# Automated Synthesis of Executable Web Service Compositions from BPEL4WS Processes

M. Pistore
University of Trento - Italy
pistore@dit.unitn.it

P. Traverso, P. Bertoli, A. Marconi
ITC-IRST - Trento - Italy
[traverso,bertoli,marconi]@itc.it

## ABSTRACT

We propose a technique for the automated synthesis of new composite web services. Given a set of abstract BPEL4WS descriptions of component services, and a composition requirement, we automatically generate a concrete BPEL4WS process that, when executed, interacts with the components and satisfies the requirement. We implement the proposed approach exploiting efficient representation techniques, and we show its scalability over case studies taken from a real world application and over a parameterized domain.

**Categories and Subject Descriptors:** H.3.5 Online Information Services: Web-Based Services

**General Terms:** Algorithms, Experimentation

**Keywords:** Web Service Composition, Business Processes, Automated Synthesis

## THE APPROACH

BPEL4WS (Business Process Execution Language for Web Services) [1] is one of the emerging standards for describing the stateful behavior of web services. More precisely, it allows for publishing the interaction protocols of the web services using BPEL4WS *abstract* process specifications, while BPEL4WS *executable* processes are used to implement the internal flow of activities defining the services and to execute them on standard business process execution engines.

The manual analysis of BPEL4WS processes, and the implementation of programs that interact with them, is usually a very difficult, time consuming, and error prone task. Automated web service composition has the potential to reduce development time and effort for new applications, by automatically synthesizing executable composite web services from the published description of existing (component) services. We propose a technique for the automatic synthesis of web service composition, which is based on the following steps.

**Step 1.** We assume that component services are described as BPEL4WS abstract processes, and we automatically translate then into *state transition systems*. They describe dynamic systems that can be in one of their possible states and can evolve to new states as a result of performing some actions (namely, input and output actions, which represent message exchanging, and special "invisible" actions used to model internal evolutions of the services). A transition relation describes how the state can evolve on the basis of input, output, and internal actions.


*WWW 2005,* May 10–14, 2005, Chiba, Japan.
ACM 1-59593-051-5/05/0005.

**Step 2.** We model the requirements for the composite service using EAGLE [4], an expressive formal requirement language allowing expressing preferences and for specifying achievement and maintenance conditions of different strengths.

**Step 3.** Given the state transition systems of the components, and the formal requirements, we automatically generate a state transition system that encodes a process behavior which satisfies the requirements expressed in EAGLE. We first build the parallel combination $\Sigma_{\|}$ of the systems $\Sigma_{W_1}, \ldots, \Sigma_{W_n}$ associated to the component services, which represents all the possible behaviors of these services. We then synthesize the composite system as a state transition system $\Sigma_c$ such that the interactions of $\Sigma_c$ with $\Sigma_{\|}$ satisfy the composition requirement. In the synthesis we need to take into account that $\Sigma_{\|}$ is only partially observable by $\Sigma_c$: due to the "invisible" internal transitions and the nondeterministic behaviors, at execution time the composite service $\Sigma_c$ cannot in general get to know exactly what is the current state of $\Sigma_{W_1}, \ldots, \Sigma_{W_n}$. In [5] we show how to adapt to this task a plan synthesis technique [2, 3, 4], known as *planning via model checking*, which is able to deal with nondeterminism and partial observability, as well as with EAGLE requirements.

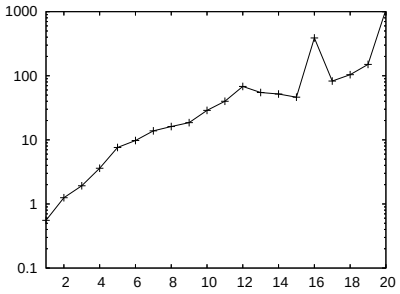
**Step 4.** We automatically translate the resulting state transition system $\Sigma_c$ into an executable concrete BPEL4WS program. The translation is conceptually simple, but particular care has been put in the implementation of this module in order to guarantee that the generated BPEL4WS is of good quality, e.g. it is emitted as a structured program rather than using on jumps and labels.

It is widely recognized that, in general, automated synthesis is a hard problem, both in theory and in practice. In the next section, we show that automated synthesis is also possible in practice. Indeed, the *planning via model checking* techniques we exploit in the synthesis have been shown to be able to scale up to rather large domains for the problem of planning under uncertainty. Moreover, in the considered cases, it is possible to generate BPEL4WS programs that are easy to read and analyze by a programmer, and the size of the program is reasonable compared with the one that can be programmed by hand.

