A Petri Net-based Model for Web Service Composition

Rachid Hamadi **Boualem Benatallah**

School of Computer Science and Engineering
The University of New South Wales
Sydney NSW 2052, Australia
{rhamadi,boualem}@cse.unsw.edu.au

Abstract

The Internet is going through several major changes. It has become a vehicle of Web services rather than just a repository of information. Many organizations are putting their core business competencies on the Internet as a collection of Web services. An important challenge is to integrate them to create new value-added Web services in ways that could never be foreseen forming what is known as Business-to-Business (B2B) services. Therefore, there is a need for modeling techniques and tools for reliable Web service composition. In this paper, we propose a Petri net-based algebra, used to model control flows, as a necessary constituent of reliable Web service composition process. This algebra is expressive enough to capture the semantics of complex Web service combinations.

Keywords: Web services, Petri net, Web service composition.

1 Introduction

In order to survive the massive competition created by the new online economy, many organizations are rushing to put their core business competencies on the Internet as a collection of Web services for more automation and global visibility. The concept of *Web service* has become recently very popular, however, there is no clear agreed upon definition yet. Typical examples of Web services include on-line travel reservations, procurement, customer relationship management (CRM), billing, accounting, and supply chain. In this paper, by Web service (or simply service) we mean an autonomous software application or component, i.e., a semantically well defined functionality, uniquely identified by a Uniform Resource Locator (URL).

The ability to efficiently and effectively share services on the Web is a critical step towards the development of the new online economy driven by the Business-to-Business (B2B) e-commerce. Existing enterprises would form alliances and integrate their services to share costs, skills, and resources in offering a value-added service to form what is known as *B2B services*. Briefly stated, a B2B service is a conglomeration of mostly outsourced services working in tandem to achieve the business goals of the desired enterprise. An example of an integrated B2B service is a financial management system that uses payroll, tax preparation, and cash management as components. The component services might all be outsourced to business partners.

To date, the development of B2B services has been largely ad-hoc, time-consuming, and requiring enormous effort of low-level programming (Benatallah, Medjahed, Bouguettaya, Elmagarmid & Beard 2000, Benatallah, Dumas, Sheng & Ngu 2002). This task would obviously be tedious and hardly scalable because of the volatility and size of the Web. As services are most likely autonomous and heterogeneous, building a B2B service with appropriate inter-service coordination would be difficult. More importantly, the fast and dynamic composition of services is an essential requirement for organisations to adapt their business practices to the dynamic nature of the Web.

As pointed out before, Internet and Web technologies have opened new ways of doing business more cheaply and efficiently. However, for B2B e-commerce to really take off, there is a need for effective and efficient means to abstract, compose, analyze, and evolve Web services in an appropriate time-frame. Ad-hoc and proprietary solutions on the one hand, and lack of a canonical model for modeling and managing Web services on the other hand, have largely hampered a faster pace in deploying B2B services. Current technologies based on Universal Description, Discovery, and Integration (UDDI)[1], Web Service Description Language (WSDL)[2], and Simple Object Access Protocol (SOAP)[3] do not realize complex Web service combinations, hence providing limited support in service composition. SOAP is a standard for exchanging XML-formatted messages over HTTP between applications. WSDL is a general purpose XML language for describing what a Web service does, where it resides, and how to invoke it. UDDI is a standard for publishing information about Web services in a global registry as well as for Web service discovery.

In this paper, we propose a Petri net-based algebra for modeling Web services control flows. The model is expressive enough to capture the semantics of complex service combinations and their respective specificities. The obtained framework enables declarative composition of Web services. We show that the defined algebra caters for the creation of dynamic and transient relationships among services.

The remainder of this paper is organized as follows. Web service modeling and specification using Petri nets are presented in Section 2. Section 3 is devoted to the algebra for composing Web services and its Petri net-based formal semantics. Section 4 discusses the analysis and verification of Web services. Section 5 gives a brief overview of related work. Finally, Section 6 provides some concluding remarks.

[1]http://www.uddi.org/.
[2]http://www.w3.org/TR/wsdl/.
[3]http://www.w3.org/TR/soap12-part1/.

2 Web Services as Petri Nets

Petri nets (Petri 1962, Peterson 1981) are a well-founded process modeling technique that have formal semantics. They have been used to model and analyze several types of processes including protocols, manufacturing systems, and business processes (Aalst 1999). A Petri net is a directed, connected, and bipartite graph in which each node is either a *place* or a *transition*. Tokens occupy places. When there is at least one token in every place connected to a transition, we say that the transition is *enabled*. Any enabled transition may *fire* removing one token from every input place, and depositing one token in each output place. For more elaborate introduction to Petri nets, the reader is referred to (Murata 1989, Reisig 1985, Peterson 1981).

The use of visual modeling techniques such as Petri nets in the design of complex Web services is justified by many reasons. For example, visual representations provide a high-level yet precise language which allows to express and reason about concepts at their natural level of abstraction.

A Web service behavior is basically a partially ordered set of operations. Therefore, it is straightforward to map it into a Petri net. Operations are modeled by transitions and the state of the service is modeled by places. The arrows between places and transitions are used to specify causal relations.

We can categorise Web services into material services (e.g., delivery of physical products), information services (create, process, manage, and provide information), and material/information services, the mixture of both.

We assume that a Petri net, which represents the behavior of a service, contains one input place (i.e., a place with no incoming arcs) and one output place (i.e., a place with no outgoing arcs). A Petri net with one input place, for absorbing information, and one output place, for emitting information, will facilitate the definition of the composition operators (see Section 3.2) and the analysis as well as the verification of certain properties (e.g, reachability, deadlock, and liveness). At any given time, a Web service can be in one of the following states: `NotInstantiated`, `Ready`, `Running`, `Suspended`, or `Completed` (similar to the ones defined in (Schuster, Georgakopoulos, Cichocki & Baker 2000)). When a Web service is in the `Ready` state, this means that a token is in its corresponding input place, whereas the `Completed` state means that there is a token in the corresponding output place.

Definition 2.1 (Service Net)
A service net is a labeled Place/Transition net, i.e., a tuple $SN = (P, T, W, i, o, \ell)$ where:

- *P is a finite set of places,*
- *T is a finite set of transitions representing the operations of the service,*
- *$W \subseteq (P \times T) \cup (T \times P)$ is a set of directed arcs (flow relation),*
- *i is the input place with ${}^\bullet i = \{x \in P \cup T \mid (x, i) \in W\} = \varnothing$,*
- *o is the output place with $o^\bullet = \{x \in P \cup T \mid (o, x) \in W\} = \varnothing$, and*
- *$\ell : T \rightarrow \mathcal{A} \cup \{\tau\}$ is a labeling function where $\mathcal{A}$ is a set of operation names. We assume that $\tau \notin \mathcal{A}$ and denotes a silent operation.* □

The flow relation W can also be interpreted as a function $W : (P \times T) \cup (T \times P) \rightarrow \{0, 1\}$. We will use both interpretations throughout this paper. Silent operations are transition firings that cannot be observed. They are used to distinguish between external and internal behavior of the service. Note that the service net, defined above, is an *ordinary* Petri net, that is, there is at most one directed arc linking a place to a transition or a transition to a place.

We give now a formal definition of a Web service.

