Planning and Monitoring Web Service Composition*

M. Pistore[1], F. Barbon[2], P. Bertoli[2], D. Shaparau[2], and P. Traverso[2]

[1] University of Trento - ITALY
pistore@dit.unitn.it
[2] ITC-irst - Trento - ITALY
[barbon,bertoli,traverso,shaparau]@irst.itc.it

Abstract. The ability to automatically compose web services, and to monitor their execution, is an essential step to substantially decrease time and costs in the development, integration, and maintenance of complex services. In this paper, we exploit techniques based on the "Planning as Model Checking" approach to automatically compose web services and synthesize monitoring components. By relying on such a flexible technology, we are able to deal with the difficulties stemming from the unpredictability of external partner services, the opaqueness of their internal status, and the presence of complex behavioral requirements. We test our approach on a simple, yet realistic example; the results provide a witness to the potentiality of this approach.

1 Introduction

The emerging paradigm of web services provides the basis for the interaction and coordination of business processes that are distributed among different organizations, which can exchange services and can cooperate to provide better services, e.g., to third parties organizations or to individuals. One of the big challenges for the taking up of web services is the provision of computer automated support to the composition of service oriented distributed processes, in order to decrease efforts, time, and costs in their development, integration, and maintenance. The ability to automatically plan the composition of web services, and to monitor their execution is therefore an essential step toward the real usage of web services.

BPEL4WS (Business Process Execution Language for Web Services) [1] is an emerging standard for the specification and execution of service oriented business processes. BPEL4WS has been designed with two functions in mind. On the one hand, *executable* BPEL4WS programs allow the specification and execution of the processes internal to an organization (*internal processes* in the following). On the other hand, *abstract* BPEL4WS specifications can be used to specify and publish the protocol that external agents have to follow to interact with a web service (*external protocols* in the following). Therefore, BPEL4WS offers the natural starting point for web service composition.

In this paper, we devise a planning technique for the automated composition and automated monitoring of web services. Automated composition allows providing services

* The work is partially funded by the FIRB-MIUR project RBNE0195K5, "Knowledge Level Automated Software Engineering".

that combine other, possibly distributed, services, in order to achieve a given business goal. Starting from the description of the external protocols (e.g., expressed as an abstract BPEL4WS specification) and given a "business requirement" for the process (i.e. the goal it should satisfy, expressed in a proper goal language), the planner synthesizes automatically the code that implements the internal process and exploits the services of the partners to achieve the goal. This code can be expressed in a process execution language such as executable BPEL4WS.

Our planning techniques are also exploited to automatically generate a monitor of the process, i.e., a piece of code that is able to detect and signal whether the external partners do not behave consistently with the specified protocols. This is vital for the practical application of web services. Run-time misbehaviors may take place even for automatically composed (and possibly validated) services, e.g. due to failures of the underlying message-passing infrastructure, or due to errors or changes in the specification of external web services.

In order to achieve these results, our planner must address the following difficulties, which are typical of planning under uncertainty:

- **Nondeterminism**: The planner cannot foresee the actual interaction that will take place with external processes, e.g., it cannot predict a priori whether the answer to a request for availability will be positive or negative, whether a user will confirm or not acceptance of a service, etc.
- **Partial Observability**: The planner can only observe the communications with external processes; that is, it has no access to their internal status and variables. For instance, the planner cannot know a priori the list of items available for selling from a service.
- **Extended Goals**: Business requirements often involve complex conditions on the behavior of the process, and not only on its final state. For instance, we might require that the process never gets to the state where it buys an item costing more than the available budget. Moreover, requirements need to express conditional preferences on different goals to achieve. For instance, a process should try first to reserve and confirm both a flight and an hotel from two different service providers, and only if one of the two services is not available, it should fall back and cancel both reservations.

We address these problems by developing planning techniques based on the "Planning as model checking" approach, which has been devised to deal with nondeterministic domains, partial observability, and extended goals. A protocol specification for the available external services is seen as a nondeterministic and partially observable domain, which is represented by means of a finite state machine. Business requirements are expressed in the EaGLe goal language [8], and are used to drive the search in the domain, in order to synthesize a plan corresponding to the internal process defining the web-service composition. Plan generation takes advantage of symbolic model checking techniques, that compactly represent the search space deriving from nondeterministic and partially observable domains. These are also exploited to produce compact monitoring automata that are capable to trace the run-time evolution of external processes, and thus to detect incorrect interactions.

