

On Compatibility of Web Services

Axel Martens*

Humboldt-Universität zu Berlin, Department of Computer Science,
Unter den Linden 6, D-10099 Berlin, Germany

Abstract

To an increasing extend business processes run across the borders of individual enterprises. Thus, there is a need to map each local subprocess into a self-contained component; a distributed business process arises from composition of such component via standardized communication protocols.

The *Web service approach* provides a standardized, platform independent and widely accepted concept of components and composition for distributed systems of all kinds. This approach comes along together with group of technologies to describe precisely the structure of one Web service and the composition of a set of Web services. Although the technological basement is given, there is a lot of open questions, e. g. *semantic compatibility* of two Web services.

This paper abstracts from concrete syntax of any proposed language definition. Instead, we apply Petri nets to model Web services. Thus, we are able to reason about essential properties, e. g. *usability* of a Web service – our notion of a quality criterion. Based on this framework, we discuss and define a criterion for semantic compatibility of Web services.

1 Introduction

In this paper, we regard Web services as components of distributed business processes. The aim is to establish locally provable quality criteria for Web services, that guarantee global properties of the business processes. Thus, we focus on structure and behavior rather than technical aspects.

1.1 Business Processes

For a couple of years business processes have come the center of enterprise information systems. Tasks like modeling, analyzes and execution of business processes are summarized under the notion of workflow management. A *business process* consists of a self-contained set of causally ordered activities. Each activity performs a certain functionality, produces and consumes data, requires or provides resources and is executed manually or automatically. There are many well established modeling languages and Petri nets are among them.

To an increasing extend business processes run across the borders of individual enterprises. The distribution by space and the organizational independence of participating companies prohibits the application of traditional workflow management. Instead, the *distributed business process* is divided into self-contained components, each with a defined interface. The dynamical composition of these components forms the business processes, whereas each components can be autonomously implemented.

To hide the heterogeneous structures within the participating companies, there is a need for standardized concept of components an composition. The Web service approach provides group of technologies that meet this requirement.

1.2 Web services

A *Web service* is a self-describing, self-contained modular application that can be described, published, located, and invoked over a network, e. g. the World Wide Web. A Web service performs an encapsulated function ranging from a simple request-reply to a full business process.

The *Web service approach* provides a standardized, platform independent and widely accepted

*E-mail: martens@informatik.hu-berlin.de

concept of components and composition, even for distributed business processes. A local subprocess is implemented by one Web service. Hence, a distributed business process can be realized by composition of a set of Web services.

Instead of one new specific technology, the Web service approach provides group of closely related, established and emerging technologies to model, publish, find and bind Web services – called the *Web service technology stack* [5]. This paper is concerned with the application of Web service approach towards the area of business processes. Thus, we focus on the behavior of a Web service, defined by its internal structure by help of the *Business Process Execution Language for Web services BPEL4WS* [4].

Most of the Web services technologies are still in the standardization process. Because there are inconsistencies and ambiguities left in the initial drafts, specification will be changed several times until a consistent status is reached. To address the core problems of distributed business processes, we prescind from the technical aspects and use Petri nets to model the behavior of Web services.

1.3 Compatibility

The Web service approach provides a technical framework to implement distributed business processes that guarantees a minimum of *syntactical* consistency. But there is a need for more analysis methods, e. g. the proof of *semantic* compatibility. Lets consider a Web service of an online shop and a Web service of a customer: The online shop waits for payment before sending the product. The customer behaves in the opposite way. Both services have a syntactically compatible interface but the resulting distributed process leads to a deadlock.

To address those kinds of problems, we abstract from the technical aspects and apply Petri nets to model the behavior of Web services. Based on this formalism, we are able to discuss and define *usability* of a Web service – our notion of a quality criterion – and derive further properties: e. g. *compatibility* of two Web services. The current paper is part of larger framework for modeling and analyzing distributed business processes by help of Web services [7].

Due to our abstract view on behavior and structure of Web services, the results presented here can be adopted easily to every concrete modeling language, for instance BPEL4WS [10].

1.4 Structure of this paper

The following Section 2 presents our modeling approach. Therefore, we give a short introduction to Petri nets and present the notion of a *workflow module* – the model of a Web service. Moreover, we show the composition of a workflow modules and discuss the notion of *usability* – an essential property of a Web service.

