Son 2002] adopted and extended Golog [Levesque et al. 1997] to allow the automatic building of Web services. The authors treat the Web service composition problem through generic procedures (predefined "plan templates") and restrictions customized by the user.

PDDL based proposal. PDDL is a widely used formal language created by McDermott [Ghallab et al. 1998] to describe different kinds of planning problems. McDermott [McDermott 2002] uses PDDL to specify plans, but extends the language to introduce a new kind of knowledge, called "value of the action", that represents the passage of information among the steps of a plan. The planner used is regression based, and allows the generation of conditional plans with ramifications.

Hierarchical planning. Hierarchical planning is an AI planning methodology that creates plans by task decomposition. Well-known hierarchical planners are SHOP and SHOP2 (Simple Hierarchical Ordered Planner) [Nau et al. 2003].

One system based in this proposal is ISP&E (Integrated Service Planning and Execution) [Madhusudan and Uttamsingh 2006], which uses SHOP. It generates several alternatives of solution and one of the alternative plans is selected using a generic notion of cost. This plan is executed and changes in the execution environment are checked periodically. When a failure happens, the plan is re-executed or, if the failure continues, a mechanism of re-planning is triggered.

Sirin et al [Sirin et al. 2004] use SHOP2 for the automatic composition of Web services. The inputs of their planner are specified in OWL-S. SHOP2 uses the concept of methods to decompose a task in sub-tasks. These methods can contain explicit actions for monitoring that allow the planner to obtain the necessary data to treat problems of incomplete information. SHOP2 can implement a kind of extended goal through the specification of composite tasks (methods) that describe the changes required

by the users. SHOP2 has demonstrated good results when using a great amount of methods and operators. It was, recently, extended to deal with non-determinism (ND-SHOP2) [Kuter and Nau 2004].

Kim and Gil's proposal. Kim and Gil [Kim and Gil 2004] argues that because Web service composition is a complex application, it requires interaction with users. So, they developed interactive tools for composing Web services where users define a high-level or partial/incomplete description of the desired composition of services and the system assists the users by providing intelligent suggestions. These suggestions are supported by service and domain ontologies. This proposal generates only linear compositions (without cycles and conditions).

Synthy. Agarwal et al [Agarwal et al. 2005] proposed a system called Synthy to solve the problem of automatic composition of Web services. This system is based in a two-staged approach that, according to the authors, addresses the information modeling aspects of Web services, provides support for contextual information during the composition process, employs efficient decoupling of functional and non-functional requirements, and leads to improve scalability and failure handling. The first stage of composition, called logic composition, generates abstract workflows (plans using service types). To do this, the system uses planning and matchmaking techniques. The second stage, called physical composition, generates executable workflows that use only service instances. The system uses optimization techniques to select the best instance associated to the service types. This approach uses a extension of OWL-S to support service types, allowing to work with big collections of Web services and with the two-stage composition.

Symbolic Model Checking (SMC) based planning. Model Checking is a formal method often used in verification of complex hardware and software systems. Traverso and Pistore [Traverso and Pistore 2004] use an approach based in a SMC for automatic composition of services described in OWL-S, called MBP (Model Based Planner). This technique has presented good practical results for the problem of planning with non-deterministic actions, partial observations of the environment, complex goals and domains (very large space of states). One of its problems is that it does not explore the hierarchical and taxonomical aspects of OWL-S.

3.2. Comparison of proposals

Table 1 summarizes the comparison of the proposals. The first column contains the requirements for planning using Web services that we pointed out in Section 2. The other columns cover the main approaches of planning systems reviewed.

Only SHOP2, MBP and Synthy use a semantic Web service description standard (OWL-S). Synthy and [Kim and Gil 2004] use a domain ontology that, associated to the service ontology, allow the representation of complex objects, the dynamic creation of objects and the generation of the dataflow associated to the composite Web service. SHOP2, Synthy and the proposal of [Kim and Gil 2004] make use of abstractions and hierarchies to represent services. However, only Synthy and [Kim and Gil 2004] make use of specializations and generalizations in their composition process.

