Modelling and Verifying Web Service Orchestration by means of the Concurrency Workbench*

Mariya Koshkina
IBM
8200 Warden Avenue
Markham, L6G 1C7, Canada
mkoshkin@ca.ibm.com

Franck van Breugel
York University
4700 Keele Street
Toronto, M3J 1P3, Canada
franck@cs.yorku.ca

ABSTRACT

Verification techniques like model checking, preorder checking and equivalence checking are shown to be relevant to web service orchestration. The Concurrency Workbench of the New Century (CWB) is a verification tool that supports these verification techniques. By means of the Process Algebra Compiler (PAC), the CWB is modified to support the BPE-calculus. The BPE-calculus is a small language, based on BPEL4WS, to express web service orchestration. Both the syntax and the semantics of the BPE-calculus are formally defined. These are subsequently used as input for the PAC. As output, the PAC produces modules that are incorporated into the CWB so that it supports the BPE-calculus and, hence, provides a verification tool for web service orchestration.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification—*formal methods, model checking*; D.3.1 [**Programming Languages**]: Formal Definitions and Theory; F.3.2 [**Logics and Meanings of Programs**]: Semantics of Programming Languages—*operational semantics*

General Terms

Languages, Theory, Verification

Keywords

Business process, web service, BPEL4WS, modelling, verification, Concurrency Workbench of the New Century, Process Algebra Compiler

1. INTRODUCTION

The *Concurrency Workbench of the New Century* (CWB) is a generic and customizable verification tool developed by Cleaveland et al. [14, 16]. The CWB supports *model checking*, *preorder checking* and *equivalence checking*. We will discuss these verification techniques below. Originally, the CWB was designed for the verification of Milner's Calculus of Communicating Systems (CCS) [35]. However, the CWB can be customized to support languages other than CCS. To support a new language, a considerable number of modules need to be implemented. This would be a time consuming task if not for the *Process Algebra Compiler* (PAC) [17, 44]. The PAC is a tool that generates these modules from a specification of the syntax and semantics of the language.

The CWB and the PAC have been successfully exploited in a number of sophisticated case studies. For example, in [6], Bhat, Cleaveland and Lüttgen model and verify several aspects of a widely used bus protocol. Cleaveland, Lüttgen, Natarajan and Sims present in [15] a model of a safety critical part of a system that controls railway signals and verify a number properties of the system. The modelling and verification of a fault tolerant active control system is addressed by Elseaidy, Cleaveland and Baugh in [18]. In [24], Kapoor and Josephs model delay insensitive circuits in CCS and verify some properties of these circuits using the CWB. Chen [9] and also Gradara, Santone, Villani and Vaglini [21] encode an abstraction of a multithreaded Java program in CCS and check properties of the encoding exploiting the CWB.

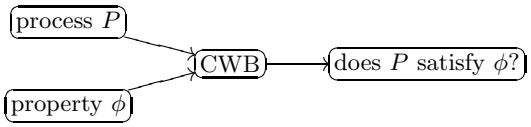
In this paper, we will show how the CWB and the PAC can be exploited to model and verify *web service orchestration*. To express web service orchestration, we introduce the *BPE-calculus*, a small language based on the *Business Process Execution Language for Web Services* (BPEL4WS) [5]–a language put forward by IBM, Microsoft et al. for web service orchestration. In the BPE-calculus we abstract from many details of BPEL4WS. Initially, we are only interested if modelling and verification of web service orchestration is feasible using the CWB and the PAC. In the concluding section of this paper we will discuss how the BPE-calculus can be extended to also capture those features of BPEL4WS that are not yet present in the BPE-calculus.

As we already mentioned, the CWB supports model checking, preorder checking and equivalence checking. Next, we will discuss these three verification techniques and their relevance to web service orchestration.

*This research is supported by IBM and the Natural Sciences and Engineering Council of Canada.

1.1 Model Checking

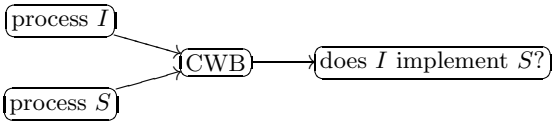
The goal of model checking is to determine whether or not a system satisfies a property. The property is typically expressed in a logic. A variety of different logics have been proposed for this purpose. The CWB supports a particularly expressive logic, namely Kozen's μ-calculus [28]. In the μ-calculus many interesting properties can be expressed including, for example, deadlock freedom. For a more detailed discussion of model checking we refer the reader to, for example, [13].



As argued by Nakajima in [36], model checking is particularly valuable for web service orchestration. For example, assume that we run a BPEL4WS program and that it deadlocks after some time. At that point, many different web services may have been invoked. The roll-back of all the transactions executed by those web services is very costly. Also a considerable amount of network traffic has become void, wasting a valuable publicly shared resource. Using model checking to detect the deadlock before the BPEL4WS program is run may prevent all those roll-backs and waste of network traffic.