The composition of two Web services yield either another (composed) Web service or a closed system realizing a distributed business process. In Section 3, we discuss the notion of *semantic compatibility* for both cases and give an adequate definition.

Finally, section 4 points to other methods that are part of our framework. Most of them are already implemented within a free available prototype.

2 Modeling

Our modeling approach is based on Petri nets. Besides the intuitive graphical representation, Petri nets allow an effective analysis. By help of an example, we present the notion of a *workflow module* and show their composition. Thereby we give references to the Petri net theory. A basic introduction into the application of Petri nets to business processes can be found in [3].

2.1 Modeling Web services

In this paper, a distributed business process comes into existence because of composition of Web services. Each of these Web services represents a local subprocess. Figure 1 shows the model of such a subprocess – the Web service of a travel agency.

A business process consists of a self-contained set of activities which are causally ordered. A Petri net $N = (P, T, F)$ consists of a set of *transitions* T (boxes), a set of *places* P (ellipses) and a *flow relation* F (arcs) [9]. A transition represents an active element, i. e. an activity (e. g. Get Itinerary). A place represents a passive element, i. e. a state between activities, a resource or a message channel (e. g. Itinerary).

Unlike a lift controller, a business process has a defined beginning and terminates after the execution of a finite number of activities. A *workflow net* is a special Petri net, that has two distinguished places $\alpha, \omega \in P$ to denote the begin (α) and the

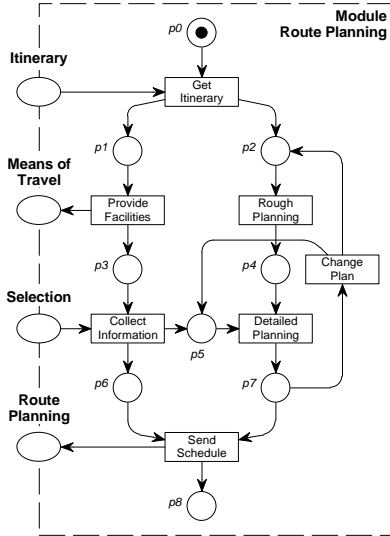


Figure 1: A workflow module

end (ω) of a process [1]. Thus, we model a business process in terms of a workflow net.

Given a workflow net, it is possible to reason about its quality. The notion of *soundness* [2] is an established quality criterion for workflow nets. Basically, soundness requires each initiated process to come to a proper final state. Additionally, each transition should be relevant, i. e. there should be at least one behavior of the process in which this transition participates.

The second requirement is reasonable, if a business process was modeled from scratch. In our approach a business process arises from composition. Thus, it might be reasonable to call a system “sound”, although not all functionality of one specific component is used in that system. Hence, we use the slightly different notion.

Definition 2.1 (Weak soundness).

A workflow net is called *weak sound*, iff:

- (i) For each reachable marking (starting at $[\alpha]$) the final marking $[\omega]$ is reachable.
- (ii) For each reachable marking m such that $m \geq [\omega]$ holds $m = [\omega]$. *

The requirements of this definition are a subset of the requirements within the definition of soundness. Thus, each sound workflow net is weak sound. For more information, e. g. the precise definition of *reachable marking* see [7].

A Web service consists of internal structures that realize a local subprocess and an interface to com-

municate with its environment, i. e. other Web services. Thus we model a Web service by help of a workflow net supplemented by an interface, i. e. a set of places representing directed message channels. Such a model we call *workflow module*. For reasons of simplicity we require each transition to be connected at most to one type of interface places (cf. [6]).

Definition 2.2 (Module).

A finite Petri net $M = (P, T, F)$ is called *workflow module* (*module* for short), iff:

- (i) The set of places is divided into three disjoint sets: *internal places* P^N , *input places* P^I and *output places* P^O .
- (ii) The flow relation is divided into *internal flow* $F^N \subseteq (P^N \times T) \cup (T \times P^N)$ and *communication flow* $F^C \subseteq (P^I \times T) \cup (T \times P^O)$.
- (iii) $\mathcal{N}(M) = (P^N, T, F^N)$ is a workflow net.
- (iv) Non of the transitions is connected both to an input place and an output place. *

Within a module, we call the workflow net $\mathcal{N}(M)$ the *internal process* and the tuple $\mathcal{I}(M) = (P^I, P^O)$ its *interface*. Figure 1 shows a workflow module. The internal process is triggered by an incoming *Itinerary*. Then the control flow splits into two concurrent threads. On the left side, the available *Means of travel* are offered to the customer and the service awaits his *Selection*. Meanwhile, on the right side, a *Rough Planning* may happen. The *Detailed Planning* requires information from the customer. Finally, the service sends a *Route Planning* to the customer.