In this paper, we define the framework for the planning of composition and monitoring of distributed web services, describe it through an explanatory example, and implement the planning algorithm in the MBP planner. We provide an experimental evaluation which witnesses the potentialities of our approach.

2 A web-service composition domain

Our reference example consists in providing a furniture purchase & delivery service. We do so by combining two separate, independent existing services: a furniture producer, and a delivery service.

We now describe the protocols that define the interactions with the existing services. These protocols can be seen as very high level descriptions of the BPEL4WS external protocols that would define the services in a real application.

The protocol provided by the furniture producer is depicted in Fig. 1.3. The protocol becomes active upon a request for a given item; the item may be available or not — in the latter case, this is signaled to the request applicant, and the protocol terminates with failure. In case the item is available, the applicant is notified with informations about the product, and the protocol stops waiting for either a positive or negative acknowledgment, upon which it either continues, or stops failing. Should the applicant decide that the offer is acceptable, the service provides him with the cost and production time; once more, the protocol waits for a positive or negative acknowledgment, this time terminating in any case (with success or failure).

The protocol provided by the delivery service is depicted in Fig. 1.2. The protocol starts upon a request for transporting an object of a given size to a given location. This might not be possible, in which case the applicant is notified, and the protocol terminates failing. Otherwise, a cost and delivery time are computed and signaled to the applicant; the protocol suspends for either a positive or negative acknowledgment, terminating (with success or failure respectively) upon its reception.

The idea is that of combining these services so that the user may directly ask the combined service to purchase and deliver a given item at a given place. To do so, we exploit a description of the expected protocol the user will execute when interacting with the service. According to the protocol (see Fig. 1.1), the user sends a request to get a given item at a given location, and expects either a signal that this is not possible (in which case the protocol terminates, failing), or an offer indicating the price and cost of the service. At this time, the user may either accept or refuse the offer, terminating its interaction in both cases.

Thus a typical (nominal) interaction between the user, the combined purchase & delivery service P&S, the producer, and the shipper would go as follows:

1. the user asks P&S for a certain item I, that he wants to be transported at location L;
2. P&S asks the producer some data about the item, namely its size, the cost, and how much time does it take to produce it;
3. P&S asks the delivery service the price and cost of transporting an object of such a size to L;
4. P&S provides the user a proposal which takes into account the overall cost (plus an added cost for P&S) and time to achieve its goal;

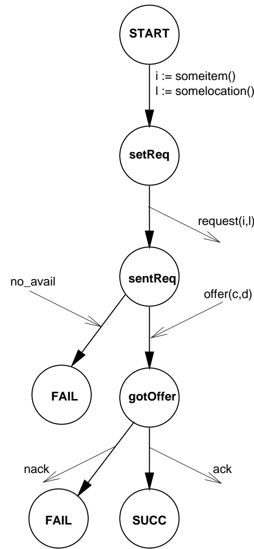


Fig.1.1: The user protocol

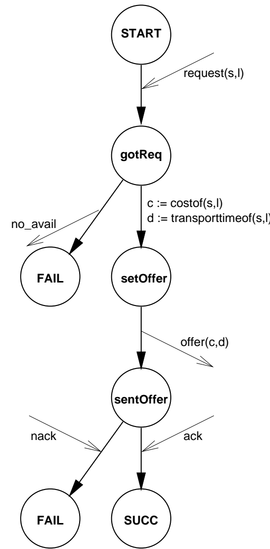
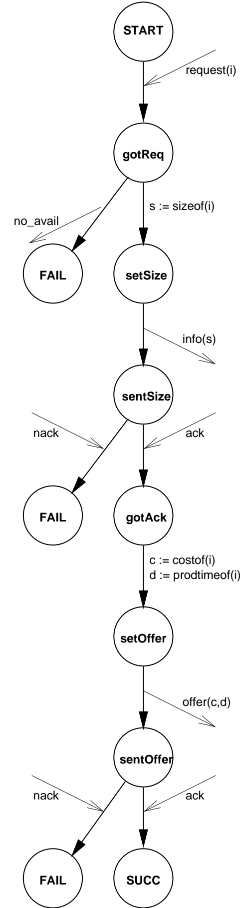


Fig.1.2: The shipper protocol



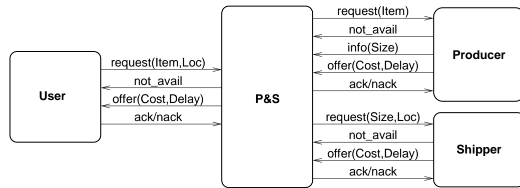


Fig.1.4: The composed web service

Fig.1.3: The producer protocol

Fig. 1. The protocols and their combination.