1.2 Preorder Checking

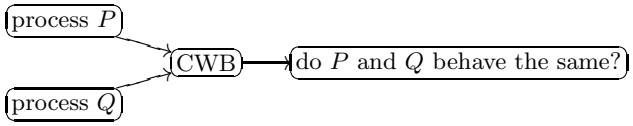
A *behavioural preorder* can be used to express that an implementation satisfies its specification. Several different behavioural preorders have been put forward. The CWB supports two key behavioural preorders: the may testing and must testing preorders of De Nicola and Hennessy [39]. Checking that an implementation and its specification are part of a behavioural preorder, that is, checking if the implementation satisfies its specification, is known as preorder checking.



Currently, web services are usually specified in the *Web Service Description Language* (WSDL) [11]. However, WSDL mainly addresses the format of the messages that are exchanged. Meredith and Bjorg [33] argue for the need of a specification language that is considerably more expressive than WSDL but abstracts from the implementation details of languages such as BPEL4WS. The specification language they propose is similar to our BPE-calculus. Preorder checking can be exploited to check if a BPEL4WS program satisfies such a specification.

1.3 Equivalence Checking

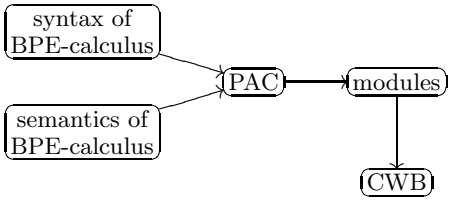
A *behavioural equivalence* equates states of a system that behave the same. A large variety of behavioural equivalences have been proposed. The CWB supports two key behavioural equivalences: bisimilarity, a notion due to Milner [34] and Park [40], and observational equivalence, also due to Milner [34]. Checking if states are behavioural equivalent is known as equivalence checking.



One important application of equivalence checking is state space minimization. That is, the state space of the system is minimized by identifying states that are behaviourally equivalent. Such state space reductions are sometimes essential in order for model checking to be feasible. In the context of web service orchestration, state space minimization is particularly fruitful when the web services are themselves are defined in BPEL4WS. In that case, each BPEL4WS program can be minimized before composition, leading to a considerable state space reduction and, hence, making model checking applicable to larger compositions of web services.

1.4 Overview

The rest of this paper is organized as follows. In Section 2 we introduce the syntax of the BPE-calculus. The semantics of the BPE-calculus is presented in Section 3. The syntax and semantics of the BPE-calculus are used as input to the PAC to generate modules for the CWB. After recompiling the CWB with these new modules, it also supports the BPE-calculus.



In Section 4 we provide some examples showing how the CWB can be used to verify BPE-processes. In the concluding section of this paper, we discuss related and future work.

2. SYNTAX OF THE BPE-CALCULUS

Rather than studying BPEL4WS in its full complexity, we restrict our attention to a small language. We call this language the BPE-calculus as it focuses on the control flow in BPEL4WS and it is similar in flavour to calculi like CCS. In the BPE-calculus, we abstract from many details of BPEL-4WS. In particular, we do not consider data, time, and fault and compensation handlers. In the concluding section of this paper we will briefly discuss how we plan to incorporate these features into the BPE-calculus.

Before defining the syntax of the BPE-calculus, we first fix

- a set $\mathbb{A}$ of external basic activities,
- the internal basic activity τ, and
- an infinite set $\mathbb{L}$ of links.

BPEL4WS contains a variety of basic activities including the invoke, receive, reply and assign activity. We can choose which of these activities we represent as external (that is, observable) basic activities and which ones we represent by the internal (that is, not observable) basic activity τ. Different choices lead to different levels of abstraction. The more basic activities of BPEL4WS are represented by τ, the more abstract the representation of a BPEL4WS business process becomes.

Definition 1. The set $\mathbb{C}$ of join conditions is defined by

$$c ::= \mathit{true} \mid \ell \mid \neg c \mid c \wedge c$$

where $\ell \in \mathbb{L}$. The set $\mathbb{P}$ of BPE-processes is defined by

$$\begin{array}{lcl} P & ::= & 0 \mid \alpha.P \mid \ell \uparrow b.P \mid c \Rightarrow P \mid P \parallel P \mid Q + Q \\ Q & ::= & \alpha.P \mid Q + Q \end{array}$$

where $\alpha \in \mathbb{A} \cup \{\tau\}$, $\ell \in \mathbb{L}$, $b \in \{\mathit{true}, \mathit{false}\}$ and $c \in \mathbb{C}$.