2.2 Composing Web services

A distributed business process is realized by the composition of a set of Web services. We will now define the pairwise composition of workflow modules. Because this yields another workflow module, recurrent application of pairwise composition allows us to compose of more than two modules.

Figure 2 shows once more the workflow module *Route Planning*. Additionally, a module *Default* is presented. The purpose of this model is to unburden the customer from making a selection. Thus, the module *Default* consumes the message on available *Means of travel* and return a default *Selection*.

Obviously, both modules can be composed. As a precondition for composition, we will define the

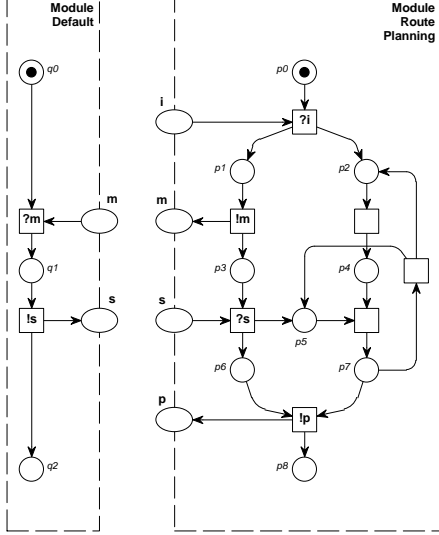


Figure 2: Syntactically compatible modules

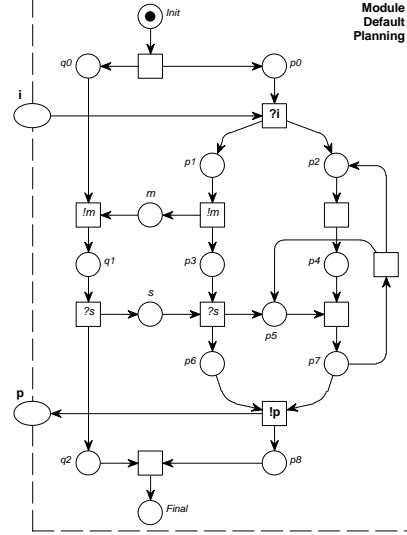


Figure 3: A composed workflow module

property of *syntactical compatibility* of two modules.

Definition 2.3 (Syntactical compatibility).

Two workflow modules A and B are called *syntactically compatible*, iff the internal processes of both modules are disjoint, and each common place is an output place of one module and an input place of the other. *

As shown in Figure 2, two syntactically compatible modules do not need to have completely matching interfaces. They might even have completely disjoint interfaces. In that case, the reason of composition is at least dubious.

When two modules are composed, the common places are merged and the dangling input and output places become the new interface. To achieve a correct module as the result of the composition, we need to add new components for initialization and termination. Figure 3 shows the composed module *Default Planning*.

Definition 2.4 (Composed system).

Let $A = (P_a, T_a, F_a)$ and $B = (P_b, T_b, F_b)$ be two syntactically compatible modules. Let $\alpha_s, \omega_s \notin (P_a \cup P_b)$ two *new* places and $t_\alpha, t_\omega \notin (T_a \cup T_b)$ two *new* transitions. The *composed system* $\Pi = A \oplus B$ is given by (P_s, T_s, F_s) , such that:

- $P_s = P_a \cup P_b \cup \{\alpha_s, \omega_s\}$
- $T_s = T_a \cup T_b \cup \{t_\alpha, t_\omega\}$

- $F_s = F_a \cup F_b \cup \{(\alpha_s, t_\alpha), (t_\alpha, \alpha_a), (t_\alpha, \alpha_b), (\omega_a, t_\omega), (\omega_b, t_\omega), (t_\omega, \omega_s)\}$

If the composed system contains more than one components for initialization and termination, the corresponding elements are merged. *

Figure 3 shows the model of a composed Web service *Route Planning* \oplus *Default*. This service offers a simpler interface to the customer. Syntactically, the result of the composition is again a workflow module.