5. the user confirms its order, which is dispatched by P&S to the delivery and producer.

Of course this is only the nominal case, and other interactions should be considered, e.g., for the cases the producer and/or delivery services are not able to satisfy the request, or the user refuses the final offer.

3 Planning via Model Checking

The 'Planning as Model Checking" framework [5] is a general framework for planning under uncertainty. It allows for *non-determinism* in the initial state and in the outcome

of action execution. It allows modeling planning domains with different *degrees of run-time observability*, where e.g. the domain state is only partially visible via sensing, and for expressing *temporally extended planning goals*, i.e., goals that define conditions on sequences of states resulting from the execution of a plan, rather than just on final states.

In this framework, a domain is a model of a generic system with its own dynamics; formally, it is expressed as a (non-deterministic) finite state automaton. The domain is controlled by a plan which is also represented as a (deterministic) finite state automaton. During execution, the plan produces *actions*, which constitute the input of the domain, on the basis of the *sensing* coming from the domain and of its internal state. In turn, the domain evolves its state in response to the action, and produced a new output. That is, the execution can be seen as the evolution of the synchronous product of the automata (formal details are presented in [5]).

This generality involves a high complexity in the plan synthesis. In particular, partial observability involves the need for dealing with uncertainty about the current domain state: at every plan execution step, the executor must consider a set of equally plausible states, also called a *belief state* or simply *belief* [7, 4]. When searching for a plan, also the planner must consider beliefs and the result of actions and sensing over them. Therefore, the search in a partially observable domain $\mathcal{D}$ can be described as search inside a correspondent fully observable "belief-level", or *knowledge-level*, domain $\mathcal{D}_K$ whose states are the beliefs of $\mathcal{D}$. Knowledge-level domain $\mathcal{D}_K$ can be produced by a power-set construction upon $\mathcal{D}$.

This leads to the important consequence that planning under partial observability can be performed by planning for full observability over the associated knowledge-level domain [3]:

Fact 1 *Let $\mathcal{D}$ be a ground-level domain and g be a knowledge-level goal for $\mathcal{D}$. Let also $\mathcal{D}_K$ be the knowledge level domain for $\mathcal{D}$. Then Π is a plan that achieves g on $\mathcal{D}$ if, and only if, Π achieves (the knowledge-level interpretation of) g on $\mathcal{D}_K$.*

Also notice that the knowledge-level domain describes all the valid observable behaviors of the domain. Therefore, we can use it as a monitor, to check at run time whether the domain really behaves as predicted in the domain model.

In the "Planning as Model Checking" framework, extended goals have been originally expressed as CTL [10] formulas, inheriting a standard semantics over the execution structure. Planning with CTL goals has been studied in [13, 14] under the hypothesis of full observability and in [5, 3] in the more general case of partial observability. However, CTL is often inadequate for planning, since it cannot express crucial aspects such as failure and recovery conditions, intentionality, and preferences amongst sub-goals. EaGLe [8] has been designed with the purpose to overcome these limitations of CTL and to provide a language for temporally extended planning goals in non-deterministic domains. EaGLe constructs permit to express goals of different strength (e.g. **DoReach** p requires achieving p, while **TryReach** p requires the controller to do its best to achieve p, but admits not reaching it); they handle failure, and sub-goal preferences (namely, in g_1 **Fail** g_2, goal g_1 is considered first, and only if its achievement or maintenance fails, then goal g_2 is used as a recovery or second-choice goal). The combination of these concepts is very powerful, and allows expressing goals such as

e.g. **TryReach** c **Fail DoReach** d: here, the sub-goal **TryReach** c requires to find a plan that tries to reach condition c. During the execution of the plan, a state may be reached from which it is not possible to reach c. When such a state is reached, goal **TryReach** c fails and the recovery goal **DoReach** d is considered. A formal semantics and a planning algorithm for EaGLe goals in fully observable domains appears in [8].

4 Planning for web-service composition

In this section we describe how the techniques provided by the planning as model checking framework can be applied to solve the problem of planning and monitoring of web-service composition.