Let us informally define the semantics of the BPE-calculus. A formal semantics will be presented in the next section. The nil process 0 does nothing at all. It corresponds to BPEL4WS' empty activity. The process $\alpha.P$ consists of the basic activity α followed by the process P. That is, P is prefixed by α. BPEL4WS supports sequencing, which is more general than prefixing since a process can be preceded by an arbitrary process rather than just by a basic activity. Sequencing is treated in [27]. In the process $P_1 \parallel P_2$, the processes P_1 and P_2 are concurrent. This corresponds to BPEL4WS' flow construct. In $\alpha_1.P_1 + \alpha_2.P_2$, process $\alpha_1.P_1$ is chosen if basic activity α_1 is enabled (for example, in case α_1 is a receive activity then it is enabled if an appropriate message is available), process $\alpha_2.P_2$ is picked if basic activity α_2 is enabled, and a nondeterministic choice between $\alpha_1.P_1$ and $\alpha_2.P_2$ is made if both α_1 and α_2 are enabled. This corresponds to the pick construct of BPEL4WS. Calculi like CCS contain all the above described constructs.

Synchronization between concurrent processes is provided by means of links. Each link has a source and a target. Furthermore, a transition condition is associated with each link. The latter is a Boolean expression that is evaluated when the source is activated. Its value is associated to the link. As long as the transition condition of a link has not been evaluated, the value of the link is undefined. In the process $\ell \uparrow b.P$, the source of the link ℓ is specified and its transition condition b is given. In the process $c \Rightarrow P$, we associate a join condition c with the process P. This join condition consists of the incoming links of the process combined by Boolean operators. Only when all the values of its incoming links are defined and its join condition evaluates to true, a process can start. As a consequence, if its join condition evaluates to false then the process never starts. For example, in the process $a_1.\ell_1 \uparrow \mathit{true}.0 \parallel a_2.\ell_2 \uparrow \mathit{true}.0 \parallel \ell_1 \wedge \ell_2 \Rightarrow a_3.0$, basic activity a_3 can only be performed after the basic activities a_1 and a_2 have both been performed.

For those that are familiar with BPEL4WS, it should be evident that the resulting calculus is considerably simpler and hence more manageable than BPEL4WS. It captures most of the features of BPEL4WS related to the flow of control. We refer the reader to [27] for a considerably richer calculus.

In BPEL4WS, each link should have a unique source and a unique target. We capture this restriction by means of the following very simple type system. The type of a process P is a pair: the set of links that are used as incoming links in P, and the set of links that are used as outgoing links in P. A process P satisfies the above restriction only if it can be typed, say $P : (I, O)$, and $I = O$.

Definition 2.

$$(\textsc{nil})\ \ 0 : (\emptyset, \emptyset)$$

$$(\textsc{act})\ \ \frac{P : (I, O)}{\alpha.P : (I, O)}$$

$$(\textsc{out})\ \ \frac{P : (I, O) \qquad \ell \notin O}{\ell \uparrow b.P : (I, O \cup \{\ell\})}$$

$$(\textsc{join})\ \ \frac{P : (I, O) \qquad \mathit{links}(c) \cap I = \emptyset}{c \Rightarrow P : (I \cup \mathit{links}(c), O)}$$

$$(\textsc{flow})\ \ \frac{P_1 : (I_1, O_1)\ \ P_2 : (I_2, O_2)\ \ I_1 \cap I_2 = \emptyset\ \ O_1 \cap O_2 = \emptyset}{P_1 \parallel P_2 : (I_1 \cup I_2, O_1 \cup O_2)}$$

$$(\textsc{pick})\ \ \frac{P_1 : (I_1, O_1)\ \ P_2 : (I_2, O_2)\ \ I_1 \cap I_2 = \emptyset\ \ O_1 \cap O_2 = \emptyset}{P_1 + P_2 : (I_1 \cup I_2, O_1 \cup O_2)}$$

In the above definition, we use $\mathit{links}(c)$ to denote the set of links that occur in the join condition c. This set is defined by

$$\begin{array}{rcl} \mathit{links}(\mathit{true}) & = & \emptyset \\ \mathit{links}(\ell) & = & \{\ell\} \\ \mathit{links}(\neg c) & = & \mathit{links}(c) \\ \mathit{links}(c_1 \wedge c_2) & = & \mathit{links}(c_1) \cup \mathit{links}(c_2) \end{array}$$

Not every process can be typed. Consider, for example, the process $\ell \uparrow \mathit{true}.0 \parallel \ell \uparrow \mathit{true}.0$. According to axiom (NIL), the process 0 has type $(\emptyset, \emptyset)$. From rule (OUT) we can conclude that process $\ell \uparrow \mathit{true}$ has type $(\emptyset, \{\ell\})$. As a consequence, the rule (FLOW) is not applicable to the process $\ell \uparrow \mathit{true}.0 \parallel \ell \uparrow \mathit{true}.0$. Hence, this process cannot be typed. However, if a process is well-typed then its type is unique.