Definition 2.2 (Web Service)
A Web service is a tuple $S = (NameS, Desc, Loc, URL, CS, SN)$ where:

- *$NameS$ is the name of the service, used as its unique identifier,*
- *$Desc$ is the description of the service provided. It summarizes what the service offers,*
- *Loc is the server the service is located in,*
- *URL is the invocation of the Web service,*
- *CS is a set of its component services. If $CS = \{NameS\}$ then S is a basic service. Otherwise S is a composite service, and*
- *$SN = (P, T, W, i, o, \ell)$ is the service net modeling the dynamic behavior of the service.* □

The place i is considered as the initial marking of the service S (i.e., only i contains a token). The execution of S starts when a token is in the place i and terminates when a token reaches the place o.

3 Composing Web Services

A Web service has a specific task to perform and may depend on other Web services, hence being composite. For example, a company that is interested in selling books could focus on this aspect while outsourcing other aspects such as payment and shipment. The composition of two or more services generates a new service providing both the original individual behavioral logic and a new collaborative behavior for carrying out a new composite task. This means that existing services are able to cooperate although the cooperation was not designed in advance. Service composition could be static (service components interact with each other in a pre-negotiated manner) or dynamic (they discover each other and negotiate on the fly).

In this section we present an algebra that allows the creation of new value-added Web services using existing ones as building blocks. Sequence, alternative, iteration, and arbitrary sequence are typical constructs specified in the control flow. More elaborate operators, dealt with in this paper, are parallel with communication, discriminator, selection, and refinement. We also give a formal semantics to the proposed algebra in terms of Petri nets as well as some nice algebraic properties.

3.1 Web Service Algebra

We describe below the syntax and informal semantics of the service algebra operators. The constructs were chosen to allow common and advanced Web service combinations. The set of services can be defined by the following grammar in BNF-like notation:

$$\begin{aligned} S \ ::= \ & \varepsilon \mid X \mid S \odot S \mid S \oplus S \mid S \Diamond S \mid \mu S \mid \\ & S \parallel_C S \mid (S|S) \rightsquigarrow S \mid \\ & [S(p, q) : S(p, q)] \mid Ref(S, a, S) \end{aligned}$$

where:

- ε represents an *empty* service, i.e, a service which performs no operation.
- X represents a service *constant*, used as an atomic or basic service in this context.

- $S_1 \odot S_2$ represents a composite service that performs the service S_1 followed by the service S_2, i.e., $\odot$ is an operator of *sequence*.
- $S_1 \oplus S_2$ represents a composite service that behaves as either service S_1 or service S_2. Once one of them executes its first operation the second service is discarded, i.e., $\oplus$ is an *alternative* (or a *choice*) operator.
- $S_1 \Diamond S_2$ represents a composite service that performs either the service S_1 followed by the service S_2, or S_2 followed by S_1, i.e., $\Diamond$ is an *unordered sequence* (or an *arbitrary sequence*) operator.
- μS represents a service that performs a certain number of times the service S, i.e., μ represents an *iteration* operator.
- $S_1 \parallel_{\mathcal{C}} S_2$ represents a composite service that performs the services S_1 and S_2 independently from each other with possibilities of communication over the set $\mathcal{C}$ of pairs of operations, that is, $\parallel_{\mathcal{C}}$ is a *parallel* operator with *communication*.
- $(S_1|S_2) \rightsquigarrow S_3$ represents a composite service that waits for the execution of one service (among the services S_1 and S_2) before activating the subsequent service S_3, i.e., $\rightsquigarrow$ is a *discriminator* operator. Note that S_1 and S_2 are performed in parallel and without communication.
- $[S_1(p_1, q_1) : S_n(p_n, q_n)]$ is a composite service that dynamically selects one service provider among n available services $S_1, \ldots, S_n$ and executes it. It behaves as follows: first a request is sent by a composer to n available service providers of a given trading community through their entry access points $p_1, \ldots, p_n$. Then based on the received responses, from their exit access points $q_1, \ldots, q_n$, and according to given ranking criteria (e.g. price, delivery date/time, or a combination of both) the best service provider is chosen. Finally the needed operations are performed. [:] is an operator of *selection*.
- $Ref(S_1, a, S_2)$ represents a composite service that behaves as S_1 except for operations in S_1 with label a that are replaced by the non empty service S_2. Ref is a *refinement* operator.

The proposed algebra verifies the closure property. It guarantees that each result of an operation on services is a service to which we can again apply algebra operators. We are thus able to build more complex services by aggregating and reusing existing services through declarative expressions of service algebra.

3.2 Formal Semantics

In this section, we give a formal definition, in terms of Petri nets, of the composition operators. Let $S_i = (NameS_i, Desc_i, Loc_i, URL_i, CS_i, SN_i)$ with $SN_i = (P_i, T_i, W_i, i_i, o_i, \ell_i)$ for $i = 1, .., n$ be n Web services such that $P_i \cap P_j = \varnothing$ and $T_i \cap T_j = \varnothing$ for $i \neq j$.

It is important to note that service composition, as will be described below, applies to syntactically different services. This is due to the fact that the places and transitions of the component services must be disjoint for proper composition. However, a service may be composed with itself. Typically, this situation occurs when services describe variants of the same operation (e.g., normal execution and exceptional situations) or, for instance, if a single supplier offers two different goods, the requests may be handled independently, as though they were from two different suppliers. In this case, the overlapping must be resolved prior to composition. This can be accomplished by renaming the sets P and T of one of the equal services. The two services remain equal up to isomorphism on the names of transitions and places. Note also that, in case of silent operations, we represent graphically the corresponding transitions as black rectangles.

3.2.1 Basic Constructs

Empty Service. The empty service ε is a service that performs no operation. It is used for technical and theoretical reasons.

Definition 3.1 *The empty service ε is defined as $\varepsilon = (NameS, Desc, Loc, URL, CS, SN)$ where:*

– $NameS =$ `Empty`,
– $Desc =$ *"Empty Web Service",*
– $Loc =$ `Null`*, stating that there is no server for the service,*
– $URL =$ `Null`*, stating that there is no URL for ε,*
– $CS = \{$`Empty`$\}$*, and*
– $SN = (\{p\}, \varnothing, \varnothing, p, p, \varnothing)$. □

Graphically, ε is represented by the Petri net of Figure 1 containing only one place.

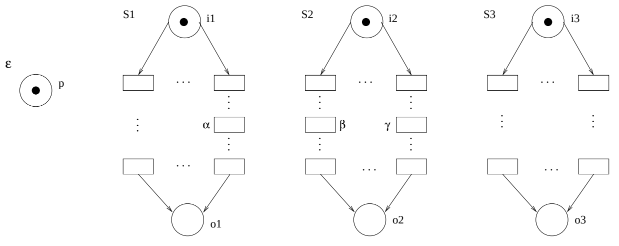

Figure 1: Services ε, S_1, S_2, and S_3

Sequence. The sequence operator allows the execution of two services S_1 and S_2 in sequence, that is, one after another. S_1 must be completed before S_2 can start. This is typically the case when a service depends on the output of the previous service. For example, the service `Payment` is executed after the completion of the service `Delivery`.

