Automated Web Service Composition by On-The-Fly Belief Space Search *

P. Bertoli[1] and **M. Pistore**[2] and **P. Traverso**[1]
[2] [traverso,bertoli]@irst.itc.it - ITC-IRST - via Sommarive 18, 38050, Trento, Italy
[1] pistore@dit.unitn.it - DIT - University of Trento - via Sommarive 14, 38050, Trento, Italy

Abstract

Composition of stateful web services expressed in BPEL4WS can be recasted as a problem of planning in asynchronous domains. In (PTB05), this is pursued by encoding the asynchronous and partially observable behaviors of services within a domain whose states represent *beliefs* on the state of each service. In this work, we propose a novel approach, where such belief-level domain is not built explicitly, but rather visited on-the-fly. We evaluate the relative merits of the approaches, showing the advantages of the on-the-fly approach for a significant class of composition problems.

Introduction

Planning is a very promising technique for the automated composition of web services. Recent works (NM02; HBCS03; BCG+03; PTB05; PMBT05) use planning techniques to address the problem of composing stateful services which implement distributed business processes. Some of these works (PTB05; PMBT05) start from abstract service descriptions given in the standard modeling and execution language BPEL4WS (ACD+03), and produce an executable BPEL4WS service as a result, which can be run by existing BPEL4WS execution engines. This form of automated composition is particularly difficult, since BPEL4WS services must be modeled with nondeterministic and partially observable behaviors. Moreover, the interactions of BPEL4WS services are intrinsically *asynchronous*, i.e., each service evolves independently and synchronizes with the other processes only via asynchronous message exchanges.

One of the most effective solutions currently available, proposed in (PTB05), is based on a pre-analysis that constructs the planning domain as a "*belief-level*" transition system, i.e. an automaton whose states represent sets of observationally indistinguishable states of the component services. This explicit construction of the belief-level domain may constitute a hard bottleneck, since belief-level domains can be larger than the domains they originate from.

In this paper, we propose a novel solution which consists in visiting *on-the-fly* the belief space induced by the set of web services being composed, using a heuristic and-or search algorithm. The intuition is that very often, a heuristic search may identify a solution by visiting only a very small portion of the belief space, which is likely to reflect in an improved composition performance. We propose a semi-symbolic and-or search approach that also deals with the existence of internal asynchronous service evolutions. This results in the first instance ever of an and-or planning algorithm for nondeterministic, partially observable, and asynchronous domains.

We compare this novel approach w.r.t. (PTB05), considering differently structured composition scenarios. We show that the novel solution scales up of orders of magnitude for the cases where services to be composed have a certain amount of complexity.

The paper is structured as follows. First, we define the composition problem and provide a general schema for its solution. We then rephrase the problem in terms of belief-level search, and discuss our approach. Finally, we sketch our implementation, and report our experimental comparison against the off-line approach of (PTB05). We wrap up discussing related work and future work directions.

Overview

Our goal is to automatically generate a new service W (called the *composite service*) that interacts with a set of published web services $W_1, \ldots, W_n$ (called the component services) and satisfies a given composition requirement. More specifically (see Fig. 1) we assume that component services are described as BPEL4WS abstract processes. Given n BPEL4WS abstract processes $W_1, \ldots, W_n$, the BPEL2STS module automatically translates each of them into a *state transition system* (STS from now on) $\Sigma_{W_1}, \ldots, \Sigma_{W_n}$. Intuitively, each Σ_{W_i} represents all the possible behaviors of service W_i, in terms of state evolutions on the basis of input, output and internal actions. From these STSs, the module STS2DOM builds a *planning domain* $\mathcal{D}$ which describes all the the possible evolutions of the set of services, and which is obtained by combining $\Sigma_{W_1}, \ldots, \Sigma_{W_n}$ via a *parallel product* operation. The domain $\mathcal{D} = \Sigma_{W_1} \parallel \ldots \parallel \Sigma_{W_n}$ is passed as an input to the planner. The second kind of input to the planner consists of the requirements ρ for the composite service. In particular, in this paper, we will consider requirements expressed as *reachability goals*.