The first step to model the domain, and to put ourselves in a position to synthesize a combined web service, is to model each of the protocols of the external partners as a planning domain. The states of the domain are used to codify the states of the protocol, the current values of the variables, and the contents of the input and output channels.

The modeling of the state of the protocol and of its variables is straightforward. The modeling of channels is more complex; each channel features a channel state (empty or full), and in case it is full, auxiliary variables are used to store the values carried by the channel. For instance, in the case of the shipper, the input channel REQUEST conveys the item size and the location as data, and is hence modeled with variables REQUEST.STATUS, REQUEST.SIZE, and REQUEST.LOC.

The only actions in the model correspond to channel read/write operations, since these are the only operations available to an external agent for controlling the evolution of the protocol. A receive action can only be executed on an output channel of the protocol which is full; its execution empties the channel and updates the protocol state. A send action, on the other hand, is possible only on an empty input channel of the protocol; if the input channel carries values, the action also specifies the actual values transmitted on the channel; when this action is executed, the channel is marked as full, and the transmitted values are stored in the appropriate channel variables.

The transitions in the model capture different aspects of the evolution of the system. From one side, they describe the update of the channel variables due to send/receive actions executed by the agent interacting with the protocol. From the other side, they model the internal evolution of the protocol. This internal evolution may correspond to the receive/send operations complementary to the send/receive actions executed externally. Also, it may correspond to updates in the internal variables, when assignment steps of the protocols are executed. Finally, no-op transitions are introduced to model the possibility of protocols being idle.

The observations of the protocol are limited to the status of the input and output channels and to the values contained in a channel when it is read.

This construction defines the ground domain for each of the external partners. Given one of these ground domains, we can perform the power-set construction to obtain the associated knowledge level domain, which is used as a run-time monitor to check whether the behaviors of the external partners conform to the protocol, and trigger an appropriate handling of failures. (It is easy to extend it with a time-out mechanism to also trap the undesired condition where a portion of some protocol is stuck.)

The goal of the planning task is to synthesize the process that interacts with the three external processes in order to provide the purchase & delivery service to the users. The planning domain consists of the combination of the three ground-level domains for the external partners, i.e. of their synchronous product. The business goal of P&S can be described as follows:

1. The service should try to reach the ideal situation where the user has confirmed his order, and the service has confirmed the associated (sub-)orders to the producer and shipper services. In this situation, the data associated to the orders have to be mutually consistent, e.g. the time for building and delivering a furniture shall be the sum of the time for building it, and that for delivering it.
2. Upon failure of the above goal, e.g., because the user refuses the offer, the service must reach a fall-back situation where every (sub-)order has been canceled. That is, there must be no chance that the service has committed to some (sub-)order before the user cancels his order.

This goal can be expressed by the following EaGLe formula, where the **TryReach** and **DoReach** clauses represents the above portions 1 and 2 of the requirement.

$$\begin{aligned}&\textbf{TryReach}\ (user.succ \wedge producer.succ \wedge shipper.succ \wedge \\ &\qquad\qquad user.d = add_delay(producer.d, shipper.d) \wedge \\ &\qquad\qquad user.c = add_cost(producer.c, shipper.c)) \\ &\textbf{Fail DoReach}(user.fail \wedge producer.fail \wedge shipper.fail)\end{aligned}$$

The planning domain is only partially observable by the executor of the process that we want to synthesize. Thus, solving the problem implies using dedicated algorithms for planning under partial observability with EaGLe goals, or, alternatively, planning for the fully observable associated knowledge level domain. We pursue this latter approach, so that we can reuse existing EaGLe planning algorithms under full observability. We proceed as follows. We generate the knowledge level domain by combining the three monitors defined previously. Similarly to what happens for the ground level domains, this computation consists of a synchronous product. Inside the knowledge level domain, we mark as $success$ the belief states which contain only states satisfying the ideal condition that the services *tries* to reach (i.e. $user.succ \wedge producer.succ \wedge shipper.succ \wedge user.d = add_delay(producer.d, shipper.d) \wedge user.c = add_cost(producer.c, shipper.c) \wedge empty_channels$), and as $failure$ the belief states which contain only states satisfying the condition that the service *has* to reach in case the preferred objective fails (i.e. $user.fail \wedge producer.fail \wedge shipper.fail \wedge empty_channels$). Finally, we plan on this domain with respect to the aforementioned EaGLe goal. Fact 1 guarantees that the approach outlined above for planning under partial observability with EaGLe goals is correct and complete.

