A graphical user interface for service adaptation

Christian Gierds[1] and Niels Lohmann[2]

[1] Humboldt-Universität zu Berlin, Institut für Informatik,
Unter den Linden 6, 10099 Berlin, Germany
gierds@informatik.hu-berlin.de
[2] Universität Rostock, Institut für Informatik, 18051 Rostock, Germany
niels.lohmann@uni-rostock.de

Abstract. Service-oriented computing aims at composing independent services to achieve a common goal. Although very flexible, this independence may result in incompatibilities. A pragmatic approach to overcome such incompatibilities offer *adapters*. An adapter is again a service which reorganizes the message exchange in a service composition to avoid incompatibilities.
Given a set of domain-specific message transformation rules, adapters can be generated fully automatically. This paper presents a graphical user interface to support the systematic design of these transformation rules.

1 Introduction

Service-oriented computing [10] aims at replacing large monolithic systems by a composition of *services*. By abstracting from underlying technologies and implementations, it is possible to focus on the functionality of a service and to reuse it in other compositions. Consequently, services can be designed independently from the compositions they are used in, which in turn allows for faster production cycles at lower costs. A downside of this flexibility is the possible incompatibility of independently designed services. To avoid the redesign of incompatible services, an *adapter* (sometimes called mediator) can resolve incompatibilities by manipulating the communication protocol between the incompatible services. State-of-the-art techniques [1,2,3,5,9,4,11,6] allow to generate adapters automatically given a set of domain-specific message transformation rules.

So far, the design of such transformation rules was out of scope of most existing adapter generation approaches, and of course transformation rules can be formulated independently of a concrete service composition. However, it is likely that the design of such rules can be accelerated if the services to be adapted is taken into account. This paper follows this idea and presents an approach to iteratively create proposals for the designer of semantic message transformation rules. These proposals are derived by diagnosing behavioral incompatibilities. The approach is complete; that is, if services can be adapted using some rule set, then this set can be constructed.

The rest of this paper is organized as follows. The next section briefly sketches the automatic generation of adapters and introduces a running example. It further discusses how transformation rule proposals can be derived from diagnosed incompatibilities. Section 3 presents the main contribution of this paper, a Web-based graphical user interface for the iterative construction of transformation rules. Finally, Sect. 4 discusses possible future extensions and concludes the paper.

2 Adapter generation

We shall briefly outline the basic concepts of an adapter generation algorithm and its meaning for finding transformation rules in this section.

2.1 Synthesis using message transformation rules rules

For adapting two services A and B several approaches agree on using *message transformation rules* (creating, removing, or changing messages) [1,2,3,5,9,4,11,6], which handle semantical incompatibilities.

We concentrate on the approach of Gierds et al. [6], consisting of two: They model transformation rules as an artifact called *engine* E. Then they try to synthesize a *controller* C, such that the composition of A, B, E, and C behaves according to a certain correctness criterion (e. g., deadlock freedom). The composition of E and C thus yields an adapter for A and B and ensures semantical and behavioral correctness of the two services.

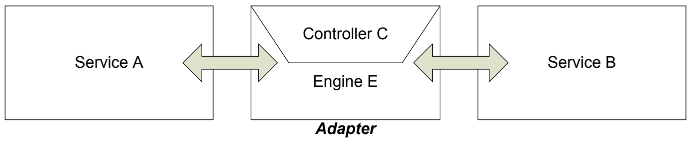
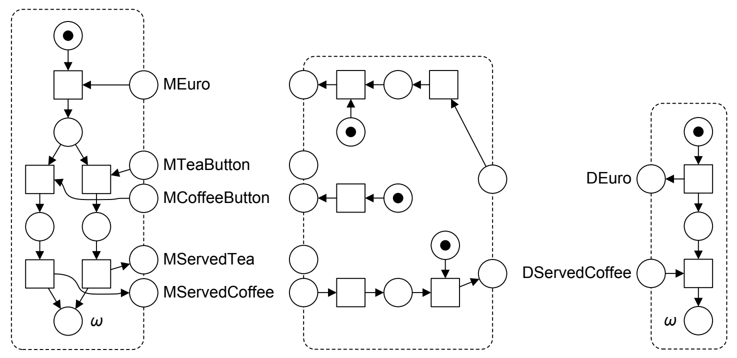