Corollary 2.1 (Composing modules):

Whenever A and B are two syntactically compatible workflow modules, the composed system $\Pi = A \oplus B$ is a workflow module too. *

This corollary can be easily proven. We therefore omit the proof here, it can be found in [7]. The components for initialization and termination are added to an composed workflow module for reasons of syntactical correctness. If we compose more than to modules, it is important to guarantee associativity of pairwise composition. Hence, if one module already contains such syntactic components, we merge the corresponding elements while composition. The next section presents an example.

2.3 Usability of Web services

In this paper, we aim at generating a *model* of the business process by composing the *models* of the

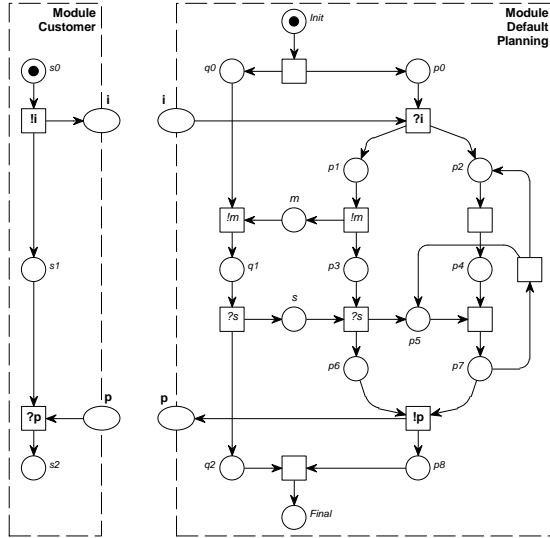


Figure 4: A module and its environment

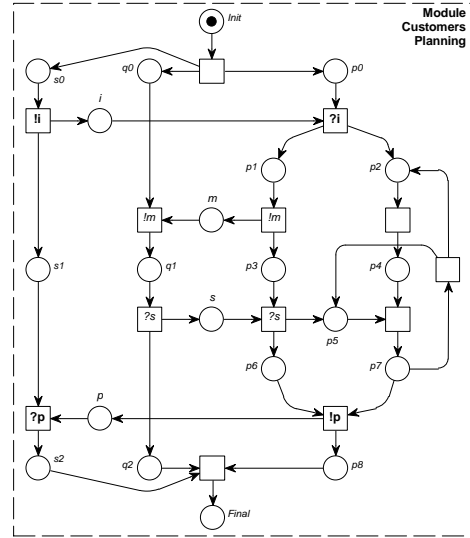


Figure 5: A composed workflow module

participating Web services. The composition of two workflow modules A and B represents a business process, if the composed system $\Pi = A \oplus B$ has an empty interface, i.e. Π is a workflow net. In that case, we call A an *environment* of B .

Definition 2.5 (Environment).

Let M and U be two syntactically compatible modules. U is called *environment* of M , iff each output place of U is an input place of M and vice versa. *

If a module U is an environment of M , obviously M is an environment of U , too. Figure 4 shows the composed workflow module *Default Planning*. Additionally, the module *Customer* is shown, which is an environment of the module *Default Planning*.

In the real world, the environment of a Web service may consist of several other Web services. If we want to reason about that specific Web service, we don't have any assumption on its potential environment. Thus, without loss of generality, we may consider its environment as one, arbitrary structured Web service.

Based on the notion of an environment we are able to discuss *usability* – the quality of a given workflow module: Obviously, the purpose of a workflow module is to be composed with an environment such that the resulting system is a proper workflow net. We require the resulting workflow net to be *weak sound*. For a given module there exist infinitely many environments. The question is: To call the given module *usable*, how many of these

environments have to form a weak sound composed system together with this module?

Figure 5 shows a workflow net which is build from the modules shown in Figure 4. The soundness property of this net can be easily proven. Thus, this net is weak sound, too.

One might think of calling a module *usable*, iff *all* possible environments form together with that module a weak sound composed system. But, this definition is not appropriate: Because of the following proposition, no module at all would be usable.