Definition 3.2 *The service $S_1 \odot S_2$ is defined as $S_1 \odot S_2 = (NameS, Desc, Loc, URL, CS, SN)$ where:*

– *$NameS$ is the name of the new service,*
– *$Desc$ is the description of the new service,*
– *Loc is the location of the new service (may be at the same server as one of the two component services),*
– *URL is the invocation of the new service,*
– $CS = CS_1 \cup CS_2$,
– $SN = (P, T, W, i, o, \ell)$ *where:*

  - $P = P_1 \cup P_2$,
  - $T = T_1 \cup T_2 \ \cup \ \{t\}$,
  - $W = W_1 \cup W_2 \ \cup \ \{(o_1, t), (t, i_2)\}$,
  - $i = i_1$,
  - $o = o_2$*, and*
  - $\ell = \ell_1 \cup \ell_2 \ \cup \ \{t, \tau\}$. □

Graphically, given S_1 and S_2 (see Figure 1), $S_1 \odot S_2$ is represented by the Petri net shown in Figure 2.

Alternative. The alternative operator permits, given two services S_1 and S_2, to model the execution of either S_1 or S_2, but not both. For instance, the `assess_claim` service is followed by either the service `indemnify_customer` or the service `convoke_customer`.

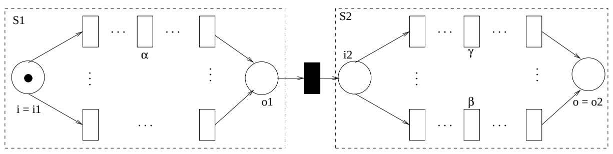

Figure 2: Service $S_1 \odot S_2$

Definition 3.3 *The service $S_1 \oplus S_2$ is defined as $S_1 \oplus S_2 = (NameS, Desc, Loc, URL, CS, SN)$ where:*

– *$NameS$ is the name of the new service,*
– *$Desc$ is the description of the new service,*
– *Loc is the location of the new service,*
– *URL is the invocation of the new service,*
– *$CS = CS_1 \cup CS_2$,*
– *$SN = (P, T, W, i, o, \ell)$ where:*

- $P = P_1 \cup P_2 \ \cup \ \{i, o\}$,
- $T = T_1 \cup T_2 \ \cup \ \{t_{i1}, t_{i2}, t_{o1}, t_{o2}\}$,
- $W = W_1 \ \cup \ W_2 \ \cup \ \{(i, t_{i1}), (i, t_{i2}), (t_{i1}, i_1), (t_{i2}, i_2), (o_1, t_{o1}), (o_2, t_{o2}), (t_{o1}, o), (t_{o2}, o)\}$, *and*
- $\ell = \ell_1 \cup \ell_2 \ \cup \ \{(t_{i1}, \tau), (t_{i1}, \tau), (t_{o1}, \tau), (t_{o2}, \tau)\}$. □

Graphically, given S_1 and S_2 (see Figure 1), $S_1 \oplus S_2$ is represented by the Petri net shown in Figure 3(a).

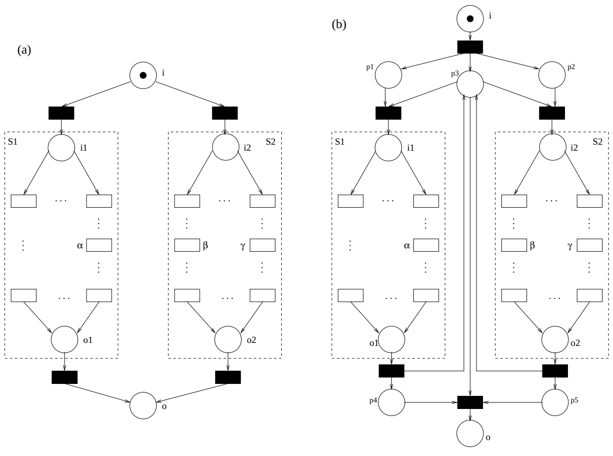

Figure 3: Services (a) $S_1 \oplus S_2$ and (b) $S_1 \diamond S_2$

Arbitrary Sequence. The arbitrary sequence operator specifies the execution of two services that must not be executed concurrently, that is, given two services S_1 and S_2, we have either S_1 followed by S_2 or S_2 followed by S_1. Suppose, for instance, that there are two goods, then acquiring a single good is useless unless the rest of the conjuncts can also be acquired. Moreover, without a deadline, there is no benefit by making the two requests in parallel, and doing so may lead to unnecessary costs if one of the conjuncts is unavailable or unobtainable. Therefore, the optimal execution is necessarily an arbitrary serial ordering of requests to suppliers.

Definition 3.4 *The service $S_1 \diamond S_2$ is defined as $S_1 \diamond S_2 = (NameS, Desc, Loc, URL, CS, SN)$ where:*

– *$NameS$ is the name of the composite service,*
– *$Desc$ is the description of the composite service,*
– *Loc is the server where the composite service is located,*
– *URL is the invocation of the composite service,*
– *$CS = CS_1 \cup CS_2$,*
– *$SN = (P, T, W, i, o, \ell)$ where:*

- $P = P_1 \cup P_2 \cup \ \{i, o, p_1, p_2, p_3, p_4, p_5\}$,
- $T = T_1 \cup T_2 \cup \ \{t_i, t_1, t_2, t_3, t_4, t_o\}$,
- $W \ = \ W_1 \cup W_2 \cup \ \{(i, t_i), (t_i, p_1), (t_i, p_2), (t_i, p_3), (p_1, t_1), (p_2, t_2), (p_3, t_1), (p_3, t_2), (p_3, t_o), (t_1, i_1), (t_2, i_2), (o_1, t_3), (o_2, t_4), (t_3, p_3), (t_4, p_3), (t_3, p_4), (t_4, p_5), (p_4, t_o), (p_5, t_o), (t_o, o)\}$, *and*
- $\ell = \ell_1 \cup \ell_2 \ \cup \ \{(t_i, \tau), (t_1, \tau), (t_2, \tau), (t_3, \tau), (t_4, \tau), (t_o, \tau)\}$. □

Graphically, given S_1 and S_2 (see Figure 1), $S_1 \diamond S_2$ is represented by the Petri net shown in Figure 3(b).

Iteration. The iteration operator models the execution of a service followed a certain number of times by itself. Typical examples where iteration is required are communication and quality control where services are executed more than once. Another example is when ordering several processors, the service `Order Processor` is executed several times.

Definition 3.5 *The service μS_1 is defined as $\mu S_1 = (NameS, Desc, Loc, URL, CS, SN)$ where:*

– *$NameS$ is the name of the new service,*
– *$Desc$ is the description of the new service,*
– *Loc is the location of the new service,*
– *URL is the invocation of the new service,*
– *$CS = CS_1$,*
– *$SN = (P, T, W, i, o, \ell)$ where:*

- $P = P_1 \ \cup \ \{i, o\}$,
- $T = T_1 \ \cup \ \{t_i, t_o, t\}$,
- $W \ = \ W_1 \ \cup \{(i, t_i), (t_i, i_1), (o_1, t_o), (t_o, o), (o_1, t), (t, i_1)\}$, *and*
- $\ell = \ell_1 \ \cup \{(t_i, \tau), (t_o, \tau), (t, \tau)\}$. □

Graphically, given S_1 (see Figure 1), μS_1 is represented by the Petri net shown in Figure 4(a).