Fig. 1. Two services A and B and an adapter (engine E and controller C) in the middle

Figure 1 depicts a schema of this approach. The two services A and B communicate via the adapter in the middle. As it is indicated, an adapter comprises two parts: The engine E implements the message transformation rules and thus ensures semantically correctness. The controller C ensures correct behavior; that is, the correct order of applying rules and sending messages to the services.

Figure 2 shows a small example based on open nets [7], an extension of classical Petri nets. Interface places are positioned on the dashed border of a net. As running example, the model of a beverage vending machine is depicted on the left (cf. Fig. 2(a)). After receiving a Euro (MEuro), either the tea (MTeaButton) or the coffee button (MCoffeeButton) must be pressed. Afterward the appropriate beverage is delivered (MServedTea and MServedCoffee, resp.). On the right (2(c)), a coffee drinker provides a Euro (DEuro), then forgets to press a button, and



(a) Beverage vending machine (b) Adapter (c) Coffee drinker

Fig. 2. The two services to be adapted with an adapter

waits for its coffee (DServedCoffee). Obviously, this service is not compatible to the vending machine. To overcome this incompatibility, the adapter in Fig. 2(b) simply transforms a DEuro message to an MEuro message and MServedCoffee to DServedCoffee, which seems obvious concerning the names. Further it creates a MCoffeeButton message. Due to structural reduction, we may identify the controller part only by the initially marked places, allowing each rule to be applied exactly once.

To synthesize such an adapter automatically, the before-mentioned three rules must be provided as input to the synthesis algorithm.

2.2 Finding additional rules

As mentioned, one of the essential parts of the adapter approach is the set of message transformation rules. Although correct in isolation (in the example, there exist compatible drinker and vending machine services, resp.), two services may only be adaptable if a certain set of rules is provided. So whenever the synthesis algorithm fails to create an adapter, this is caused by missing rules.

Previous approaches almost totally rely on Semantic Web technologies for providing rules. We will briefly sketch an idea on how to extend the set of transformation rules by behavioral diagnosis. During controller synthesis deadlocks will be reached if no deadlock free controller exists. These deadlocks provide valuable information, how an additional rule might look like. The setting does not allow to change one of the services, but we are free to add as many new rules as we like. Let m be a deadlock marking, then we can analyze which messages remain in the engine, thus are *pending* and could be transformed. Further we check whether one of the services could continue if we provided a certain message, so we check which messages are *required.* A new rule then may transform pending into required messages. Consequently, m will no longer be a deadlock marking, because we can apply the newly added transformation rule now (which behaves like a transition added to a net, which is enabled by the pending messages). This step can be repeated until we find a controller and therefore an adapter, or until we are no longer able to add meaningful transformation rules.

For our example in Fig. 2, starting with an empty set of transformation rules, our proposed algorithm will state that DEuro is pending (in Fig. 2(c) the appropriate transition is activated and thus fired), MEuro is required, and thus we may add the rule transforming MEuro to DEuro. In the next step (as shown in Fig. 3) we will see, that MTeaButton and MCoffeeButton are required. After providing a corresponding rule, we will see, that MServedCoffee is pending, and DServedCoffee is required. Finally the rule set is sufficient and we gain an adapter for our example.

In the given example the single steps are straightforward. For more complex examples, the number of deadlocks as well as the number of details for each deadlock grows significantly. Thus we need a good representation for this kind of information.