Proposition 2.2 (Malicious environment):

For each workflow module M there is a malicious environment U such that the composed system $M \oplus U$ is not weak sound. *

The proof together with a set of illustrating examples can be found in [7]. Because of that we give another, more appropriate definition of usability:

Definition 2.6 (Usability).

Let M be a workflow module.

- (i) An environment U *utilizes* the module M , iff the composed system $\Pi = M \oplus U$ is weak sound.
- (ii) The module M is called *usable*, iff there exists at least one environment U , such that U utilizes M . *

Concerning this definition, the module *Customer* utilizes the module *Default Planning* and vice versa. Thus, both modules are called usable. Intuitively,

we want to call the modules Default Planning and Customer *semantically compatible*. In the following section, we will derive a definition of *semantic compatibility* that meets our intuition.

3 Compatibility

The notion of *syntactical compatibility* – defined in the previous section – is a precondition for the composition of two workflow modules and can be checked easily. But as the example of the online shop and its customer from the very beginning of this paper has shown, this precondition is not sufficient for the soundness of the composed system. Hence, there is a need for *semantic compatibility*.

To deal with this notion, we discuss two different scenarios: the composition of a workflow module and its environment and the composition of two workflow module with dangling interface places. Anyhow, we will present *one* adequate definition that covers both cases. Finally, we take a look on the composition of more than two workflow modules.

3.1 Compatible environments

Again, we take a look on the modules Customer and Default Planning shown in Figure 4. These two syntactically compatible modules have a completely matching interface. Thus, module Customer is an environment of modules Default Planning (and vice versa) and the composition of these two yields a workflow net – shown in Figure 5.

Intuitively, we want to call two workflow modules *semantically compatible*, if the composed system is *error-free* in some sense. Concerning workflow nets, we have introduced the notion of *weak soundness* that guarantees the absence of serious errors. Thus, the weak soundness of the composed system should be sufficient for the semantic compatibility of its components.

But, we cannot apply the notion of weak soundness to a workflow module with a non-empty interface. Hence, we chose a different approach and express afterwards the relationship between weak soundness and semantic compatibility within a theorem.

3.2 Compatible modules

We take a look on the modules Default and Route Planning shown in Figure 2. If these two modules

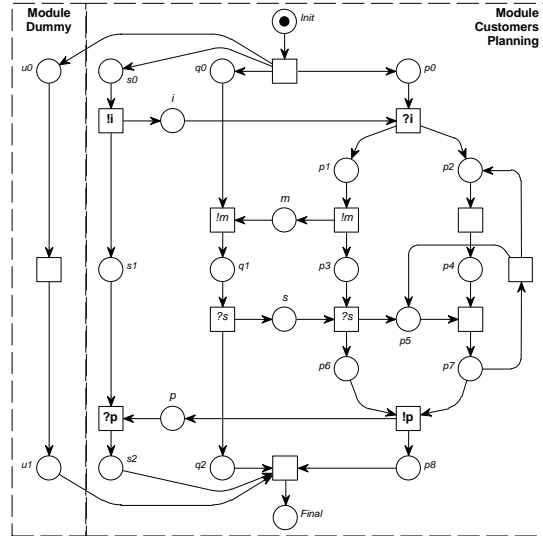


Figure 6: Workflow net with environment

are composed, the result is again a workflow module that requires an input (Itinerary) and produces an output (Route Planning) – shown in Figure 3.

Concerning workflow modules, we have introduced the notion of *usability* that guarantees the absence of serious errors. Thus, the usability of the composed module should be sufficient for the semantic compatibility of its components. We propose the following definition.

Definition 3.1 (Semantic compatibility).

Let A and B be two syntactically compatible modules. A and B are called *semantically compatible*, iff the composed system $A \oplus B$ is usable. *

Because the module Customer the is an utilizing environment, the module Default Planning is proven to be usable. Thus, according to Definition 3.1, the modules Default and Route Planning are semantically compatible. Moreover, the usability of each workflow module is an precondition for their compatibility.

Theorem 3.1 (Usability & compatibility).