The proposals based in Golog, McDermott's approach and Synthy use conditional plans. These proposals are limited in dynamic domains because conditional planning supposes that every failure can be known before plan execution. Thus, they only partially

	Golog	PDDL	ISP&E	Kim&Gil	Synthy	SHOP2	MBP
Use of standard	Y	Y	Y	Y	Y	Y	Y
Complex objects	N	Y	N	Y	Y	N	N
Abstraction / Hierarchy	N	N	Y	Y	Y	Y	N
Non-determinism / partial obs. of the world	Y	Y	N	N	Y; partially	Y ; ND-SHOP2	Y
Generation of non-linear plans	Y	Y; partially	NM	N	Y; partially	Y; partially	Y
Automation level	SA	A	A	SA	A, SA	A	A
Plan selection	N	N	Y; partially	N	Y	N	N
Concurrency	Y	N	Y	NM	NM	N	Y
Scalability	NM	NM	NM	NM	G	G	G
Extended goals	Y	N	NM	NM	NM	Y	Y

N → No; Y → Yes; SA → Semi-automatic; A → Automatic
NM → Not mentioned; W → Weak; G → Good.

Table 1. Comparison among planner proposals

solve the problems of non-determinism and partial observability. ISP&E uses re-planning to solve non-determinism problems. The solution using SMC is the most general. Other proposals, such as Kim and Gil's, do not mention non-determinism. MBP is the best alternative with respect to the generation of non-linear plans. Other proposals only allow conditional plans (Synthy, PDDL). The work of [Sirin et al. 2004] treats Web service composition as a SHOP2 planning problem. They encode composite tasks within SHOP2's methods, allowing the representation of iterations and conditional processes.

Synthy is the only proposal that allows automatic and semi-automatic composition and that uses non-functional attributes (allowing QoS features) in the process of composition. ISP&E mentions the use of optimization in plan selection, but in a simplified way.

We also observe in Table 1 that none of the reviewed solutions covers all issues stated in column 1. Planning using Web services is a recent research area and the majority of these projects are at an experimental stage. For this reason, we propose a framework that collects the best characteristics of the proposals analyzed. In our prototype, we decided to use SHOP2 for planning, because it provides the following benefits: (a) it enables embedding domain knowledge to control the search space and improve efficiency; (b) it has been successfully used in a variety of real-world planning-based applications; (c) it allows inclusion of different types of precondition constraints for service operators as well as calls to external systems; (d) it enables modeling process abstractions in terms of method/operator hierarchies; and (e) it enables reuse by facilitating selection of appropriate methods from domain-related operator libraries. Moreover, we combined SHOP2 with others approaches, such as the two stage composition (from Synthy), the iterative composition supported by ontologies (from Kim and Gil) and re-planning (from ISP&E).

4. An architecture for automatic composition via planning

This section outlines our general architecture for composition of Web services. We recall that we transform the problem of service composition into that of constructing a workflow that invokes these services. Our plans are consequently specifications of scientific workflows. Thus, in this section, we will use indistinctly both terms (plans and workflows).

4.1. Architecture overview

Figure 1 shows our architecture, highlighting the main modules, repositories and their interactions. It extends the framework defined by Rao and Su [Rao and Su 2004]. This

architecture can be used in different application domains. Section 4.6 discusses one particular implementation, for bioinformatics.

Repositories are key concepts to allow semantic composition and annotation of the services. The entire workflow design and execution process is based on combining planning techniques with information stored in three repositories: (1) **Ontology Repository:** contains the domain and service ontologies that will be used to allow automatic composition and annotation of services and workflows; (2) **Service Catalog:** plays the role of a UDDI registry with extended functionalities to allow the storing of service non-functional attributes, such as execution time, reliability and availability. While the Ontology Repository stores information about domain and service types, the Service Catalog stores information about service instances; and (3) **Workflow Repository:** stores annotated (sub)workflows. A workflow can range from a simple atomic task to a complex specification (including services, produced data and annotations). Beyond storing only workflow specifications, the Workflow Repository also stores execution data. It includes the resulting data of each step of workflow execution and the metadata associated with this execution. Instances in the Service Catalog and in the Workflow Repository are annotated with terms from the Ontology Repository, which qualify the corresponding types.

The user interacts with the architecture via the Interface - Workflow Editor. It allows the user to design, search and edit workflows. It also allows a user to request the execution of a workflow and interact with this execution.

This paper is mainly concerned with iterative and automatic composition through AI planning. We recall that in these cases (especially in the latter) the process starts from an abstract specification and proceeds to a concrete (executable) one - in the text, these specifications will be also called abstract and concrete workflows. The Workflow Repository stores workflows at all these levels, to support distinct levels of workflow design and reuse of workflow components [Medeiros et al. 2005].

4.2. Manual Composition

In manual composition, the user interacts with the Service Discovery module through the Interface - Workflow Editor, (Figure 1 (1)). This module goes to the Service Catalog, Workflow Repository and the Ontology Repository (2), and lists available services (3). The user can set the Service Discovery to lists only service instances (Web services or concrete workflows). Whenever the user modifies a workflow using manual composition, the Checker module can be called (4) to verify the workflow. This verification is made based on the semantic compatibility of the service interfaces (5) and (6). The Checker send warnings and suggestions to the user (7).