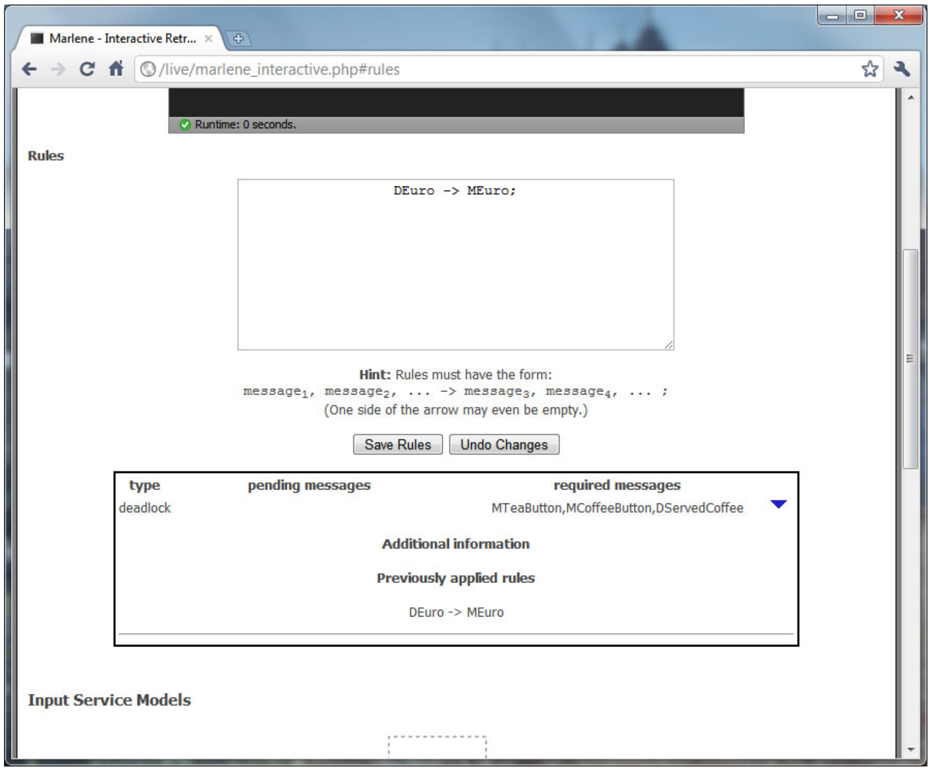
3 Using the Web as graphical user interface

Interactive approaches highly benefit from a concise way of presenting information. A user must be able to quickly access all relevant information. Graphical solutions with means to highlight or hide information based on a user's demands clearly excel console applications in this point. Marlene as single purpose tool has already been integrated into *service-technology.org/live* [8], which is our platform to demonstrate the functionality of our tools and allow a user to perform more complex tasks involving several of our tools by simply using a Web browser. The previously described interactive approach has also been integrated there and can be tested at the URL `http://service-technology.org/live/marlene`.

In an interactive approach, we do not only need to present the input and output artifacts, but also intermediate information which shall enable the user to make a next step. In our case, we have to list all possible suggestions for adding new transformation rules without showing all details at once and thus confusing the user.

Figure 3 shows the essential part for our approach: an editing field for transformation rules and below a table with information on all deadlocks, which may help in providing additional rules. Additionally, but not depicted here, the services are visualized. We divide the table in the following columns:

- *type* might either be deadlock or a livelock (in case we want an adapter ensuring also livelock freedom);
- the *pending messages*, which can be used in a rule on the left side;
- the *required messages*, of which at least one must be provided for resolving a deadlock in one of the services
- the *triangle* button, for showing additional information on a deadlock or livelock


Marlene - Interactive Retr...
/live/marlene_interactive.php#rules
Save Rules
Undo Changes

Runtime: 0 seconds.

Rules

```
DEuro -> MEuro;
```

Hint: Rules must have the form:
$message_1$, $message_2$, ... -> $message_3$, $message_4$, ... ;
(One side of the arrow may even be empty.)

type	pending messages	required messages
deadlock		MTeaButton,MCoffeeButton,DServedCoffee

Additional information

Previously applied rules

DEuro -> MEuro

Input Service Models

Fig. 3. Screenshot of interactive site

The additional information might state, that one of the services is already in a final state thus needing no further attention, and which rules have been applied prior to reaching a certain deadlock.