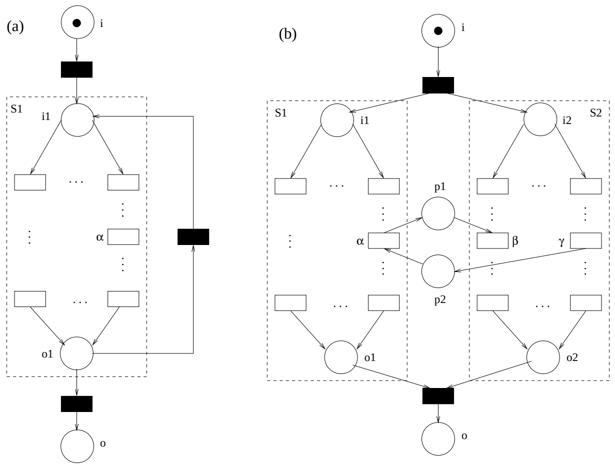

Figure 4: Services (a) μS_1 and (b) $S_1 \parallel_{\mathcal{C}} S_2$

3.2.2 Advanced Constructs

Parallelism with Communication. The parallel operator represents the concurrent execution of two services. Concurrent services may synchronize and exchange information. For instance, in an `Online Computer Store` Web service, after receiving an order for a computer from a customer, two parallel services are triggered: `Order Monitor` and `Order Processor`.

Definition 3.6 *Let $\mathcal{C} = \{(\alpha,\beta) \mid (\alpha,\beta) \in T_1 \times T_2 \cup T_2 \times T_1\}$ be a set of communication elements. The service $S_1 \parallel_{\mathcal{C}} S_2$ is defined as $S_1 \parallel_{\mathcal{C}} S_2 = (NameS, Desc, Loc, URL, CS, SN)$ where:*

- *$NameS$ is the name of the composite service,*
- *$Desc$ is the description of the composite service,*
- *Loc is the location of the composite service,*
- *URL is the invocation of the composite service,*
- *$CS = CS_1 \cup CS_2$,*
- *$SN = (P, T, W, i, o, \ell)$ where:*
  - $P = P_1 \cup P_2 \cup \{i, o\} \cup \{p_i \; / \; (\alpha_i, \beta_i) \in \mathcal{C}\}$,
  - $T = T_1 \cup T_2 \cup \{t_i, t_o\}$,
  - $W = W_1 \; \cup \; W_2 \; \cup \; \{(i,t_i),(t_i,i_1),(t_i,i_2),(o_1,t_o),(o_2,t_o),(t_o,o)\} \; \cup \; \{(\alpha_i,p_i),(p_i,\beta_i) \; / \; (\alpha_i,\beta_i) \in \mathcal{C}\}$, *and*
  - $\ell = \ell_1 \cup \ell_2 \; \cup \; \{(t_i,\tau),(t_o,\tau)\}$. □

Given S_1, S_2 (see Figure 1), and $\mathcal{C} = \{(\alpha,\beta),(\gamma,\alpha)\}$, $S_1 \parallel_{\mathcal{C}} S_2$ is represented graphically by the Petri net shown in Figure 4(b).

Discriminator. Web services are unreliable; they have a relatively high probability of failing or of being unacceptably slow. Delays of only a few seconds could result in service providers losing significant sums of money or disappointing their customers. Different service providers may provide the same or similar services. Therefore, it should be possible to combine unreliable services to obtain more "reliable" services. The discriminator operator is used, for instance, to place redundant orders to different suppliers offering the same service to increase reliability. The first to perform the requested service triggers the subsequent service and all other late responses are ignored for the rest of the composite service process. This construct is similar to the discriminator pattern introduced in (Aalst, Hofstede, Kiepuszewski & Barros 2002) but our implementation, in terms of Petri nets, is different.

Definition 3.7 *The service $(S_1|S_2) \rightsquigarrow S_3$ is defined as $(S_1|S_2) \rightsquigarrow S_3 = (NameS, Desc, Loc, URL, CS, SN)$ where:*

- *$NameS$ is the name of the new service,*
- *$Desc$ is the description of the new service,*
- *Loc is the location of the new service,*
- *URL is the invocation of the new service,*
- *$CS = CS_1 \cup CS_2 \cup CS_3$,*
- *$SN = (P, T, W, i, o, \ell)$ where:*
  - $P = P_1 \cup P_2 \cup P_3 \; \cup \; \{i, o, p_1, p_2\}$,
  - $T = T_1 \cup T_2 \cup T_3 \; \cup \; \{t_i, t_1, t_2, t_3, t_o\}$,
  - $W = W_1 \cup W_2 \cup W_3 \; \cup \; \{(i,t_i),(t_i,i_1),(t_i,i_2),(t_i,p_2),(o_1,t_1),(o_2,t_2),(t_1,p_1),(t_2,p_1),(p_1,t_3),(p_1,t_o),(p_2,t_3),(t_3,i_3),(o_3,t_o),(t_o,o)\}$, *and*
  - $\ell = \ell_1 \cup \ell_2 \cup \ell_3 \cup \; \{(t_i,\tau),(t_1,\tau),(t_2,\tau),(t_3,\tau),(t_o,\tau)\}$. □

Graphically, given S_1, S_2, and S_3 (see Figure 1), $(S_1|S_2) \rightsquigarrow S_3$ is represented by the Petri net shown in Figure 5(a). Note that $|$, in $(S|S) \rightsquigarrow S$, is similar to a parallel operator without communication, i.e., $\parallel_{\varnothing}$.

Selection. Relying on a single supplier puts a company at its mercy. To reduce risk, a company should maintain relationships with multiple suppliers. These suppliers may, e.g., charge different prices, propose different delivery dates and times, and have different reliabilities. The selection construct allows to choose the best service provider, by using a ranking criteria, among several competing suppliers to outsource a particular operation.

Definition 3.8 *Let $p_i, q_i \in P_i$ be service access points of S_i for $i = 1,..,n$. The service $[S_1(p_1,q_1) : S_n(p_n,q_n)]$ is defined as $[S_1(p_1,q_1) : S_n(p_n,q_n)] = (NameS, Desc, Loc, URL, CS, SN)$ where:*

- *$NameS$ is the name of the composite service,*
- *$Desc$ is the description of the composite service,*
- *Loc is the location of the composite service,*
- *URL is the invocation of the composite service,*
- *$CS = \bigcup_{i=1}^{n} CS_i$,*
- *$SN = (P, T, W, i, o, \ell)$ where:*
  - $P = \bigcup_{i=1}^{n} P_i \; \cup \; \{i, o, p, q\}$,
  - $T = \bigcup_{i=1}^{n} T_i \; \cup \; \{t, u, v\} \; \cup \; \{t_i, t'_i \mid 1 \le i \le n\}$,
  - $W = \bigcup_{i=1}^{n} W_i \; \cup \; \{(i,t),(u,p),(q,v),(v,o)\} \; \cup \; \{(t,p_j),(q_j,u),(p,t'_j),(t'_j,i_j),(o_j,t_j),(t_j,q) \mid 1 \le j \le n\}$, *and*
  - $\ell = \bigcup_{i=1}^{n} \ell_i \; \cup \; \{(t,\texttt{send\_req\_serv}),(u,\texttt{select\_serv}),(v,\tau)\} \; \cup \; \{(t_i,\tau),(t'_i,\tau) \mid 1 \le i \le n\}$. □

Graphically, given $S_1, \ldots, S_n$, and $p_i, q_i \in P_i$ for $i = 1,..,n$, $[S_1(p_1,q_1) : S_n(p_n,q_n)]$ is represented by the Petri net shown in Figure 5(b). To allow the discovery of the service, we assume that service providers register their services with a particular trading community (Benatallah et al. 2000, Benatallah et al. 2002) and that information about available services is known. We also assume that each service provider contains two distinct parts, one part to process the service request and the other part to perform the service itself. The pair of operations `send_req_serv` and `select_serv` represents a predefined selection strategy chosen, among others, by the designer of the composite service.