5 Experimental results

In order to test the effectiveness and the performance of the approach described above, we have conducted some experiments using the MBP planner. Some extensions to the

planner have been necessary to the purpose. First, we have implemented the procedure for translating protocols similar to the ones described in Fig. 1 into the ground-level planning domains represented in the input language of MBP. Second, we have implemented a routine that performs the power-set domain construction, building the monitors corresponding to the three external protocols. MBP already provides algorithms for planning with EaGLe goals, which we have exploited in the last step of the planning algorithm.

We have run MBP on four variants of the case study considered in this paper, of different degrees of complexity. In the easiest case, CASE 1, we considered a reduced domain with only the user and the shipper, and with only one possible value for each type of objects in the domain (item, location, delay, cost, size). In CASE 2 we have considered all three protocols, but again only one possible value for each type of object. In CASE 3 we have considered the three protocols, with two objects for each type, but removing the parts of the shipper and producer protocols concerning the size of the product. In CASE 4, finally, is the complete protocol. We remark that CASE 1 and CASE 2 are used to test our algorithms, even if they are admittedly unrealistic, since the process knows, already before the interaction starts, the item that the user will ask and the cost will be charged to the user. In CASE 3 and CASE 4, a real composition of services is necessary to satisfy the goal.

In all four cases we have experimented also with a variant of the shipper protocol, which does not allow for action $nack$.

The experiments have been executed on an Intel Pentium 4, 1.8 GHz, 512 MB memory, running Linux 2.4.18. The results, in Fig. 2, report the following information:

- Generate: the time necessary to generate the MBP description of knowledge domains from their description of the three external protocols.
- BuildMonitor: the time necessary to parse and build the three internal MBP models corresponding to the monitors of the three external protocols; after the models have been built, it is possible to monitor in real-time the evolution of the protocols.
- BuildDomain: the time necessary to parse and build the internal MBP model of the combination of the three knowledge level domains.
- Planning: the time required to find a plan (or to check that no plan exists) starting from the planning domain built in the previous step.
- Result: whether a plan is found or not.

The last two results are reported both in the original domains and in the domains without "$nack$" being handled by the shipper.

The experiments show that the planning algorithm correctly detects that it is not possible to satisfy the goal if we remove the $nack$ action handling in the shipper, since we cannot unroll the contract with the shipper and to satisfy the recovery goal **DoReach** $failure$ in case of failure of the main goal. Moreover, MBP has been able to complete all the planning and monitor construction tasks with a very good performance, even for the most complex protocol descriptions. Indeed, in the worse case, MBP takes around two minutes to complete the task; as a reference, we asked an experienced designer to develop the protocol combination, and this took him more than one hour.

	With shipper nack					Without shipper nack	
	Generate	BuildMonitor	BuildDomain	Planning	Result	Planning	Result
CASE 1	1 sec.	1 sec.	2 sec.	1 sec.	YES	0 sec.	NO
CASE 2	1 sec.	1 sec.	6 sec.	8 sec.	YES	6 sec.	NO
CASE 3	8 sec.	15 sec.	32 sec.	23 sec.	YES	14 sec.	NO
CASE 4	12 sec.	65 sec.	63 sec.	118 sec.	YES	23 sec.	NO

Fig. 2. Results of the experiments.

6 Related work and conclusions

In this paper, we define, implement and experiment with a framework for planning the composition and monitoring of BPEL4WS web services. As far as we know, there is no previous attempt to automatically plan for the composition and monitoring of service oriented processes that takes into account nondeterminism, partial observability, and extended goals.

The problem of simulation, verification, and automated composition of web services has been tackled in the semantic web framework, mainly based on the DAML-S ontology for describing the capabilities of web services [2]. In [12], the authors propose an approach to the simulation, verification, and automated composition of web services based a translation of DAML-S to situation calculus and Petri Nets, so that it is possible to reason about, analyze, prove properties of, and automatically compose web services. However, as far as we understand, in this framework, the automated composition is limited to sequential composition of atomic services for reachability goals, and do not consider the general case of possible interleavings among processes and of extended business goals. Moreover, Petri Nets are a rather expressive framework, but algorithms that analyze them have less chances to scale up to complex problems compared to symbolic model checking techniques.