Let A be a workflow module. If A is not usable, there exists no other workflow module B such that A and B are semantically compatible. *

Proof (Theorem 3.1): If A and B was semantically compatible, the composed system $A \oplus B$ would be usable. That means, there would exist an utilizing environment U , i. e. the composed system $A \oplus B \oplus U$ would be weak sound. In such a case

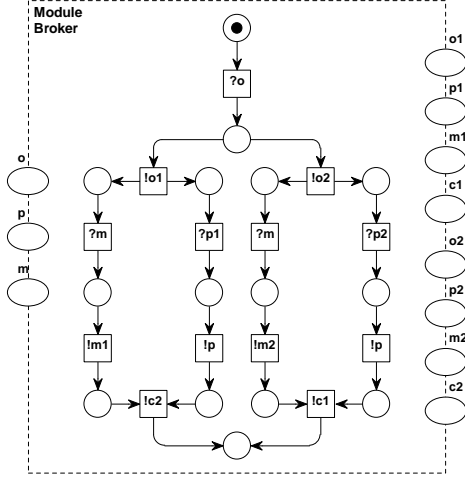


Figure 7: One common broker

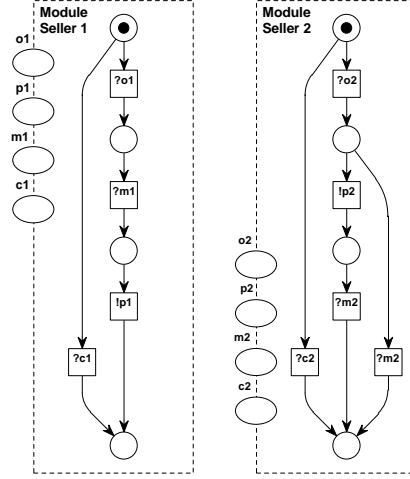


Figure 8: Two different seller

the module $B \oplus U$ would be an utilizing environment of A . Hence, A would be usable itself. This is a contradiction of the precondition. \square

One might wonder, but Definition 3.1 does also cover the composition of a module and its environment. Lets have another a look at the composed system Customers Planning shown in Figure 5. The system is a workflow net, i.e. a workflow module with an empty interface. For each workflow module we can find an environment: In case of the module Customers Planning, we chose a very simple environment, of course with an empty interface, too. Figure 6 shows such an environment – called Dummy, already composed with the module Customers Planning.

The module Dummy is a workflow net and consists just of the two distinguished places (denoting the begin and the end) and one transition in between. Obviously, this workflow net is weak sound. The parallel composition of two weak sound workflow nets yields again a weak sound workflow net. The following theorem meets our intuition on the relationship between weak soundness and semantic compatibility.

Theorem 3.2 (Soundness & compatibility).

Let M be a workflow module and let U be an environment of M . M and U are semantically compatible, iff $M \oplus U$ is weak sound. $*$

Proof (Theorem 3.2): If $M \oplus U$ is weak sound and we compose $M \oplus U$ with another weak sound workflow net D (e.g. with the workflow module Dummy), the resulting composed system is weak

sound, too. Thus, the workflow net D is an utilizing environment of $M \oplus U$, i.e. M and U are semantically compatible.

If M and U are semantically compatible, there is a module D such that $M \oplus U \oplus D$ is weak sound. Because $M \oplus U$ has an empty interface, D has no impact on the behavior of $M \oplus U$. Thus, $M \oplus U$ must be weak sound itself. \square

We have presented an uniform definition of semantic compatibility that is based on *usability* of workflow modules. This property can be decided effectively. A more detailed characterization including the algorithms can be found in [7]. Because of the two theorems presented here, our definition meet the intuitive notion of compatibility concerning both cases of composition.

3.3 Multiple composition

In the previous section we have focussed on the compatibility of two given workflow modules. But, in many cases problems are not visible until more than two modules are composed. Thus, we want to characterize the semantic compatibility of a set of modules. As the following example shows, pairwise compatibility is a necessary precondition, but not sufficient.

We consider an e-commerce example: Figure 8 shows the workflow modules of two different sellers. Both sellers do their business in different ways, but they are linked together via a common broker – shown in Figure 7. After Seller 1 has got the order (via channel $o1$), he waits for the money (channel $m1$) and send the requested product, afterwards

(channel $p1$). Seller 2 sends the product immediately after he has received the order. In case he received the money first, he would not send the product at all. Both sellers will terminate their process, if they received the signal to cancel (via channel $c1$ or $c2$). Obviously, a customer could directly utilize both sellers without any problems.