PROPOSITION 3. *If $P : (I_1, O_1)$ and $P : (I_2, O_2)$ then $I_1 = I_2$ and $O_1 = O_2$.*

PROOF. Follows immediately from the fact that there is at most one rule applicable to each process. □

Furthermore, each type is finite.

PROPOSITION 4. *If $P : (I, O)$ then I and O are finite sets of links.*

PROOF. By structural induction on P. □

3. SEMANTICS OF THE BPE-CALCULUS

In the previous section, we introduced the syntax of the BPE-calculus. Next, we present its semantics. We model the BPE-calculus by means of the structural operational approach due to Plotkin [42].

In our model, we need to keep track of the values of the links. The value of a link is either true, false, or undefined. The latter we denote by $\perp$. The values of the links is captured by an element of

$$\Lambda = \mathbb{L} \to \{true, false, \perp\}.$$

Initially, all the links are undefined. Setting the value of all the links in the set L to the Boolean value b we denote by $\{L \mapsto b\}$. That is, $\lambda\{L \mapsto b\}$ is defined by

$$\lambda\{L \mapsto b\}(\ell) = \begin{cases} b & \text{if } \ell \in L \\ \lambda(\ell) & \text{otherwise.} \end{cases}$$

Instead of $\lambda\{\{\ell\} \mapsto b\}$ we will often write $\lambda\{\ell \mapsto b\}$.

A *labelled transition system* is similar to a nondeterministic finite automaton. It consists of a set of states, a set of labels and a set of transitions. Note that the sets of states and labels may be infinite and that there are no initial or final states. In our labelled transition system for the BPE-calculus, each state consists of a pair: a process and the values of the links. That is, a state is an element of $\mathbb{P} \times \Lambda$. As labels we use basic activities, that is, elements of $\mathbb{A} \cup \{\tau\}$. The transitions are defined as a relation, that is, a subset of $(\mathbb{P} \times \Lambda) \times (\mathbb{A} \cup \{\tau\}) \times (\mathbb{P} \times \Lambda)$. This relation is denoted by $\to$. Instead of $(\langle P, \lambda\rangle, \alpha, \langle P', \lambda'\rangle) \in \to$ we will write $\langle P, \lambda\rangle \xrightarrow{\alpha} \langle P', \lambda'\rangle$. Such a transition tells us that the process P whose links have values specified by λ can perform the basic activity α to become the process P' whose links have values captured by λ'. Performing the basic activity α often involves interaction with the environment. For example, in case α is a receive activity, a message is received from the environment. The transition relation is defined in terms of a collection of axioms and rules that are driven by the syntactic structure of the processes involved.

Definition 5.

(ACT) $\langle \alpha.P, \lambda\rangle \xrightarrow{\alpha} \langle P, \lambda\rangle$

(OUT) $\langle \ell \uparrow b.P, \lambda\rangle \xrightarrow{\tau} \langle P, \lambda\{\ell \mapsto b\}\rangle$

(JOIN$_t$) $$\frac{\mathcal{C}(c)(\lambda) = true}{\langle c \to P, \lambda\rangle \xrightarrow{\tau} \langle P, \lambda\rangle}$$

(JOIN$_f$) $$\frac{\mathcal{C}(c)(\lambda) = false \quad P : (I, O)}{\langle c \to P, \lambda\rangle \xrightarrow{\tau} \langle 0, \lambda\{O \mapsto false\}\rangle}$$

(FLOW$_\ell$) $$\frac{\langle P_1, \lambda\rangle \xrightarrow{\alpha} \langle P_1', \lambda'\rangle}{\langle P_1 \parallel P_2, \lambda\rangle \xrightarrow{\alpha} \langle P_1' \parallel P_2, \lambda'\rangle}$$

(FLOW$_r$) $$\frac{\langle P_2, \lambda\rangle \xrightarrow{\alpha} \langle P_2', \lambda'\rangle}{\langle P_1 \parallel P_2, \lambda\rangle \xrightarrow{\alpha} \langle P_1 \parallel P_2', \lambda'\rangle}$$

(PICK$_\ell$) $$\frac{\langle P_1, \lambda\rangle \xrightarrow{\alpha} \langle P_1', \lambda'\rangle \quad P_2 : (I_2, O_2)}{\langle P_1 + P_2, \lambda\rangle \xrightarrow{\alpha} \langle P_1', \lambda'\{O_2 \mapsto false\}\rangle}$$

(PICK$_r$) $$\frac{\langle P_2, \lambda\rangle \xrightarrow{\alpha} \langle P_2', \lambda'\rangle \quad P_1 : (I_1, O_1)}{\langle P_1 + P_2, \lambda\rangle \xrightarrow{\alpha} \langle P_2', \lambda'\{O_1 \mapsto false\}\rangle}$$

Let us briefly discuss the above axioms and rules.