Different planning approaches have been proposed for composing web services, from HTNs [16] to regression planning based on extensions of PDDL [9], but how to deal with nondeterminism, partial observability, and how to generate conditional and iterative behaviors (in the style of BPEL4WS) in these frameworks are still open issues.

Other planning techniques have been applied to related but somehow orthogonal problems in the field of web services. The interactive composition of information gathering services has been tackled in [15] by using CSP techniques. In [11] an interleaved approach of planning and execution is used; planning technique are exploited to provide viable plans for the execution of the composition, given a specific query of the user; if these plans turn out to violate some user constraints at run time, then a re planning task is started. Finally, works in the field of Data and Computational Grids is more and more moving toward the problem of composing complex workflows by means of planning and scheduling techniques [6].

While the results in this paper are extremely promising, there is a wide space for improving. In particular, the computationally complex power-set construction of the knowledge level domain can be avoided altogether by providing algorithms for natively planning with extended goals under partial observability. This is a is a main goal in our research line; a preliminary result appears in [3], focusing on the CTL goal language.

The extension of this work to the EaGLe language, and the exploitation of symbolic representation techniques for this problem, is far from trivial.

Finally, we intend to extend our experimentation on a set of realistic case studies expressed using the BPEL4WS language; this will require an extension of MBP to natively handle the language.

References

[1] Andrews, T.; Curbera, F.; Dolakia, H.; Goland, J.; Klein, J.; Leymann, F.; Liu, K.; Roller, D.; Smith, D.; Thatte, S.; Trickovic, I.; and Weeravarana, S. 2003. Business Process Execution Language for Web Services, Version 1.1.

[2] Ankolekar, A. 2002. DAML-S: Web Service Description for the Semantic Web. In *Proc. of the 1st International Semantic Web Conference (ISWC 02)*.

[3] Bertoli, P., and Pistore, M. 2004. Planning with Extended Goals and Partial Observability. In *Proc. of ICAPS'04*.

[4] Bertoli, P.; Cimatti, A.; Roveri, M.; and Traverso, P. 2001. Planning in Nondeterministic Domains under Partial Observability via Symbolic Model Checking. In Nebel, B., ed., *Proc. of IJCAI 2001*.

[5] Bertoli, P.; Cimatti, A.; Pistore, M.; and Traverso, P. 2003. A Framework for Planning with Extended Goals under Partial Observability. In *Proc. ICAPS'03*.

[6] Blythe, J.; Deelman, E.; and Gil, Y. 2003. Planning for Workflow Construction and Maintenance on the Grid. In *Proc. of ICAPS'03 Workshop on Planning for Web Services*.

[7] Bonet, B., and Geffner, H. 2000. Planning with Incomplete Information as Heuristic Search in Belief Space. In *Proc. AIPS 2000*.

[8] Dal Lago, U.; Pistore, M.; and Traverso, P. 2002. Planning with a Language for Extended Goals. In *Proc. AAAI'02*.

[9] McDermott, D. 1998. The Planning Domain Definition Language Manual. Technical Report 1165, Yale Computer Science University. CVC Report 98-003.

[10] Emerson, E. A. 1990. Temporal and modal logic. In van Leeuwen, J., ed., *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*. Elsevier.

[11] Lazovik A.; Aiello M.; and Papazoglou M. 2003. Planning and Monitoring the Execution of Web Service Requests. In *Proc. of ICSOC'03*.

[12] Narayanan, S., and McIlraith, S. 2002. Simulation, Verification and Automated Composition of Web Services. In *Proc. of WWW-11*.

[13] Pistore, M., and Traverso, P. 2001. Planning as Model Checking for Extended Goals in Non-deterministic Domains. In *Proc. IJCAI'01*.

[14] Pistore, M.; Bettin, R.; and Traverso, P. 2001. Symbolic Techniques for Planning with Extended Goals in Non-Deterministic Domains. In *Proc. ECP'01*.

[15] Thakkar, S.; Knoblock, C.; and Ambite, J. L. 2003. A View Integration Approach to Dynamic Composition of Web Services. In *Proc. of ICAPS'03 Workshop on Planning for Web Services*.

[16] Wu, D.; Parsia, B.; Sirin, E.; Hendler, J.; and Nau, D. 2003. Automating DAML-S Web Services Composition using SHOP2. In *Proc. of ISWC2003*.