We have decided to initially show only the first line for each deadlock (the line starting with *deadlock*, thus hiding all additional information at first). As we can see in Fig. 3, providing all available information on a deadlock in a clear way requires a lot of space. Presenting a larger number of deadlock then would almost immediately require the user to scroll the page. This would clearly hamper deciding which deadlock to resolve, because a direct comparison of deadlocks would always depend on scrolling.

In our understanding, pending and required messages are the most important information for a certain deadlock. Thus showing only this piece of information should be sufficient in most case. By clicking the triangle at the end of a line the user gets additional information as described above.

The user is frfee to add and change the rules arbitrarily in the text field. By clicking *Save Rules* the page is updated and information based on the new rule set are provided. Finally, if a sufficient set of rules was added, the generated adapter is presented.

4 Conclusion and outlook

We have presented a first idea for the interactive retrieval of transformation rules in the setting of service adaptation. We have also focused on an appropriate visualization of information. First tests indicate that interaction as described here with the proposed degree of initial information offers easy access to the approach. This of course is only a first step.

First, the algorithm for finding new transformation rules has to be described in detail and we have to proof its feasibility in adapter generation (i. e., that when an adapter exists, the algorithm leads to a corresponding set of rules). Second, we have to evaluate acceptance of the Web site. Only feedback of real users playing through real-word examples will give us valuable hints on how to improve presentation of our tool.

Especially the order of different deadlocks might facilitate decision, which one to resolve — the higher the position of a deadlock, the more likely it will be considered. Here we have to find heuristics based on user behavior and its reason to prefer certain deadlocks. Also highlighting certain situations (e. g., both services are already in a final state, but superfluous messages must be removed) might help a user to pick more goal leading deadlocks.

Although not having finished the approach, yet, using a Web front-end for our prototype allows us to test our approach from the very beginning and distribute it easily, thus gaining valuable feedback from prospective users.

References

1. Benatallah, B., Casati, F., Grigori, D., Motahari Nezhad, H.R., Toumani, F.: Developing adapters for web services integration. In: CAiSE. pp. 415–429 (2005)
2. Bracciali, A., Brogi, A., Canal, C.: A formal approach to component adaptation. Journal of Systems and Software 74(1), 45–54 (Jan 2005)
3. Brogi, A., Canal, C., Pimentel, E., Vallecillo, A.: Formalizing web service choreographies. In: WS-FM'04. ENTCS, vol. 105, pp. 73–94 (2004)
4. Brogi, A., Popescu, R.: Automated generation of BPEL adapters. In: ICSOC. pp. 27–39 (2006)
5. Dumas, M., Spork, M., Wang, K.: Adapt or perish: Algebra and visual notation for service interface adaptation. In: Business Process Management. pp. 65–80 (2006)
6. Gierds, C., Mooij, A.J., Wolf, K.: Reducing adapter synthesis to controller synthesis. Transactions on Services Computing (accepted for publication) (2010)
7. Kindler, E.: A compositional partial order semantics for Petri net components. In: ATPN'97. LNCS, vol. 1248, pp. 235–252 (1997)
8. Lohmann, N.: service-technology.org/live - replaying tool experiments in a Web browser. In: BPM 2010 Demonstration Track. CEUR Workshop Proceedings, vol. 615, pp. 64–68. CEUR-WS.org (2010)
9. Motahari Nezhad, H.R., Benatallah, B., Martens, A., Curbera, F., Casati, F.: Semi-automated adaptation of service interactions. In: WWW. pp. 993–1002 (2007)
10. Papazoglou, M.P.: Web Services: Principles and Technology. Pearson - Prentice Hall, Essex (Jul 2007)
11. Wang, K., Dumas, M., Ouyang, C., Vayssière, J.: The service adaptation machine. In: ECOWS. pp. 145–154 (2008)