The selection decision may also be based on automated negotiation and auctions (e.g, sealed bid and open-cry auctions) where the parties involved, that is, the composer (auctioneer) and the service providers, are automated. By doing so, automated online auctions become a fundamental building block for selection decision.

Refinement. The refinement construct, in which operations are replaced by more detailed non empty services, is used to introduce additional component services into a service. Refinement is the transformation of a design from a high level abstract form to a lower level more concrete form hence allowing hierarchical modeling.

Definition 3.9 *Let $a \in \mathcal{A}$. The service $Ref(S_1, a, S_2)$ is defined as $Ref(S_1, a, S_2) = (NameS, Desc, Loc, URL, CS, SN)$ where:*

- *$NameS$ is the name of the refined service,*
- *$NameS$ is the name of the refined service,*
- *$Desc$ is the description of the refined service,*
- *Loc is the location of the refined service (may be at the same location as S_1),*
- *URL is the invocation of the refined service,*
- *$CS = \begin{cases} CS_1 \cup CS_2 & if \; a \in \ell_1(T_1) \\ CS_1 & otherwise \end{cases}$,*
- *$SN = (P, T, W, i, o, \ell)$ where:*
  - $P = P_1 \; \cup \; \{< p, t > \; \mid p \in P_2 \setminus \{i_2, o_2\}, t \in \ell_1^{-1}(a)\}$,

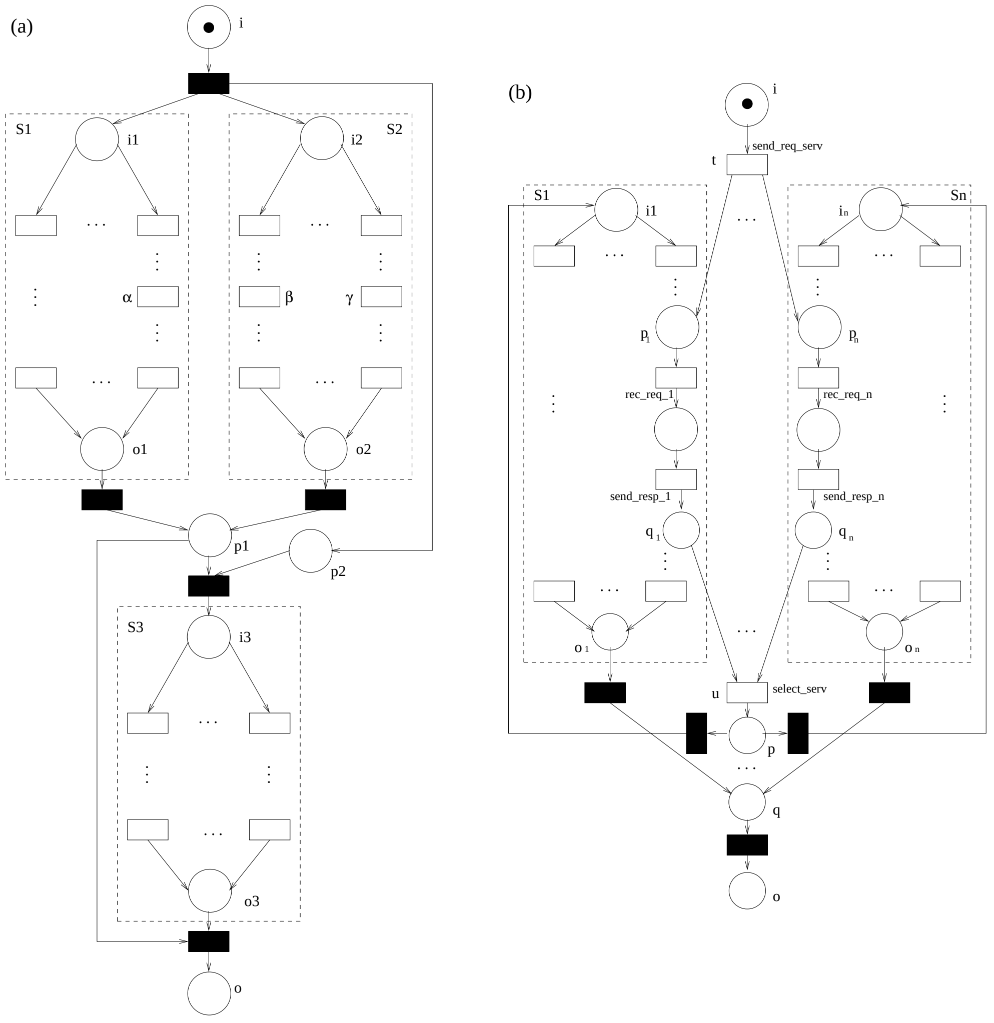

Figure 5: Services (a) $(S_1|S_2) \rightsquigarrow S_3$ and (b) $[S_1(p_1, q_1) : S_n(p_n, q_n)]$

- $T = T_1 \setminus \ell_1^{-1}(a) \ \cup \ \{< x, t > \ | \ x \in T_2, t \in \ell_1^{-1}(a)\}$,
- $W(x,y) = \begin{cases} W_1(x,y) & if\ x, y \in P_1 \cup T_1 \setminus \ell_1^{-1}(a) \\ W_2(x',y') & if\ x =< x', t >, \ y =< y', t > \ for\ t \in \ell_1^{-1}(a) \\ W_1(x,t) & if\ y =< y', t >, \ x \in {}^\bullet t\ for\ t \in \ell_1^{-1}(a)\ and\ y' \in i_2^\bullet \\ W_1(t,y) & if\ x =< x', t >, \ y \in t^\bullet\ for\ t \in \ell_1^{-1}(a)\ and\ x' \in {}^\bullet o_2 \\ 0 & otherwise \end{cases}$
- $i = i_1$,
- $o = o_1$, *and*
- $\ell(t) = \begin{cases} \ell_1(t) & if\ t \in T_1 \setminus \ell_1^{-1}(a) \\ \ell_2(x) & if\ t =< x, t' > for\ x \in T_2\ and\ t' \in \ell_1^{-1}(a) \end{cases}$

□

Figure 6 shows an example of a refined service where S_2 (performing either `assess_simple_claim` or `assess_complex_claim`) is a refinement (i.e., *specialization*) of the *generic* operation `assess_claim` in S_1.