4.3. Iterative Composition

Iterative composition (Figure 1) looks at first glance like manual composition. In the first iteration, the user request services to Service Discovery (1) that goes to the Ontology Repository (2) and returns abstract services (3). The great difference is that instead of just selecting and composing services the user will progressively decompose services into more concrete specifications. To allow this top down specification, at each iteration, the Service Discovery lists (3) the services that decompose a given abstract service. The user starts from a very abstract workflow and end up with a concrete (executable) workflow. Every step of the iterative composition is also verified by the Checker (4) to (7).

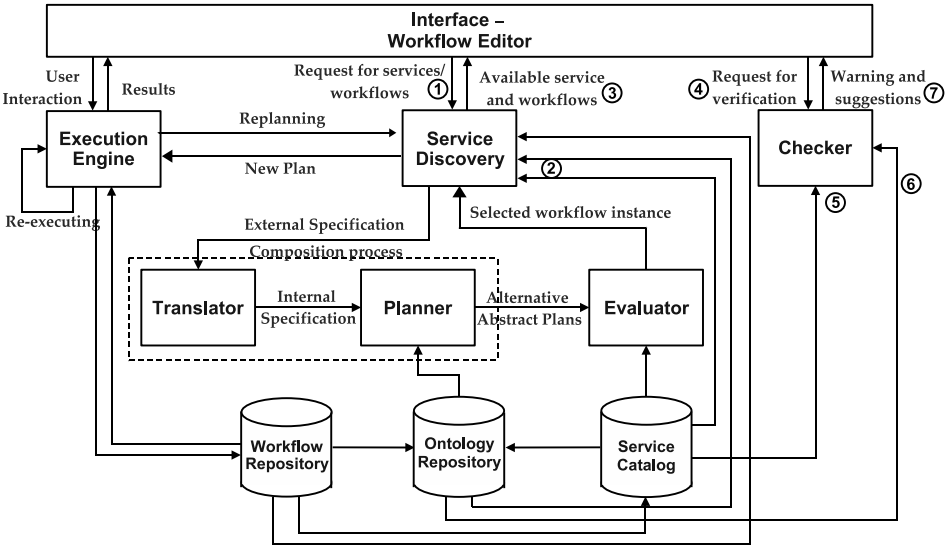

Figure 1. Manual and Iterative Composition

4.4. Automatic Composition

Figure 2 highlights the modules and messages involved in the automatic composition process. This process starts when a user (human or software agent) interacts with Interface requesting a service. This request is passed to the Service Discovery Module (1). This module tries to find a service (simple Web service or a workflow) in the repositories (2) that suits the request. The search is based on a matchmaking algorithm. If one or more services are found, they are sent to the Interface - Workflow Editor (3). If no service is found, the request is sent to the Translator module (4) that is responsible for translating the request to the Planner language (5). The Planner Module, in our prototype an extension of SHOP2, combined with the approaches mentioned in Section 3.2, generates alternative plans (abstract workflows) that meet the request. In order to generate these plans, additional data must be retrieved from the Ontology Repository (6). The abstract plans contain only service types. These abstract plans are sent to the Evaluator (7), where the types are instantiated by actual services that best suit each service type in order to obtain the most efficient plan. The Evaluator module uses non-functional service attributes, stored in the Service Catalog (8), to rank the plans through the use of Quality of Service (QoS) techniques [Zeng et al. 2004]. The highest ranked plan (closest to user goals) is sent to Service Discovery (9) and forwarded to Interface - Workflow Editor (10).

4.5. Workflow Execution

Workflow execution has not yet been implemented, and this section describes how this is integrated into the architecture. Its implementation requires coupling the architecture with some worflow engine, e.g. [WfMOpen , Altintas et al. 2004]. The Execution Engine is responsible for plan (workflow) execution. Workflow execution can occur for any kind of composition. Figure 2 contains the messages exchanged in the execution process. This process starts when a user requests the execution of the current workflow using Interface - Workflow Editor (i). Every piece of data produced during the execution is stored in the Workflow Repository (ii). Whenever a fault is detected, during the execution, the Engine

