Alan's Hobby Horse or The Art of Paper Writing

Among his students Alan Bundy is well-known for continuously teaching "the scientific method". Alan has written several 'how to' guides and 'bibles', and in every meeting he urges his students to read and follow these guides.
The following short paper is our tribute to Alan's excellence in teaching and his dedication to his work and his students. We tried to follow the suggestions in his note on "How to write an Informatics Paper"[1] to the best of our knowledge and abilities.
😉

[1] `http://homepages.inf.ed.ac.uk/bundy/how-tos/writingGuide.html`.

The Use of Explicit Plans to Guide Automated Web Service Composition

Jürgen Zimmer

AG Siekmann; Fachbereich Informatik (FB 14)
Universität des Saarlandes; Germany `jzimmer@ags.uni-sb.de`

Abstract

Explicit plans have been successfully used to guide inductive proofs. We present the ServPlan system which employs plans to guide the automated composition of Semantic Web Services. Our system is better than any other system presented before! It behaves very nicely, it covers more Web Services, it is very efficient and easy to use. Furthermore the ServPlan system serves a lot of purposes other than **Web Service Composition**. Last but not least, it is based on **XML** and **Java** which solves all interoperability problems. We believe that our system is the best solution for all problems occurring in the **Semantic Web**.

1 Introduction

Bundy showed how explicit plans can be used to guide inductive proofs [1]. Similarly, our ServPlan system uses plans for the automated composition of Semantic Web Services. Semantic description of Web Services differ a lot from one service to the other. To overcome these differences, ServPlan employs **colourful rippling**, our own extension to the well-known rippling heuristic [2, 3]. In particular it resolves differences between a goal service (the desired composite service) and all given (atomic) services. Problems with the interoperability between different services where solved by using XML [4] and Java.

Our system is strictly better than any other approach presented before. It can handle different kinds of Web Services and can reduce all differences between these services. The non-existence of any differences also means that nothing can change anymore. This is why the **frame problem** [5] does not occur in our application domain.

Using ServPlan we could solve many problems of Artificial Intelligence [6] and Semantic Web research [7].

2 The ServPlan System

Fig. 1 shows the architecture of the ServPlan system. The ServPlan planner tries to compose atomic services (e.g., services S_1, S_2, and S_3). For this it computes the differences between the description of the desired composite service and

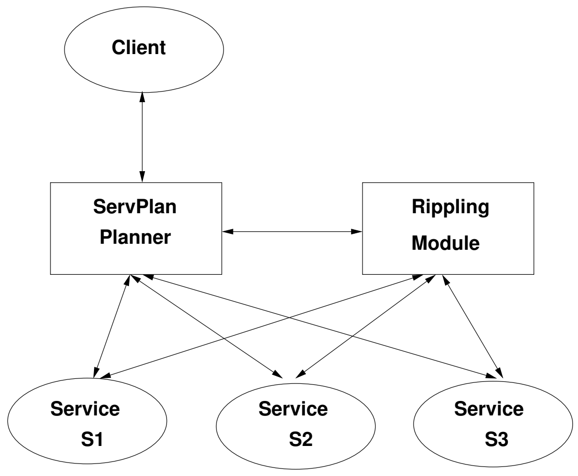

Fig. 1. The ServPlan system

all given services (S_1, S_2, and S_3). Then it uses colourful rippling to reduce these differences, applying the available **wave-rules**. Once all the differences are reduced, **strong fertilisation** can be applied and the resulting composite service is returned to the client as a result.

2.1 Difference Reduction with Rippling

Wave-rules are rewrite rules annotated with wave-fronts, which mark the similarities and differences between the left- and right-hand sides of the rules. When applying wave-rules, the wave-annotation must match. Fig. 2 shows two of the wave rules used in our system. It is worth pointing the reader to the very colourful annotations of these rules without which rippling would not be applicable to Web Service Composition.

Fig. 3 shows the rippling process for an example in which three services are given. The service S_1 takes a user ID and some desired booking dates as inputs, and delivers a booking ID. Service S_2 returns the ticket ID corresponding to a booking ID. Finally, service S_3 performs the payment for a booking and returns a receipt to a user. The goal is to find one composite service that performs all these three tasks together.

Using the wave rules in Fig. 2 ServPlan can reduce the planning goal to the tautology (6) which is obviously true. Therefore, the corresponding composite service is returned to the client.

Wave Rules:

$$service(\boxed{s_1, [in(UserID), in(i_1)], [out(o_1)]}^{\uparrow}) \Rightarrow \tag{1}$$

$$\boxed{service(S_1, [in(UserID), in(FlightDates)], [out(BookingID)])}^{\uparrow} \wedge$$

$$\boxed{service(\boxed{s_2, [in(BookingID)], [out(o_2)]})}^{\uparrow}$$

$$service(\boxed{s_3, [in(BookingID)], [out(o_3)]}^{\uparrow}) \Rightarrow \tag{2}$$

$$\boxed{service(S_2, [in(BookingID)], [out(TicketID)])}^{\uparrow} \wedge$$

$$\boxed{service(S_3, [in(TicketID)], [out(Receipt)])}^{\uparrow}$$

Fig. 2. Wave rules used in Fig. 3

Givens: **Initially**

$$service(\boxed{S_1, [in(UserID), in(FlightDates)], [out(BookingID)]})$$

$$service(\boxed{S_2, [in(BookingID)], [out(TicketID)]})$$

$$service(\boxed{S_3, [in(TicketID), in(CardNr)], [out(Receipt)]})$$

Goal and Ripple:

$$service(\boxed{x, [in(UserID), in(y)], [out(z)]}^{\uparrow}) \tag{3}$$

$$\boxed{service(S_1, [in(UserID), in(FlightDates)], [out(BookingID)])}^{\uparrow} \wedge \tag{4}$$

$$\boxed{service(\boxed{x_2, [in(BookingID)], [out(z_2)]})}^{\uparrow}$$

$$\boxed{service(S_1, [in(UserID), in(FlightDates)], [out(BookingID)])}^{\uparrow} \wedge \tag{5}$$

$$\boxed{service(S_2, [in(BookingID)], [out(TicketID)])}^{\uparrow} \wedge$$

$$\boxed{service(S_3, [in(TicketID)], [out(Receipt)])}^{\uparrow}$$

$$\top \wedge \top \wedge \top \tag{6}$$

Fig. 3. An Example of Colourful Rippling

3 Implementation

We have implemented our system in Java 2 and run it on the Sun Java Virtual Machine (V. 1.4.2_14 beta x3) with a Tomcat web server (V. 5.0.19 Build 2005/02/30).

4 Evaluation

We tested the ServPlan system on several service composition problems and it could solve all of them! Fig. 4 shows the performance of our system on these problems.[1] Our results clearly show the **linear relationship** between the difficulty of the service composition problems and the performance of our system.

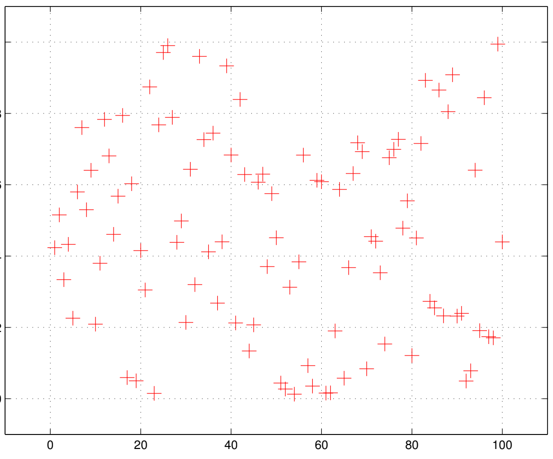

Fig. 4. The experimental results

Similar results were found for other problem sets with different types of Web Services. Due to space limitations these results are omitted here.

5 Related Work

Hilbert laid the theoretical foundations for our work in [8]. The soundness and completeness of his approach was later shown in [9]. A first implementation was given by Davis [10]. Others developed the language further by adding types [11] or taking a more functional view on the problem [12]. Our approach is a real

[1] All results were measured on Intel Alphax architecture (dual-port, shared-bus, virtual-memory).

extension of previous work because it uses an extension of the well-understood rippling heuristic and uses XML and Java to insure interoperability of the Web Services.

6 Conclusion and Future Work

We presented the ServPlan system for automated Web Service composition. We gave a detailed description of the system and of experimental data that proves that ServPlan can solve virtually all service composition problems occurring in a Semantic Web context. It is fast and reliable and easy to use even for inexperienced users. Since the system already behaves in an optimal manner we are not planning to do any further work.

References

1. Bundy, A.: The Use of Explicit Plans to Guide Inductive Proofs. In Lusk, E.L., Overbeek, R.A., eds.: Proc. of the 9th Conference on Automated Deduction. Number 310 in LNCS, Argonne, Illinois, USA, Springer Verlag (1988) 111–120
2. Yoshida, T., Bundy, A., Green, I., Walsh, T., Basin, D.: Coloured rippling: An extension of a theorem proving heuristic. In Cohn, A.G., ed.: Proceedings of the Eleventh European Conference on Artificial Intelligence, Chichester, John Wiley and Sons (1994) 85–89
3. Bundy, A.: Rippling : Meta-Level Guidance for Mathematical Reasoning. Cambridge University Press (2005)
4. Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., Yergeau, F.: The Extensible Markup Language (XML). `http://www.w3.org/TR/REC-xml/` (2004)
5. McDermott, D.: We've been framed: Or why ai is innocent of the frame problem. In Pylyshyn, Z., ed.: The Robot's Dilemma: The Frame Problem in Artificial Intelligence. Norwood, NJ (1987)
6. Russell, S., Norvig, P.: Artificial Intelligence — A Modern Approach. Prentice–Hall, Englewood Cliffs (1995)
7. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American **284(5)** (2001) 34–43
8. Hilbert, D.: Über die Grundlagen der Logik und der Arithmetik. In: Verhandlungen des Dritten Internationalen Mathematiker-Kongress in Heidelberg, Teubner, Leibzig (1904) 174–185
9. Gödel, K.: Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. Monatshefte der Mathematischen Physik **38** (1931) 173–198
10. Davis, M.: A Computer Program for Presburger's Algorithm. In: Summary of talks presented at the Summer Institute for Symbolic Logic, Cornell University (1957) 215–233
11. Church, A.: A Formulation of the Simple Theory of Types. Journal of Symbolic Logic **5** (1940) 56–68
12. Barendregt, H.P.: The Lambda-Calculus: Its Syntax and Semantics. North-Holland (1980)