Given $\mathcal{D}$ and ρ, the planner generates a plan π that is then translated into a STS Σ_c. Σ_c encodes the new composite service W, which dynamically receives and sends invocations from/to the composite services $W_1, \ldots, W_n$ and behaves depending on responses received from the external services.

*This work is partially funded by the MIUR-FIRB project RBNE0195K5, "Knowledge Level Automated Software Engineering", and by the MIUR-PRIN 2004 project "Advanced Artificial Intelligence Systems for Web Services".

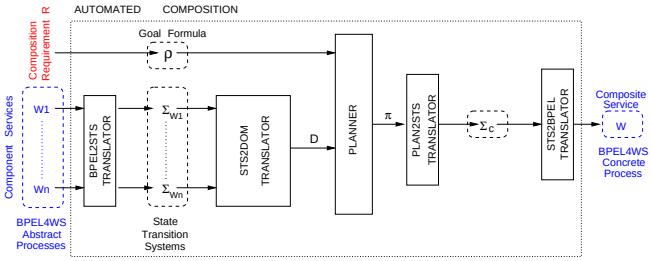

Figure 1: The approach.

Σ_c is such that $\Sigma_c \triangleright \Sigma_{\|}$ satisfies the requirement ρ, where $\Sigma_c \triangleright \Sigma_{\|}$ is the STS representing the evolutions of the component services as they are controlled by the composite service. In particular, this implies that Σ_c is deadlock-free, i.e. it must be such that no chance of a deadlock due to critical runs may ever occur in $\Sigma_c \triangleright \Sigma_{\|}$.

The STS Σ_c is then translated into a concrete BPEL4WS process that implements the desired composite web service.

Thus the composition problem we solve can be formally stated as follows (see also (PTB05)):

Definition 1 (composition problem)
Let $\Sigma_1, \ldots, \Sigma_n$ be a set of state transition systems, and let ρ be a reachability goal. The composition problem for $\Sigma_1, \ldots, \Sigma_n$ and ρ is the problem of finding a deadlock-free controller Σ_c such that $\Sigma_c \triangleright (\Sigma_1 \parallel \ldots \parallel \Sigma_n) \models \rho$.

Belief-level search

In this section, we rephrase the composition problem so to explicitly represent the search space used to look for Σ_c. In order to do so, we have to consider that $\Sigma_{\|} = \Sigma_1 \parallel \ldots \parallel \Sigma_n$ is a (nondeterministic) STS that is only partially observable by Σ_c. That is, at execution time, the composite service Σ_c cannot in general get to know exactly what is the current state of the component services modeled by $\Sigma_{W_1}, \ldots, \Sigma_{W_n}$.

This means that, during our search, we need to group together the observationally indistinguishable executions of the controlled system we are building, by considering, at each step of the execution, the set of states plausible given the current incomplete knowledge. Such a set of states is called a *belief*. The initial belief for the execution is the set of initial states $\mathcal{S}^0$ of $\Sigma_{\|}$. This belief is updated whenever Σ performs an observable (input or output) transition. More precisely, if $B \subseteq \mathcal{S}$ is the current belief and an action $a \in \mathcal{I} \cup \mathcal{O}$ is observed, then the new belief $B' = \mathit{Evolve}(B, a)$ is defined as follows: $s \in \mathit{Evolve}(B, a)$ if, and only if, there is some state s' reachable from B by performing a (possibly empty) sequence of τ transitions, such that $\langle s', a, s \rangle \in \mathcal{R}$.

Notice that, since a belief contains sequences of states connected by τ-transitions, the notion of property satisfaction must take this into account. Namely, a belief B satisfies a property ρ, written $B \models_\Sigma \rho$, iff every sequence of τ transitions in its τ-closure reaches a final state for which the property holds; we will call the set of such final states the *τ-frontier* of the belief.

We can now define a "belief-level system", that is, a synchronous and deterministic STS for $\Sigma_{\|}$ whose states are the beliefs that may be traversed by the executions of $\Sigma_{\|}$, and whose transitions describe belief evolutions.