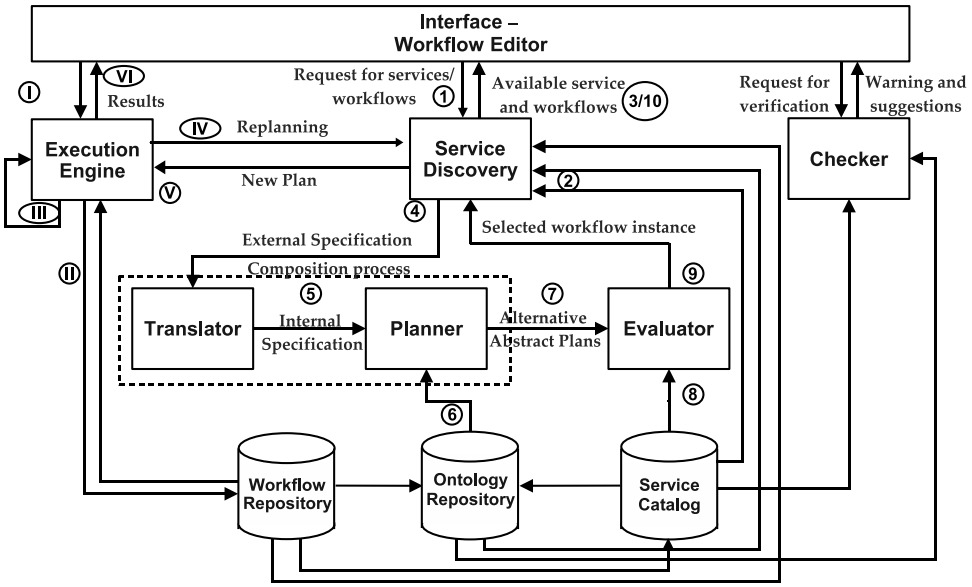

Figure 2. Automatic Composition and Execution

tries to re-execute (iii) the plan (considering the actual state of the world). This kind of solution is adequate when a Web service is temporarily unavailable. If the fault persists, it tries to re-plan starting from the state of the world after the failure. In this process, the Execution Engine sends a request (iv) to Service Discovery for a new plan, which is returned to the Engine (v), to continue the execution of the workflow. The user interacts with the Execution Engine through the Interface and can change the execution flow, pause, cancel and restart the execution and see the intermediary results (i). The results are passed from the Execution Engine to the user through the Interface - Workflow Editor (vi).

4.6. Case Study: genome assembly and annotation

We implemented a prototype of our architecture to solve the problems of genome assembly and annotation [Digiampietri et al. 2005]. Currently, we are using SHOP2 as planner. Our system allows the three kinds of composition in order to support scientific workflows.

In order to implement the architecture we had to construct the appropriate ontologies. In particular, we have developed a detailed ontology, specific to genome assembly and annotation, that extends a generic bioinformatics ontology [Stevens et al. 2000]. Through our ontology we annotated bioinformatics data and tools/services in order to allow semantic search and automatic composition of services. A detailed description of the prototype is shown in [Digiampietri et al. 2007].

5. Conclusions and ongoing work

This paper reviewed some of the main proposals in service composition. It presented a new proposal that supports automatic and interactive service composition through the use of AI planning. Service search and composition take advantage of the use of ontology semantic annotations. We built a prototype to verify and validate our proposal, for bioinformatics problems, specifically for genome assembly and annotation [Digiampietri et al. 2007]. The choice of this specific area was made due to our experience with these tasks and the great need of this kind of system in bioinformatics.

Our main contributions lie in the comparative study, and in proposing and prototyping a solution for specifying scientific workflows in the Web by taking advantage of AI planning techniques, combined with ontologies and Semantic Web standards. Our architecture is specified in a generic way, and thus can be utilized to solve any problem that involves the storage, coordinated execution and automatic composition of scientific and business processes. It combines the main advantages of several approaches for Web service composition based in AI Planning.

As future work we intend to explore other promising ways for plan synthesis, such as combining SHOP2 with Symbolic Model Checking [Kuter et al. 2005]. We also plan to study the use of plan repair techniques to decrease the need for re-planning. Finally, we have in mind to couple our prototype with a workflow engine, to validate our proposal of on-the-fly workflow reconstruction using our planning strategy.

Acknowledgments

The work described in this paper was partially financed by CAPES, FAPESP, Microsoft Research fellowship, CNPq and Autonomous University of Bucaramanga.

References

Agarwal, V., Chafle, G., Kumar, K. A., Mittal, S., and Srivastava, B. (2005). Synthy: A System for End to End Composition of Web Services. *Journal of Web Semantics*, 3:311–339.

Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludäscher, B., and Mock, S. (2004). Kepler: An extensible system for design and execution of scientific workflows. In *16th Intl. Conference on Scientific and Statistical Database Management(SSDBM)*.