Another interesting use of the refinement operator is its combination with the selection operator for best outsourcing operations. In

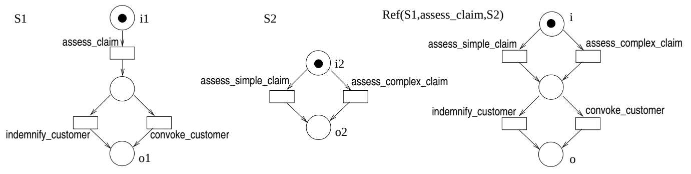

Figure 6: An Example of Refinement

$Ref(S_1, \texttt{assess\_claim}, [A_1 : A_n])$, for instance, `assess_claim` is performed by the suitable service chosen among n available assessors.

The composite Web service generated by using the above constructs is a service net that has two special places, which correspond to the beginning and termination of the processing of the composite Web service. This interesting property relates to the structure of the underlying Petri net.

Proposition 3.1 (Preservation) *The service net of a composite Web service S obtained using the above defined constructs contains one input place i and one output place o.*

Proof. Immediate consequence of Definition 2.2 and of the definitions of the composition constructs defined previously. □

In what follows, we give an example of a composite service modeled as a Petri net to illustrate some of the constructs defined above.

Example 3.1
Figure 7 shows a Web service composed of three basic services, OCS representing an `Online Computer Store` and SM and IP, representing respectively the `Sony Monitors` and the `Intel Processors`. Upon reception of an order `rec_ord_PC` for a computer from a customer, OCS starts, in parallel, the outsourced services SM to order a monitor and IP to order a processor by performing the operations `send_ord_mon` and `send_ord_pr` respectively. The set of communication elements are $\mathcal{C}_1 = \{$(`send_ord_mon`, `rec_ord_mon`), (`send_del_mon`, `rec_del_mon`)$\}$ and $\mathcal{C}_2 = \{$(`send_ord_pr`, `rec_ord_pr`), (`send_del_pr`, `rec_del_pr`)$\}$. Once the requested items are received, OCS performs the `assemble_PC` operation.

Note that, for the sake of simplicity and clarity, not all the operations of the scenario (e.g, delivery and billing) are represented and labels are used instead of names for the transitions.

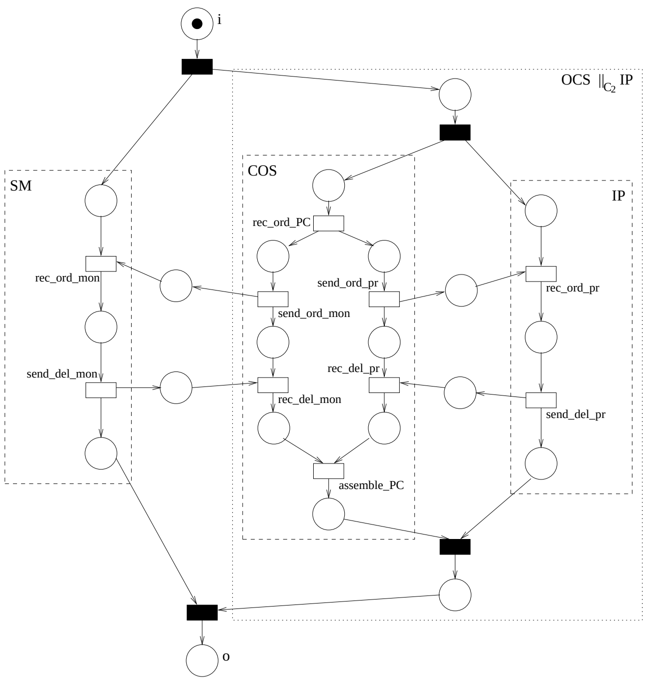

Figure 7: Service $SM \|_{\mathcal{C}_1} (OCS \|_{\mathcal{C}_2} IP)$

Let us assume now that, instead of a basic service `Sony Monitors`, we use a composite service `Monitors`. The monitor provider can be dynamically selected among the available services (e.g., $M_1, \ldots, M_n$) which are members of `Monitors` (see Figure 8). Note that $\mathcal{C}_3 = \{$(`send_ord_mon`, `send_req_serv`), (τ, `rec_del_mon`)$\}$. □

3.3 Algebraic Properties

Our semantics can be used to prove algebraic properties of the constructs. As a general observation, it may be the case that a designer produces a complex Web service by combining a set of existing Web services using the algebra operators. The algebraic properties could then be used to transform and optimize composed Web services based on component Web services' operational metrics such as cost and duration, although not investigated in this paper.

A description of the algebraic properties of the operators introduced above is given in Table 1 (where $\prod_{i=1}^{n} S_i$ stands for $[S_1(p_1, q_1) : S_n(p_n, q_n)]$). The algebraic operators satisfy some usual properties, such as commutativity, associativity, and reflexivity.

$$S_1 \odot (S_2 \odot S_3) \ = \ (S_1 \odot S_2) \odot S_3 \tag{1}$$
$$\varepsilon \odot S \ = \ S \tag{2}$$
$$S \odot \varepsilon \ = \ S \tag{3}$$
$$S_1 \oplus S_2 \ = \ S_2 \oplus S_1 \tag{4}$$
$$S_1 \oplus (S_2 \oplus S_3) \ = \ (S_1 \oplus S_2) \oplus S_3 \tag{5}$$
$$S \oplus S = S \tag{6}$$
$$(S_1 \oplus S_2) \odot S_3 \ = \ (S_1 \odot S_3) \oplus (S_2 \odot S_3) \tag{7}$$
$$S_1 \Diamond S_2 \ = \ (S_1 \odot S_2) \oplus (S_2 \odot S_1) \tag{8}$$
$$\mu\varepsilon = \varepsilon \tag{9}$$
$$S_1 \|_{\mathcal{C}} S_2 = S_2 \|_{\mathcal{C}} S_1 \tag{10}$$
$$S_1 \|_{\varnothing} (S_2 \|_{\varnothing} S_3) \ = \ (S_1 \|_{\varnothing} S_2) \|_{\varnothing} S_3 \tag{11}$$
$$S \|_{\varnothing} \varepsilon \ = \ S \tag{12}$$
$$(S_1|S_2) \rightsquigarrow S_3 \ = \ (S_2|S_1) \rightsquigarrow S_3 \tag{13}$$
$$(S_1|\varepsilon) \rightsquigarrow S_2 \ = \ S_1 \|_{\varnothing} S_2 \tag{14}$$
$$(S_1|S_2) \rightsquigarrow \varepsilon \ = \ S_1 \|_{\varnothing} S_2 \tag{15}$$
$$\forall \{i_1, ..., i_n\} = \{1, ..., n\} \quad \prod_{i \in \{i_1, \ldots, i_n\}} S_i = \prod_{i=1}^{n} S_i \tag{16}$$
$$\text{If } S_j = \varepsilon \text{ then } \prod_{i=1}^{n} S_i = \prod_{i=1, i \neq j}^{n} S_i \tag{17}$$
$$Ref(S_1, a, S_2) = S_1 \text{ if } a \notin \ell_1(T_1) \tag{18}$$
$$S_1 \Diamond S_2 \ = \ S_2 \Diamond S_1 \text{ (from (8) and (4))} \tag{19}$$
$$S \Diamond S \ = \ S \odot S \text{ (from (8) and (6))} \tag{20}$$
$$S \Diamond \varepsilon \ = \ S \text{ (from (8), (2), (3), and (6))} \tag{21}$$

Table 1: Desired Properties of the Service Algebra

Simple properties can be easily derived, for example (2), (3), (9), and (12). A range of other equations can be derived, each by an easy case analysis.