(NIL) There is no rule applicable to the process 0 and, hence, it cannot make any transitions.

(ACT) The process $\alpha.P$ is capable of making a transition labelled α. After the basic activity α has been performed, the process P remains to be executed.

(OUT) The value of the outgoing link ℓ in the process $\ell \uparrow b.P$ is set to b. This is reflected by the change of λ to $\lambda\{\ell \mapsto b\}$. The update of λ is modelled by a τ-transition since it is an internal (that is, not observable) action. After the update, the process P remains.

(JOIN) To model the evaluation of join conditions, we introduce the function $\mathcal{C} : \mathbb{C} \to \Lambda \to \{true, false, \perp\}$ defined by

$$\begin{array}{rcl} \mathcal{C}(true)(\lambda) & = & true \\ \mathcal{C}(\uparrow\ell)(\lambda) & = & \lambda(\ell) \\ \mathcal{C}(\neg c)(\lambda) & = & \neg\mathcal{C}(c)(\lambda) \\ \mathcal{C}(c_1 \wedge c_2)(\lambda) & = & \mathcal{C}(c_1)(\lambda) \wedge \mathcal{C}(c_2)(\lambda) \end{array}$$

where

	true	*false*	$\perp$
$\neg$	*false*	*true*	$\perp$

and

$\wedge$	*true*	*false*	$\perp$
true	*true*	*false*	$\perp$
false	*false*	*false*	$\perp$
$\perp$	$\perp$	$\perp$	$\perp$

In case the join condition c evaluates to true, the process P is executed. In case the join condition c is false, the process P is skipped. In addition to skipping P, dead-path-elimination (DPE) [30] is triggered in that case. It sets all the outgoing links of process P to false. This is done to eliminate processes that will never be executed. For example, consider the process $false \Rightarrow \ell \uparrow true.0 \parallel \ell \Rightarrow P$. Process P will never be executed. DPE "garbage collects" this process. For a detailed study of DPE we refer the reader to [8]. In both rules a τ-transition is used, since the evaluation of a join condition and the execution of DPE are both internal actions.

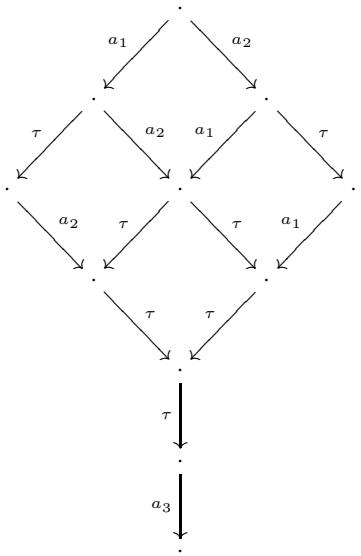
(FLOW) In the process $P_1 \parallel P_2$, the transitions of the processes P_1 and P_2 are interleaved.

(PICK) The pick construct performs a choice. This choice is dictated by the environment. The process that is capable of interacting with the environment is chosen. If both processes are capable of such interaction then the choice is nondeterministic. The process that is not chosen is simply discarded. Furthermore, DPE sets all of the outgoing links of the discarded process to false. Consider, for example, the process $(a_1.0 + a_2.\ell \uparrow true.0) \parallel \ell \Rightarrow P$. If this process interacts with the environment through basic activity a_1 then process P will never be executed. In that case, DPE "garbage collects" process P.

Let us look at an example. Consider $a_1.\ell_1 \uparrow \mathit{true}.0 \parallel a_2.\ell_2 \uparrow \mathit{true}.0 \parallel \ell_1 \wedge \ell_2 \Rightarrow a_3.0$. According to axiom (ACT), state $\langle a_1.\ell_1 \uparrow \mathit{true}.0, \lambda\rangle$ can make a transition labelled a_1 to state $\langle \ell_1 \uparrow \mathit{true}.0, \lambda\rangle$. Applying rule (FLOW$_\ell$) twice, we can conclude that state $\langle a_1.\ell_1 \uparrow \mathit{true}.0 \parallel a_2.\ell_2 \uparrow \mathit{true}.0 \parallel \ell_1 \wedge \ell_2 \Rightarrow a_3.0, \lambda\rangle$ can make a transition labelled a_1. The resulting state is $\langle \ell_1 \uparrow \mathit{true}.0 \parallel a_2.\ell_2 \uparrow \mathit{true}.0 \parallel \ell_1 \wedge \ell_2 \Rightarrow a_3.0, \lambda\rangle$. Subsequent transitions can be proved similarly. For example, we can prove the following sequence of transitions.