## EXPERIMENTAL EVALUATION

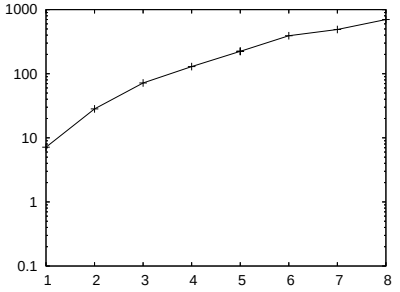
In order to test the performance of the proposed technique, we have conducted some experiments. In the first set of experiments, we test automated synthesis w.r.t. the number of services to be composed. Each component is represented by a very simple abstract BPEL4WS process that is requested to provide a service and can respond either positively or negatively. The composition requirement is that either all services end successfully or a failure is reported to the invoker of the composed service. The results are shown in the following graph, where we report the number of components on the horizontal

axis, and the total time for the automated synthesis (inclusive of conversion from/to BPEL4WS) on the vertical axis:




With these examples, the time required for the automated synthesis increases less than exponentially and manages to deal with a rather high number of components in a rather short time: in the case of 20 components, the automated composition takes about 1000 seconds.

The component web services used in the previous experiment are very elementary, and implement essentially an invoke-response protocol very common in the domain of web services. To model more complex situations, we have complicated the parameterized domain by imposing a composition that requires a high degree of interleaving between components. Here, the interactions with each component are more complex than a single invoke-response step, and, to achieve the goal, it is necessary to carry out interactions with all components in an interleaved way. The results are shown in the following graph:




Automated synthesis in this case is more difficult than in the previous set of experiments. In spite of the fact that the required interleaving reduces performances, the technique still manages to deal with a rather large number of components: we expect that realistic BPEL4WS compositions will include no more than a few components.

To validate the results of this experimental evaluation, we have also conducted some experiments of automated composition on problems extracted from realistic web service domains. The results are reported in the following table:

| | nr. of components | composition time |
|---|---|---|
| P&S | 3 | 9.4 sec. |
| P&S + BANK | 4 | 75.0 sec. |
| WMO1 | 5 | 210.1 sec. |
| WMO2 | 5 | 221.7 sec. |
| WMO3 | 5 | 295.5 sec. |

The first case is the P&S example explained in [5], where a shipper and a producer must be integrated to provide a user with furniture. Automated composition is very fast, since in spite of the interleaving required, we have just three components (shipper, producer, and user). We also experiment with a more complex version of P&S, where a bank is delegated to handle the user's payment, and the interleaving of interactions is increased by the necessity of receiving a payment confirmation from the bank before the order can be confirmed to the producer and the shipper. For this more difficult problem, automated composition time increases of one order of magnitude. Finally, we experiment with a case study taken from a real e-government application we are developing for a private company. We aim at providing a service that implements a (public) waste management office (WMO), i.e., a service that manages user requests to open sites for the disposal of dangerous waste. According to the existing Italian laws, such a request involves the interaction of different offices of the public administration. The composition requirement here can be described with a set of constraints on the order of execution of different procedural steps performed by different offices. We consider three variants of this domain, corresponding to an increasing interleaving between the different services, getting in all cases a very good performance.

In all the realistic examples, automated synthesis has shown to be feasible and take a rather low amount of time, surely much faster than manual development of BPEL4WS composite processes. Moreover, the times required for the synthesis confirm the trends of the scalable experiments reported previously.

An important question is the quality of the generated BPEL4WS processes. To evaluate this aspect, we have asked one of our experienced programmers to develop manually the BPEL4WS program for the basic P&S case and we have compared the automatically generated and the hand-written solutions. As a result, we discovered that the two solutions implement the same strategy and have a similar structure. The main difference is that the automatically generated process contains a big switch implementing the preparation of the offer for the user, while the program developed manually uses additional temporary variables to perform the preparation without branching. As a consequence, the automatically generated code is larger than the manual one (22 KBytes vs. 11 KBytes). Except for this problem, that could be solved by optimizing the generation of the branches, the automatically generated code is reasonable, and rather easy to read and understand.

## ACNOWLEDGEMENTS

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".

The authors want to thank all members of the Astro project (http://www,astroproject.org/) for their collaboration and their feedback.

## 1. REFERENCES

[1] 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.

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

[3] A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. Weak, Strong, and Strong Cyclic Planning via Symbolic Model Checking. *Artificial Intelligence*, 147(1-2):35–84, 2003.

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

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