An interesting observation is that our semantics equates the services $S_1 \Diamond S_2$ and $(S_1 \odot S_2) \oplus (S_2 \odot S_1)$ (property (8)). This means that arbitrary sequence can be expressed using sequence and alternative. However, it is still useful to include the arbitrary sequence operator in the algebra since the obvious implementation will be inefficient for the service $(S_1 \odot S_2) \oplus (S_2 \odot S_1)$.

4 Web Service Analysis

A composite Web service is a system that consists of several conceptually autonomous but cooperating units. It is difficult to specify how this system should behave and ensure that it behaves as required by the specification. The reason is that, given the same input and initial state, the system may produce several different outputs.

Web services interact with each other for conducting a specific task although they are created and deployed by independent service providers. The pieces of work within the specific task are tightly coupled. For example, an `Online Computer Store` may need services provided by `Monitor` and `Processor` Web services. The business logic of those two Web services are independent of each other but, for a particular computer request, the two pieces of work, that is, the monitor ordering and the processor ordering, are related. Since Web services which contain errors may lead to angry customers and loss of goodwill, it is thus important to analyze Web services before they are put into operation. The goal is to provide mechanisms to support correct Web service composition.

The properties to be verified may be general, such as absence of deadlocks and livelocks, or application-specific, such as if a customer keep reserving and canceling the reservation indefinitely, in an `Online Ticket Sales` Web service, a sophisticated system might deny serving such a customer after a certain

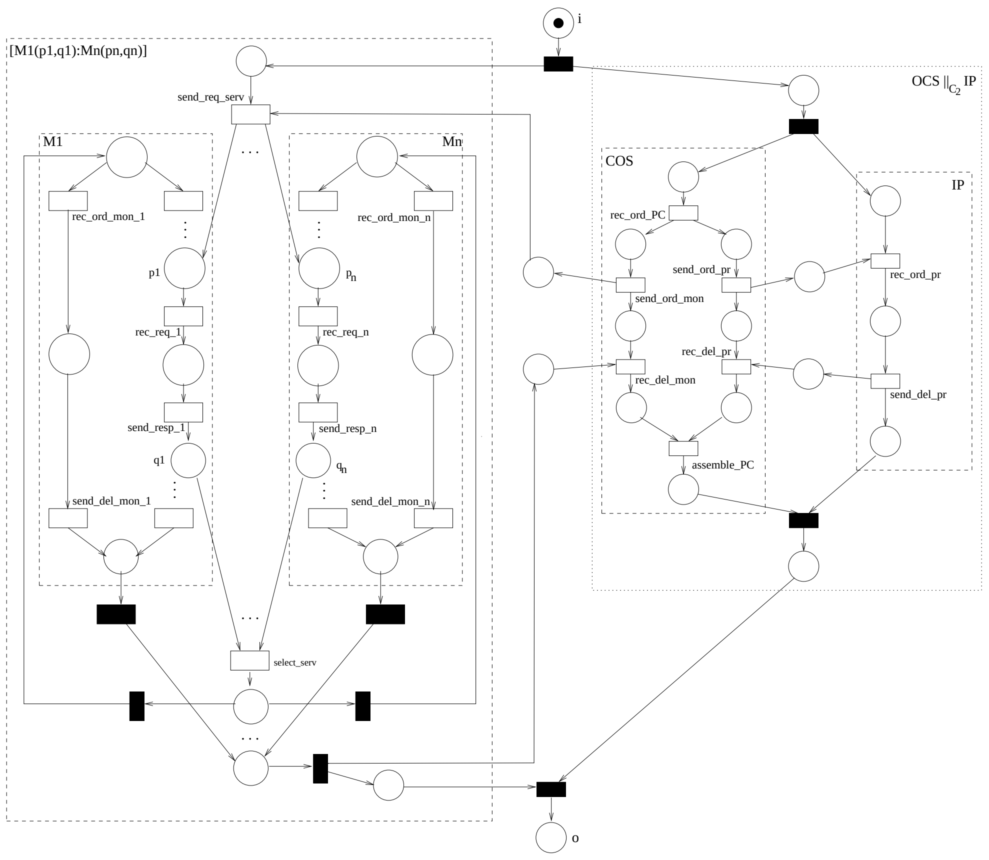

Figure 8: Service $[M_1(p_1, q_1) : M_n(p_n, q_n)] \parallel_{\mathcal{C}_3} (OCS \parallel_{\mathcal{C}_2} IP)$

number of reservation or cancellation requests. The correctness of services is vital to companies. A Web service which contains errors (e.g., a deadlock) may lead to angry customers and loss of goodwill. It is thus important to analyze a service before it is put into operation. The correct termination, is one of the main properties a proper Web service should satisfy. The verification of the correctness property consists in checking whether the underlying Petri net is live and bounded (Murata 1989).

Behavioral equivalences are useful in verification as they lay the conceptual basis for deciding that the behavior of two Web services can be considered to be "the same". They can also be used as a tool for reducing verification effort by replacing the Petri net of a service by a smaller (in size), but "equivalent" one. The *bisimulation* equivalence (Park 1981, Milner 1989) between two Web services is a relation between their evolutions such that for each evolution of one of the services there is a corresponding evolution of the other service such that the evolutions are observationally "equivalent" and lead to services which are again bisimilar. This characterization of the behavior of Web services using the notion of bisimulation helps service designer optimize composite services by, e.g., changing their component Web services with "equivalent" ones. Another motivation is *customization* of services. To enhance competitiveness a service provider may modify his/her service for a customer's convenience (e.g., to conform to his/her own business model) and this customized service must conform to the original one.

5 Related Work

Several approaches investigated the issue of sharing Web services. However, none of them offers a general theoretical framework for the composition of Web services. In this section, we briefly overview the approaches that are closely related to our work.

Workflow management systems (Georgakopoulos, Hornick & Sheth 1995, WfMC 1999) are based on the premise that the success of an enterprise requires the management of business processes in their entirety. Indeed, an increasing number of organizations have already automated their internal process management using workflow technology and enjoyed substantial benefits in doing so. However, B2B e-commerce requires the flexible support of cross-enterprises relationships. Current research efforts in the workflow area promise to deliver a next generation workflow systems that has the ability to easily thread together cross-organizational business processes, supporting the integration of diverse users, applications, and systems. The purpose of cross-organization workflows is to automate business processes that interconnect and manage communication among disparate systems. Early projects in this direction focus mostly on the integration of small numbers of tightly coupled business processes. Recent emerging workflow projects such as CMI (Georgakopoulos, Schuster, Cichocki & Baker 1999), eFlow (Casati, Ilnicki, Jin & Shan 2000), and CrossFlow (Ludwig & Hoffner 1999, Hoffner, Ludwig, Gulcu & Grefen 2000) focus on loosely coupled processes. However, they lack a formal model for specifying and verifying Web services. They also do not address the semantics of Web service composition operators.