Definition 2 (belief-level system)
Let $\Sigma = \langle \mathcal{S}, \mathcal{S}^0, \mathcal{I}, \mathcal{O}, \mathcal{R}, \mathcal{L} \rangle$ be a STS. The corresponding belief-level STS, denoted $\textsc{Bel}(\Sigma)$, is $\Sigma_{\mathcal{B}} = \langle \mathcal{S}_{\mathcal{B}}, \mathcal{S}^0_{\mathcal{B}}, \mathcal{I}, \mathcal{O}, \mathcal{R}_{\mathcal{B}}, \mathcal{L}_{\mathcal{B}} \rangle$, where:

- *$\mathcal{S}^0_{\mathcal{B}} = \{\mathcal{S}^0\}$ is the initial belief;*
- *$\mathcal{S}_{\mathcal{B}}$ are the beliefs of Σ reachable from the initial belief;*
- *$\mathcal{R}_{\mathcal{B}}$ is defined as follows: if $\mathit{Evolve}(B, a) = B' \neq \emptyset$ for some $a \in \mathcal{I} \cup \mathcal{O}$, then $\langle B, a, B' \rangle \in \mathcal{R}_{\mathcal{B}}$;*
- *$\mathcal{L}_{\mathcal{B}}(B) = \{p \in \mathcal{P}rop : B \models_\Sigma p\}$.*

We now recast the notion of composition problem by imposing conditions not on the controller, but on the executions it induces on the controlled domain. This will allow us to look at the problem as to one of searching a satisfactory, properly structured subset of the belief-level system. In particular, since we consider reachability goals, we can restrict our attention to controllers whose execution is finite and loop-free; they induce behaviors that correspond to DAG-structured subsets of the belief level system. Moreover, to guarantee deadlock-freedom, such controllers may never discard outputs of the controlled system, nor provide outputs when the controlled system is providing an input. This results in the following definition:

Definition 3 (Controlled execution subtree)
Let $\Sigma_{\mathcal{B}} = \langle \mathcal{S}_{\mathcal{B}}, \mathcal{S}^0_{\mathcal{B}}, \mathcal{I}, \mathcal{O}, \mathcal{R}_{\mathcal{B}}, \mathcal{L}_{\mathcal{B}} \rangle$ be the belief-level system for a domain $\Sigma = \langle \mathcal{S}, \mathcal{S}^0, \mathcal{I}, \mathcal{O}, \mathcal{R}, \mathcal{L} \rangle$. We say that $\Sigma'_{\mathcal{B}} = \langle \mathcal{S}'_{\mathcal{B}}, \mathcal{S}^0_{\mathcal{B}}, \mathcal{I}, \mathcal{O}, \mathcal{R}'_{\mathcal{B}}, \mathcal{L}'_{\mathcal{B}} \rangle$ is a controlled execution subtree *of $\Sigma_{\mathcal{B}}$, denoted $\Sigma'_{\mathcal{B}} \subseteq \Sigma_{\mathcal{B}}$, if and only if:*

1. $\mathcal{S}'_{\mathcal{B}} \subseteq \mathcal{S}_{\mathcal{B}}$
2. $\forall \langle n, O, n' \rangle \in \mathcal{R}_{\mathcal{B}} : O \in \mathcal{O} \wedge n \in \mathcal{S}'_{\mathcal{B}} \rightarrow \langle n, O, n' \rangle \in \mathcal{R}'_{\mathcal{B}}$
3. $\forall n \in \mathcal{S}_{\mathcal{B}} : \exists O \in \mathcal{O}, I \in \mathcal{I} : \langle n, O, n' \rangle \in \mathcal{R}_{\mathcal{B}} \wedge \langle n, I, n'' \rangle \in \mathcal{R}_{\mathcal{B}} \rightarrow \langle n, I, n'' \rangle \notin \mathcal{R}'_{\mathcal{B}}$

It is now easy to formulate a condition that identifies whether a subtree represents a satisfactory behavior of the controlled system:

Definition 4 (solution subtree) *Let $\Sigma'_{\mathcal{B}} \subseteq \Sigma_{\mathcal{B}}$ be a deadlock-free controlled subtree of $\Sigma_{\mathcal{B}}$. $\Sigma'_{\mathcal{B}}$ is a satisfactory subtree for the problem of composing $\Sigma_1, \ldots, \Sigma_n$ for the requirement ρ iff, for every leaf B of $\Sigma'_{\mathcal{B}}$, $B \models_\Sigma \rho$.*