$$
\begin{aligned}
&\langle a_1.\ell_1 \uparrow \mathit{true}.0 \parallel a_2.\ell_2 \uparrow \mathit{true}.0 \parallel \ell_1 \wedge \ell_2 \Rightarrow a_3.0, \lambda\rangle \\
&\quad \xrightarrow{a_1} \quad \langle \ell_1 \uparrow \mathit{true}.0 \parallel a_2.\ell_2 \uparrow \mathit{true}.0 \parallel \ell_1 \wedge \ell_2 \Rightarrow a_3.0, \lambda\rangle \\
&\quad \xrightarrow{a_2} \quad \langle \ell_1 \uparrow \mathit{true}.0 \parallel \ell_2 \uparrow \mathit{true}.0 \parallel \ell_1 \wedge \ell_2 \Rightarrow a_3.0, \lambda\rangle \\
&\quad \xrightarrow{\tau} \quad \langle 0 \parallel \ell_2 \uparrow \mathit{true}.0 \parallel \ell_1 \wedge \ell_2 \Rightarrow a_3.0, \lambda\{\ell_1 \mapsto \mathit{true}\}\rangle \\
&\quad \xrightarrow{\tau} \quad \langle 0 \parallel 0 \parallel \ell_1 \wedge \ell_2 \Rightarrow a_3.0, \lambda\{\{\ell_1, \ell_2\} \mapsto \mathit{true}\}\rangle \\
&\quad \xrightarrow{\tau} \quad \langle 0 \parallel 0 \parallel a_3.0, \lambda\{\{\ell_1, \ell_2\} \mapsto \mathit{true}\}\rangle \\
&\quad \xrightarrow{a_3} \quad \langle 0 \parallel 0 \parallel 0, \lambda\{\{\ell_1, \ell_2\} \mapsto \mathit{true}\}\rangle
\end{aligned}
$$

All possible transition sequences are combined into the following diagram.



Note that basic activity a_3 can only be performed after the basic activities a_1 and a_2 have both been executed.

Next, we show that if a process can be typed, then all processes reachable from that process by means of a transition can be typed as well. This property is known as subject reduction.

PROPOSITION 6. *If $P : (I, O)$ and $\langle P, \lambda\rangle \xrightarrow{\alpha} \langle P', \lambda'\rangle$ then $P : (I', O')$ for some $I' \subseteq I$ and $O' \subseteq O$.*

PROOF. By induction on the proof of $\langle P, \lambda\rangle \xrightarrow{\alpha} \langle P', \lambda'\rangle$. □

Bisimilarity is one of the key behavioural equivalence relations on the states of a labelled transition system. In our setting, it amounts to the following.

Definition 7. A relation $\mathcal{R} \subseteq (\mathbb{P} \times \Lambda) \times (\mathbb{P} \times \Lambda)$ is a bisimulation if for all $(\langle P_1, \lambda_1\rangle, \langle P_2, \lambda_2\rangle) \in \mathcal{R}$,

- if $\langle P_1, \lambda_1\rangle \xrightarrow{\alpha} \langle P_1', \lambda_1'\rangle$ then $\langle P_2, \lambda_2\rangle \xrightarrow{\alpha} \langle P_2', \lambda_2'\rangle$ for some $\langle P_2', \lambda_2'\rangle$ such that $(\langle P_1', \lambda_1'\rangle, \langle P_2', \lambda_2'\rangle) \in \mathcal{R}$.
- if $\langle P_2, \lambda_2\rangle \xrightarrow{\alpha} \langle P_2', \lambda_2'\rangle$ then $\langle P_1, \lambda_1\rangle \xrightarrow{\alpha} \langle P_1', \lambda_1'\rangle$ for some $\langle P_1', \lambda_1'\rangle$ such that $(\langle P_1', \lambda_1'\rangle, \langle P_2', \lambda_2'\rangle) \in \mathcal{R}$.

States $\langle P_1, \lambda_1\rangle$ and $\langle P_2, \lambda_2\rangle$ are bisimilar, denoted $\langle P_1, \lambda_1\rangle \sim \langle P_2, \lambda_2\rangle$, if $(\langle P_1, \lambda_1\rangle, \langle P_2, \lambda_2\rangle) \in \mathcal{R}$ for some bisimulation $\mathcal{R}$. Processes P_1 and P_2 are bisimilar, denoted $P_1 \sim P_2$, if $\langle P_1, \lambda\rangle$ and $\langle P_2, \lambda\rangle$ are bisimilar for all $\lambda \in \Lambda$.

If λ_1 and λ_2 agree on the values of the links that are used as incoming links in process P, then the states $\langle P, \lambda_1\rangle$ and $\langle P, \lambda_2\rangle$ are bisimilar. The restriction of λ to I we denote by $\lambda \restriction I$.