Andrews et al., T. (2003). Business Process Execution Language for Web Services Version 1.1. http://www.ibm.com/developerworks/library/ws-bpel/ (as of 2007-02-11).

Austin, D., Grainger, W., Barbir, A., Ferris, C., and Garg, S. (2002). Web services architecture requirements. Technical report, W3C.

Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The Semantic Web. *Scientific American*, 284(5):28–37.

Digiampietri, L., Pérez-Alcázar, J., and Medeiros, C. (2007). An ontology-based framework for bioinformatics workflows. *Int. Journal of Bioinformatics Research and Applications*. Accepted for publication.

Digiampietri, L. A., Medeiros, C. B., and Setubal, J. C. (2005). A framework based in Web services orchestration bioinformatics workflow management. *Genetics and Molecular Research*, 4(3).

Fensel, D. and Bussler, C. (2002). The Web Service Modeling Framework WSMF. *Electronic Commerce Research and Applications*, 1(2).

Ghallab, M., Howe, A., Knoblock, C., McDermott, D., Ram, A., Veloso, M., Weld, D., and Wilkins, D. (1998). PDDL the planning domain definition language. In *Proc. of AIPS-98 Planning Committee*.

Kim, J. and Gil, Y. (2004). Towards Interactive Composition of Semantic Web Services. In *AAAI 2004*.

Kuter, U. and Nau, D. (2004). Forward-Chaining Planning in Nondeterministic Domains. In *AAAI 2004*, pages 513–518.

Kuter, U., Nau, D., Pistore, M., and Traverso, P. (2005). A Hierarchical Task-Network Planner based on Symbolic Model Checking. In *(ICAPS 2005)*, pages 300–310.

Levesque, H. J., Reiter, R., Lesperance, Y., Lin, F., and Scherl, R. B. (1997). Golog: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31(1–3):59–84.

Long, D. and Fox, M. (2003). The 3rd International Planning Competition: Results and Analysis. *Journal of Artificial Intelligence Research*, 20:1–59.

Madhusudan, T. and Uttamsingh, N. (2006). A declarative approach to composing web services in dynamic environments. *Decision Support Systems*, 41(2):325–357.

Martin et al., D. (2004). OWL-S: Semantic Markup for Web Servi-ces. http://www.daml.org/services/owl-s/1.1B/owl-s.pdf (as of 2005-09-08).

McDermott, D. (2002). Estimated-Regression Planning for Interactions with Web Services. In *AIPS 2001*.

McIlraith, S. A. and Son, T. C. (2002). Adapting Golog for Composition of Semantic Web Services. In *KR2002*, pages 482–493.

Medeiros, C. B., Perez-Alcazar, J., Digiampietri, L., Pastorello, G., Santanche, A., Torres, R., Madeira, E., and Bacarin, E. (2005). WOODSS and the Web: Annotating and Reusing Scientific Workflow. *ACM SIGMOD Record*, 34(3):18–23.

Nau, D., Au, T., Ilghami, O., Kuter, U., Murdock, W., Wu, D., and Yaman, F. (2003). SHOP2: An HTN Planning System. *Journal of Artificial Intelligence Research*, 20:379–404.

Rao, J. and Su, X. (2004). A Survey of Automated Web Service Composition Methods. In *SWSWPC 2004*, volume 3387, pages 43–54.

Sirin, E., Parsia, B., Wu, D., Hendler, J. A., and Nau, D. S. (2004). HTN planning for Web Service composition using SHOP2. *Journal of Web Semantics*, 1(4):377–396.

Srivastava, B. and Koehler, J. (2003). Web Service Composition - Current Solutions and Open Problems. In *ICAPS 2003*, pages 28–35.

Stevens, R., Baker, P., Bechhofer, S., Ng, G., Jacoby, A., Paton, N. W., Goble, C. A., and Brass, A. (2000). TAMBIS: Transparent Access to Multiple Bioinformatics Information Sources. *Bioinformatics*, 16(2):184–186.

Traverso, P. and Pistore, M. (2004). Automated Composition of Semantic Web Services into Executable Processes. *Lecture Notes in Computer Science*, 3298:380–394.

W3C (2003). Extensible Markup Language (XML). http://www.w3.org/XML/ (as of 2007-02-06).

WfMOpen. WfMOpen project. http://wfmopen.sourceforge.net/ (as of 2007-02-11).

Zeng, L., Benatallah, B., Ngu, A., Dumas, M., Kalagnanam, J., and Chang, H. (2004). Qosaware middleware for web services composition. *IEEE Trans. Software Eng.*, 30:311–327.