A solution subtree can be associated to a satisfying controller, and vice versa. Thus we can link the existence of a satisfactory controller for the composition problem to the one of a satisfactory execution subtree:

Theorem 5 (composition problem at the belief-level)
Let $\Sigma_1, \ldots, \Sigma_n$ be a set of state transition systems, let ρ be a reachability goal, let Σ_c be a controller, and let $\pi = \Sigma_c \triangleright \textsc{Bel}(\Sigma_1 \parallel \ldots \parallel \Sigma_n)$. Σ_c is a solution to the problem of composing $\Sigma_1, \ldots, \Sigma_n$ for requirement ρ iff every leaf B of π is such that $B \models_\Sigma \rho$.

The belief-level representation of the problem hints at two possible ways to pursue the solution to our problem:

1. To build "off-line" $\textsc{Bel}(\Sigma_1 \parallel \ldots \parallel \Sigma_n)$, and to visit it by a search algorithm, in order to find an execution subtree satisfying ρ, from which Σ_c can then be extracted;
2. To visit $\textsc{Bel}(\Sigma_1 \parallel \ldots \parallel \Sigma_n)$ while generating it, "on-the-fly", and to identify (the tree associated with) Σ_c.

In the second approach, which we introduce in this paper, no belief-level construction is needed, thus completely eliminating the bottleneck of the "off-line" approach. This approach requires operating directly on the original representation of the domain, and as such the beliefs are actually represented as sets of states - rather than as states in a different "belief-level" domain, like in the off-line approach. Thus, we will need to evolve beliefs by first progressing each of their states, and then computing, for each outcome, the transitive closure of the asynchronous τ-transitions (called τ-closure, see (PTB05)).

Our belief-level search problem is rather similar to the problem of finding a conditional, tree-structure plan for a partially observable domain - a problem which can be solved by and-or search over a synchronous STS. Indeed, to solve our problem, we start from the and-or search approach presented in (BCR01), to which we add the treatment of asynchronism - a notable conceptual departure from the synchronous model adopted by existing planning approaches.

```
1  procedure TauClose(B)
2    B1 := B
3    B0 := ∅
4    while (B1 ≠ B0) do
5      B0 := B1
6      B1 := B0 ∪ Exec(B0, τ)
7    done
8    B1 = B1 ∩ ({s ∈ S : ∀s' ∈ S : ⟨s, τ, s'⟩ ∉ R} ∪
9                {s ∈ S : ∃a ∈ I ∪ O, s' ∈ S : ⟨s, a, s'⟩ ∈ R})
10   return B1;
11 end
```

Figure 2: The routine for τ-closure and pruning.

The algorithm basically performs forward chaining on the and-or tree constituting the belief-level system, while enforcing loop avoidance to guarantee termination of the search. A semi-symbolic representation is exploited to compactly represent sets of states (i.e. beliefs) as single Binary Decision Diagrams, and to effectively manipulate them.

In our extension, the existence of τ-transitions must be taken into account at each forward expansion of a belief. This is performed by the routine reported in Fig. 2, which computes the τ-closure, and prunes away "transient" states, maintaining only those in the τ-frontier, according to the definition of belief evolution. The two tasks are performed by a symbolic fix-point computation, followed by a filtering operation.

The performance of the and-or search algorithm is strongly influenced by the way heuristics select the order of expansion of the nodes. Among the several possible search heuristics, we devised one implementing a high-level "greedy" approach towards maximizing the number of models satisfactory for the goal.

Experiments

In this section, we compare our approach with the off-line approach of (PTB05), on two structurally different categories of composition scenarios.

We first consider the composition of large sets of simple services; this can be seen as a representation of situations likely to be found in, e.g., grid services, where each service performs an atomic operation, and we need to compose several of them to achieve a complex goal. In particular, we consider the scenario described in (PTB05), where a set of services is available, each of them accepting a single request, and either signaling failure, or returning a function of the input. A "not-acknowledge" message drives a component to a final failure state. The composition goal consists in computing a nested function, if possible, and in any case in leaving no service suspended.