PROPOSITION 8. *If $P : (I, O)$ and $\lambda_1 \restriction I = \lambda_2 \restriction I$ then $\langle P, \lambda_1\rangle \sim \langle P, \lambda_2\rangle$.*

PROOF. It suffices to show that

$$\mathcal{R} = \{\, (\langle P, \lambda_1\rangle, \langle P, \lambda_2\rangle) \mid P : (I, O) \wedge \lambda_1 \restriction I = \lambda_2 \restriction I \,\}$$

is a bisimulation. First, we can prove that if $\langle P, \lambda\rangle \xrightarrow{\alpha} \langle P', \lambda'\rangle$, $P : (I, O)$ and $P' : (I', O')$, then $\lambda \restriction (\mathbb{L} \setminus O) \cup O' = \lambda' \restriction (\mathbb{L} \setminus O) \cup O'$. This can be shown by transition induction. Second, we can prove that if $\langle P, \lambda_1\rangle \xrightarrow{\alpha} \langle P', \lambda_1'\rangle$, $P : (I, O)$, $P' : (I', O')$, and $\lambda_1 \restriction I = \lambda_2 \restriction I$, then $\langle P, \lambda_2\rangle \xrightarrow{\alpha} \langle P', \lambda_2'\rangle$ for some λ_2 such that $\lambda_1' \restriction I' \cup (O \setminus O') = \lambda_2' \restriction I' \cup (O \setminus O')$. Also this second property can be proved by transition induction, exploiting the first property for the rules (FLOW$_\ell$) and (FLOW$_r$). From this second property we can immediately conclude that $\mathcal{R}$ is a bisimulation. □

As a consequence, to check if processes P_1 and P_2 with types (I_1, O_1) and (I_2, O_2) are bisimilar, we only need to check $\langle P_1, \lambda\rangle \sim \langle P_2, \lambda\rangle$ for all $\lambda \in \Lambda$ with $\lambda(\ell) = \bot$ for all $\ell \notin I_1 \cup I_2$.

Next, we show that bisimilarity is a congruence.

PROPOSITION 9. *If $P_1 \sim P_1'$ then $\alpha.P_1 \sim \alpha.P_1'$, $\ell \uparrow b.P_1 \sim \ell \uparrow b.P_1'$, $c \Rightarrow P_1 \sim c \Rightarrow P_1'$, $P_1 \parallel P_2 \sim P_1' \parallel P_2$ and $P_1 + P_2 \sim P_1' + P_2$.*

PROOF. The proof of this proposition is a straightforward modification of the proof that bisimilarity is a congruence in CCS as shown in, for example, [35, Proposition 4.10]. For example, one can show that

$$\{\, (\langle P_1 \parallel P_2, \lambda\rangle, \langle P_1' \parallel P_2, \lambda'\rangle) \mid \langle P_1, \lambda\rangle \sim \langle P_1', \lambda'\rangle \,\}$$

is a bisimulation. □

As a consequence, we can reason about bisimilarity in a compositional way. For example, consider that process P_2

can be obtained from process P_1 by replacing the subprocess Q_1 of P_1 with Q_2. If Q_1 and Q_2 are bisimilar then we can conclude from the above proposition that P_1 and P_2 are bisimilar as well.

For the definition of observational equivalence, another key behavioural equivalence relation on the states of a labelled transition system, we refer the reader to [34]. Properties similar to the ones formulated in Proposition 8 and 9 can be proved for observational equivalence.

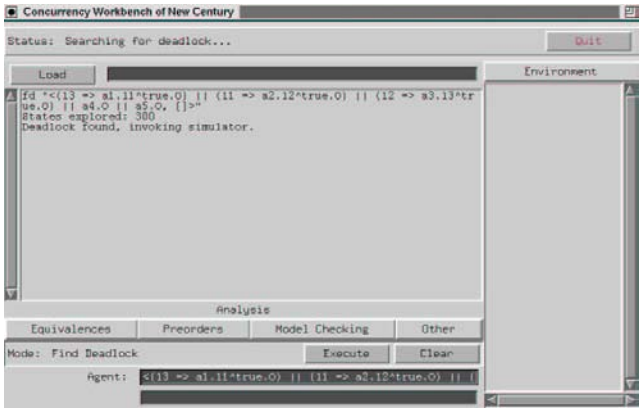
4. EXAMPLES

Below, we present some toy examples illustrating how the CWB, modified to support the BPE-calculus, can be exploited to verify BPE-processes. We also briefly discuss a more realistic example. For more details, we refer the reader to [27].