Other approaches modeling business processes using Petri nets are in the workflow setting. Workflow nets, a class of Petri nets similar to our service net, have been introduced in (Aalst 1997, Aalst 1998) for the representation and verification of workflow processes. In (Adam, Alturi & Huang 1998), the authors developed a Petri net-based approach that uses several structural properties for identifying inconsistent dependency specification in a workflow, testing for its safe termination, and checking for the feasibility of its execution for a given starting time when temporal constraints are present. An implementation is provided for conducting the above analyses. However, the approach is restricted to acyclic workflows. Nevertheless, these previous approaches treat every workflow process as a separate entity. They do not consider business process composition.

Web service integration requires more complex functionality than SOAP, WSDL, and UDDI can provide. The functionality includes transactions, workflow, negotiation, management, and security. There are several efforts that aim at providing such functionality, for example, the recently released Business Process Execution Language for Web Services (BPEL4WS)[4], which represents the merging of IBM's Web Services Flow Language (WSFL) (Leymann 2001) and Microsof's XLANG (Thatte 2001), is positioned to become the basis of a standard for Web service composition. These languages are based on SOAP+WSDL+UDDI basic stack, are complex procedural languages, and very hard to implement and deploy. There are also some proposals such as DAML-S[5] which is a part of DARPA Agent Markup Language project that aims at realizing the Semantic Web concept. However, DAML-S is a complex procedural language for Web service composition.

6 Conclusions

In this paper, we proposed a Petri net-based algebra for composing Web services. The formal semantics of the composition operators is expressed in terms of Petri nets by providing a direct mapping from each operator to a Petri net construction. Thus, any service expressed using the algebra constructs can be translated into a Petri net representation. By means of a set of algebra properties, we are able to transform and optimize Web service expressions guaranteeing the same semantics of initial expressions. In addition, the use of a formal model allows the verification of properties and the detection of inconsistencies both within and between services.

There are other issues in B2B e-commerce which, we believe, could be successfully addressed by extending the framework presented in this paper, e.g., to include management of time and resources. We expect that these problems can be dealt with using a suitable high-level Petri net, such as timed and coloured Petri nets (Jensen 1997). Another issue is that a strong link with process algebras, such as CCS (Milner 1989) and ACP (Bergstra & Klop 1985), should allow one to import process algebra specific verification techniques (e.g., axiomatizations of behavioral equivalences) into the Petri net based framework. Finally, composing Web services that satisfy requirements such as transactional process, negotiation, exception handling, and security through externally visible interaction or conversation is a challenging issue.

Acknowledgments

The authors would like to thank Associate Professor Arthur ter Hofstede from Queensland University of Technology for his constructive criticisms on an earlier version of this paper.

References

Aalst, W. v. d. (1997), Verification of Workflow Nets, *in* P. Azema & G. Balbo, eds, 'Proceedings of the Application and Theory of Petri Nets'97', Toulouse, France.

Aalst, W. v. d. (1998), 'The Application of Petri Nets to Workflow Management', *The Journal of Circuits, Systems and Computers* **8**(1), 21–66.

Aalst, W. v. d. (1999), 'Interorganizational Workflows: An Approach based on Message Sequence Charts and Petri Nets', *Systems Analysis - Modelling - Simulation* **34**(3), 335–367.

Aalst, W. v. d., Hofstede, A. t., Kiepuszewski, B. & Barros, A. (2002), Workflow Patterns, Technical Report FIT-TR-2002-02, Queensland University of Technology, Brisbane, Australia.

Adam, N., Alturi, V. & Huang, W.-K. (1998), 'Modeling and Analysing of Workflows Using Petri Nets', *Journal of Intelligent Information Systems* **10**(2), 131–158.

Benatallah, B., Dumas, M., Sheng, Q. & Ngu, A. (2002), Declarative Composition and Peer-to-Peer Provisioning of Dynamic Web Services, *in* 'Proceedings of the 18th International Conference on Data Engineering (ICDE'02)', IEEE Computer Society, California, USA, pp. 297–308.

Benatallah, B., Medjahed, B., Bouguettaya, A., Elmagarmid, A. & Beard, J. (2000), Composing and Maintaining Web-based Virtual Enterprises, *in* 'Proceedings of the Workshop on Technologies for E-Services (in Cooperation with VLDB'00)', Cairo, Egypt.

Bergstra, J. & Klop, J. (1985), 'Algebra of Communicating Processes with Abstraction', *TCS* **37**, 77–121.

Casati, F., Ilnicki, S., Jin, L. & Shan, M.-C. (2000), An Open, Flexible, and Configurable System for E-Service Composition, Technical Report HPL-2000-41, HP Labs.

Georgakopoulos, D., Hornick, M. & Sheth, A. (1995), 'An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure', *Distributed and Parallel Databases* **3**(2).

Georgakopoulos, D., Schuster, H., Cichocki, A. & Baker, D. (1999), 'Managing Process and Service Fusion in Virtual Enterprises', *Information Systems, Special Issue on Information Systems Support for Electronic Commerce* **24**(6), 429–456.

Hoffner, Y., Ludwig, H., Gulcu, C. & Grefen, P. (2000), Architecture for Cross-Organizational Business Processes, Research report, IBM, Zurich, Switzerland.

Jensen, K. (1997), *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use. Vol. 1, Basic Concepts*, EATCS Monographs on Theoretical Computer Science, Springer-Verlag.

Leymann, F. (2001), *Web Services Flow Language (WSFL) version 1.0*, IBM Software Group. http://ibm.com/software/solutions/webservices /pdf/WSFL.pdf/.

Ludwig, H. & Hoffner, Y. (1999), Contract-based Cross-Organisational Workflows - The CrossFlow Project, *in* D. Georgakopoulos, W. Prinz & A. Wolf, eds, 'Proceedings of the International Joint Conference on Work Activities Coordination and Collaboration (WACC'99)', San Fransisco, USA.

[4]http://www.ibm.com/developerworks/library/ws-bpel/.

[5]http://www.daml.org/services/.

Milner, R. (1989), *Communication and Concurrency*, Prentice Hall Int.

Murata, T. (1989), Petri Nets: Properties, Analysis and Applications, *in* 'Proceedings of the IEEE', Vol. 77(4), pp. 541–580.

Park, D. (1981), Concurrency and Automata on Infinite Sequences, *in* P. Deussen, ed., 'Proceedings of the 5th GI Conference', LNCS 104, Springer Verlag.

Peterson, J. (1981), *Petri Net Theory and the Modeling of Systems*, Prentice Hall, Englewood Cliffs.

Petri, C. (1962), Kommunikation mit Automaten, PhD thesis, University of Bonn, Germany. (In German).

Reisig, W. (1985), *Petri Nets: An Introduction*, EATCS Monographs on Theoretical Computer Science Vol.4, Springer-Verlag, Berlin, Germany.

Schuster, H., Georgakopoulos, D., Cichocki, A. & Baker, D. (2000), Modeling and Composing Service-based and Reference Process-based Multi-enterprise Processes, *in* 'Proceedings of the 12th Conference on Advanced Information Systems Engineering (CAiSE'00)', Stockholm, Sweden.

Thatte, S. (2001), *XLANG: Web Services for Business Process Design*, Microsoft Corporation. http://www.gotdotnet.com/team/xml_wsspecs /xlang-c/default.htm/.

WfMC (1999), *Workflow Management Coalition, Terminology and Glossary*, Document Number WFMC-TC-1011. http://www.wfmc.org/standards/docs.htm/.