The results for the tests are shown in Fig.3 (left). We first report the domain construction times only, and then report the overall performance. It is evident that the price of computing offline the belief-level system grows only linearly with the number of components, and paves the way to a fast symbolic search. On the contrary, the search time in our "on-line" approach grows fast, together with the number of branches of the solution subtree. Considering the overall performance, the on-line approach is convenient up to 6 web services, after which the off-line approach is winning.

We then consider the composition of a small set of rather large web services, each one implementing a complex protocol. This situation is typical of e.g. e-Government scenarios, where the number of partners is often very small, but each partner realizes a vast set of complex procedures. As an example of this kind of scenario, we start from the "producer and shipper" example of (PTB05), where each of the three involved entities describe a rather complex protocol, also reflecting in the complexity of the controller resulting from the composition. We then gradually introduce some variations to the protocols, so to consider a variety of different configurations, featuring different complexities in terms of states of the component web services (and thus of the associated belief-level system). Fig. 3 (center) shows the results.

While both approaches degrade on more complex configurations, they do so in a radically different way. In particular, the impact of constructing the ground-level planning domain is negligible in our approach, while the size of the belief-level domain becomes huge and its explicit construction becomes soon unbearable for the off-line approach. The actual plan synthesis time of (PTB05) is also negatively influenced by the size of the belief domain, which - in spite of using symbolic techniques - makes it very hard to evolve large state frontiers; on the opposite, the performance of the on-the-fly approach basically depends on the size of the produced controller.

We then observe that the (PTB05) scenario, and the variants we considered so far, require that every state of each component service may have to be traversed while executing the composed service - i.e. every portion of every component is relevant to the requirement goal. In general, however, a requirement goal may involve only a partial usage of (some of) the component services. E.g. in an e-Government scenario, a service may realize several different functionalities, of which only one is relevant to the goal. We represent this situation by considering a simpler goal requirement, where we intend just to establish the possibility to conduct an acquirement transaction, without then carrying it out.

The results for this test are given in Fig. 3 (right). Just as before, the off-line approach pays an unaffordable price to

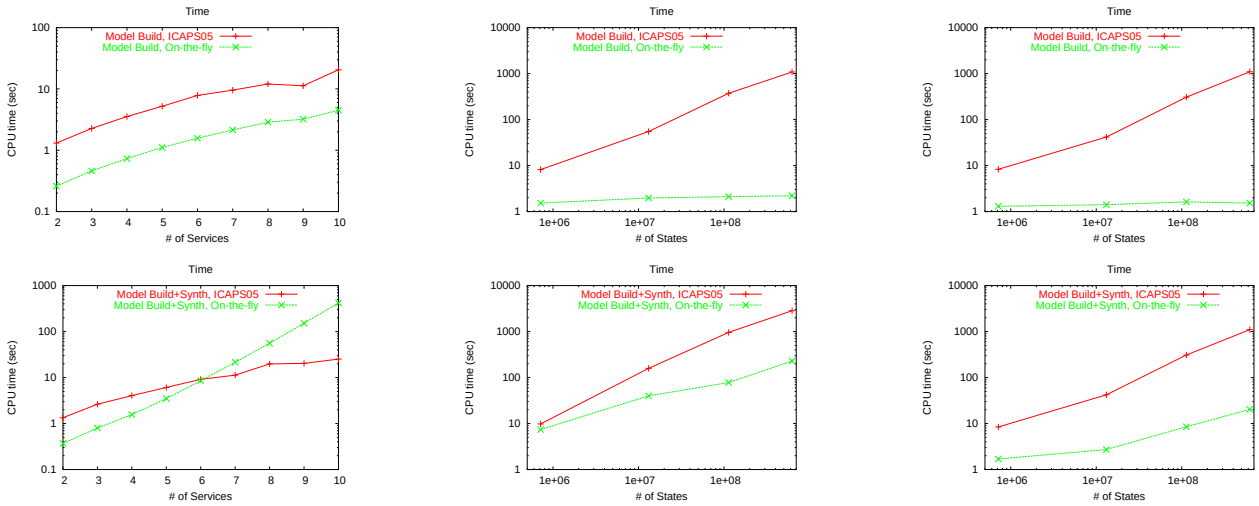

Figure 3: Tests: small scalable, complex, complex with simple solution.

build a large belief-level domain, only to discover later that the produced plan is actually rather simple. The on-the-fly approach, instead, scales up very nicely.