Figure 7 shows the module of the **Broker**. He awaits the order from the customer (via channel o) and chooses a seller, to whom he forwards the order (via channel $o1$ or $o2$). Afterwards he redirects the money and the product. Finally, he sends the signal to cancel to the non chosen seller.

In isolation, the workflow module **Broker** is usable, too. Moreover, it can be proven that two of the three module each with other form an usable composed system. Thus, the three modules are pairwise semantically compatible. But the composition of all three yields a non-usable module, i. e. there is no workflow module of a customer that utilizes the module **Broker** \oplus **Seller 1** \oplus **Seller 2**: If the customer waits for the product (after he has send his order) and the **Broker** has chosen **Seller 1** (waiting for the money first), the system ends in a deadlock. In case the customer has send the money directly after his order, the **Broker** might choose **Seller 2** and the money is gone; the system ends in a deadlock, too.

This example shows: If a set of workflow modules has to be composed, it is not sufficient to prove semantic compatibility in pairs. It is necessary to verify the usability of the overall composed system.

4 Summary

In this paper, we have presented an approach to decide *compatibility* of Web services – a fundamental requirement for the realization of distributed processes.

Most of the Web services technologies are still in the standardization process and the specifications will be changed several times until a consistent status is reached. Hence, we have chosen Petri nets to model business processes and Web services. As we have shown, Petri nets are an adequate modeling language for Web services.

Each Web service has an interface and an internal structure. We have introduced the notion of a *workflow module* – a workflow net with a set of interface places. Based on this formalism, we have defined the notion of *usability* – our criterion for

the quality of a workflow module. This property can be checked effectively, the algorithms are implemented within an available prototype [8].

Two module are composed by fusion of common interface places. Therefore, *syntactical compatibility* is a necessary precondition. But this precondition is not sufficient for the soundness of the composed system. Hence, there is a need for a notion of *semantic compatibility*: Two workflow modules are called semantically compatible, iff the composed system is usable.

The current work is part of a framework for modeling and analyzing distributed processes that arise from composition of Web services [7]. Beside the results presented here, the notion of usability is the basis for further investigations on Web services. On the one hand, the analysis offers a starting point for the simplification of Web service models and for re-engineering of such components. On the other hand, the equivalence of two Web services can be decided. This is exceedingly important for a dynamic exchange of components within a running system: Does the new component behave exactly the way the replaced component did?

Due to our abstract view on behavior and structure of Web services, the results presented here can be adopted easily to every concrete modeling language, for instance BPEL4WS [10].

References

- [1] W. M. P. van der Aalst. A class of petri net for modeling and analyzing business processes. Computing science report 95/26, Eindhoven University of Technology, 1995.
- [2] W. M. P. van der Aalst. Structural characterizations of sound workflow nets. Computing science report 96/23, Eindhoven University of Technology, 1996.
- [3] W. M. P. van der Aalst. The application of petri nets to workflow management. *Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.
- [4] BEA, IBM, Microsoft, and SAP. *BPEL4WS–Business Process Execution Language for Web Services*. Version 1.1, July 2002.
- [5] Karl Gottschalk. *Web Services architecture overview*. IBM developerWorks, Whitepaper, September 2000.

- [6] E. Kindler, A. Martens, and W. Reisig. Interoperability of workshop applications – local criteria for global soundness. In W. van der Aalst, J. Desel, and A. Oberweis, editors, *Business Process Management*. Springer-Verlag, LNCS 1806, 2000.
- [7] Axel Martens. *Verteilte Geschäftsprozesse – Modellierung und Verifikation mit Hilfe von Web Services*. Phd thesis, Humboldt-Universität zu Berlin, 2003.
- [8] Axel Martens. WOMBAT4WS – *Workflow Modeling and Business Analysis Toolkit for Web Services*. Humboldt-Universität zu Berlin, Manual, 2003. <http://www.informatik.hu-berlin.de/top/wombat4ws/>.
- [9] W. Reisig. *Petri Nets*. Springer-Verlag, Berlin, Heidelberg, New York, Tokyo, eatcs monographs on theoretical computer science edition, 1985.
- [10] Wolfgang Reisig and Axel Martens. *Task Force – Geschäftsprozesssprache BPEL4WS*. Humboldt-Universität zu Berlin, Lehrstuhl Theorie der Programmierung, 2003.