The data from our tests confirm our intuition: in the "off-line" approach of (PTB05) the bottleneck is associated to the size of the component services, while the complexity of the goal only contributes in a limited way. Instead, our "on-the-fly" approach only pays in proportion to the complexity of the produced controller. When component services are very complex, and/or the solution for the problem is not extremely complex, the "on-the-fly" technique exhibits a better performance. The "off-line" technique, thanks to its modular approach to belief-level construction, naturally lends to scenarios involving a large number of simple services.

Conclusions and related work

In this paper, we propose a novel approach to the problem of composing a set of stateful asynchronous web-services, expressed in the standard BPEL4WS language. The approach leverages on a belief-level search technique inspired by a planning approach for partially observable, nondeterministic domains, and extended to deal with asynchronism.

While several automated planning techniques have been proposed to tackle the problem of service composition, see, e.g., (WPS$^+$03; Der98; SdF03), most of them cannot deal with nondeterminism and partial observability. Even the few frameworks capable to deal with partially observable, nondeterministic domains, such as (HBCS03), where service behaviors are described as automata, are limited to synchronous systems, and cannot tackle the problem we consider. Moreover, the considered e-composition problem is fundamentally different from ours, since it is seen as the problem of coordinating the executions of a given set of available services. No concrete and executable processes can be generated with that approach. This is the main difference also with the work described in (BCG$^+$03).

A notable exception is the work in (PTB05), which relies on explicitly building a belief-level STS to exploit existing effective planning algorithms for fully observable, synchronous domains. The computational weight of producing such belief-level system is often unacceptable, as demonstrated by our tests.

In the future, we will investigate on extending this approach to deal with more expressive forms of requirements, e.g. to allow for preferences in the style of (DLPT02).

References

[ACD$^+$03] T. Andrews, F. Curbera, H. Dolakia, J. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weeravarana. Business Process Execution Language for Web Services (version 1.1), 2003.

[BCG$^+$03] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. Automatic composition of E-Services that export their behaviour. In *Proc. ICSOC'03*, 2003.

[BCR01] P. Bertoli, A. Cimatti, and M. Roveri. Conditional Planning under Partial Observability as Heuristic-Symbolic Search in Belief Space. In *Proc. of ECP'01*, 2001.

[Der98] D. Mc Dermott. The Planning Domain Definition Language Manual. Technical Report 1165, Yale Computer Science University, 1998. CVC Report 98-003.

[DLPT02] U. Dal Lago, M. Pistore, and P. Traverso. Planning with a Language for Extended Goals. In *Proc. AAAI'02*, 2002.

[HBCS03] R. Hull, M. Benedikt, V. Christophides, and J. Su. E-Services: A Look Behind the Curtain. In *Proc. PODS'03*, 2003.

[NM02] S. Narayanan and S. McIlraith. Simulation, Verification and Automated Composition of Web Services. In *Proc. WWW2002*, 2002.

[PMBT05] M. Pistore, A. Marconi, P. Bertoli, and P. Traverso. Automated Composition of Web Services by Planning at the Knowledge Level. In *Proc. of IJCAI'05*, 2005.

[PTB05] M. Pistore, P. Traverso, and P. Bertoli. Automated Composition of Web Services by Planning in Asynchronous Domains. In *Proc. ICAPS'05*, 2005.

[SdF03] M. Sheshagiri, M. desJardins, and T. Finin. A Planner for Composing Services Described in DAML-S. In *Proc. AAMAS'03*, 2003.

[WPS$^+$03] D. Wu, B. Parsia, E. Sirin, J. Hendler, and D. Nau. Automating DAML-S Web Services Composition using SHOP2. In *Proc. ISWC'03*, 2003.