4.1 Model Checking

Recall that the goal of model checking is the determine if a process satisfies a property. Consider, for example, the process $\ell_3 \Rightarrow a_1.\ell_1 \uparrow true.0 \parallel \ell_1 \Rightarrow a_2.\ell_2 \uparrow true.0 \parallel \ell_2 \Rightarrow a_3.\ell_3 \uparrow true.0 \parallel a_4.0 \parallel a_5.0 \parallel a_6.0$. Since deadlock freedom can be expressed in terms of a formula of the μ-calculus and the CWB supports model checking of μ-calculus formulae, we can use the CWB to check if the above process is free of deadlock. It will not come as a surprise that the above process does not satisfy this property. The CWB reports this.

Alternatively, we can also press the "find deadlock" button on the graphical user interface of the CWB. Since a deadlock is found, as shown below, CWB's simulator is invoked.



This simulator allows us to trace a sequence of transitions from the initial state to the one that causes a deadlock. In the picture below, we have only traced one transition. That is, state 2 is the current state and state 3 is the next state.

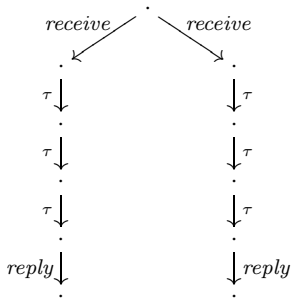
Besides checking for deadlock, we can verify other interesting properties as well. Some examples will be presented in Section 4.4.

4.2 Preorder Checking

To verify that an implementation satisfies its specification we can exploit preorder checking. Consider, for example, the process $(receive.\tau.\ell_1 \uparrow true.0 + receive.\tau.\ell_2 \uparrow true.0) \parallel \ell_1 \vee \ell_2 \Rightarrow reply.0$. In this process, the τ's may be invocations of web services. To check that this process satisfies the specification captured by the process $receive.reply.0$ we can exploit the CWB. The CWB reports that the specification (process) and its implementation (process) are related according to both the may testing preorder and the must testing preorder.

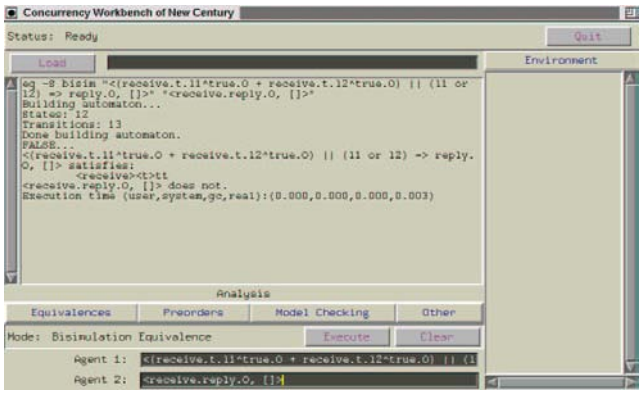
4.3 Equivalence Checking

Recall that equivalence checking can be exploited to identify states that are behaviourally equivalent which may result in a system with a smaller state space. For example, consider the process $(receive.\tau.\ell_1 \uparrow true.0 + receive.\tau.\ell_2 \uparrow true.0) \parallel \ell_1 \vee \ell_2 \Rightarrow reply.0$. Its transitions can be depicted as follows.



The system has 11 states. However, the process is observationally equivalent to the process $receive.reply.0$. This system has only 3 states. The CWB reports that the processes are observationally equivalent. The CWB also tells us that the processes are not bisimilar. Furthermore, in that case the CWB also provides a property, formulated in terms of

the μ-calculus, that is satisfied by one of the processes but not by the other process, as shown below.

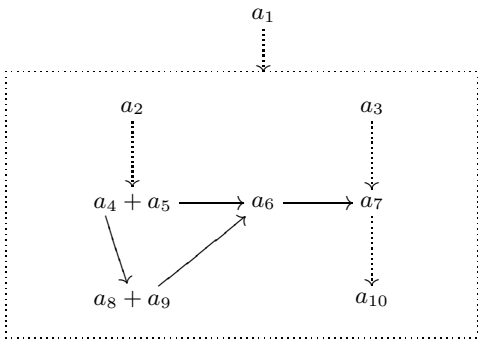


Alternatively, we can also press the "minimize" button on the graphical user interface of the CWB. We can minimize the process either with respect to observational equivalence or bisimilarity.

4.4 Book Ordering

Next, we consider the book ordering process described by Van der Aalst in [2]. We have modelled this business process in our BPE-calculus and we have verified it using the CWB. Below, we present a simplified version of this business process.

The process starts when an order is received from the customer. This basic activity is denoted a_1. The order is processed by checking if the book is in stock (activity a_2) and by calculating the price of the order (activity a_3) concurrently. Either the book is in stock (activity a_5) in which case it can be shipped (activity a_6) or the book is not in stock (activity a_4) in which case an attempt is made to re-stock. If the attempt is successful (activity a_9) then the book can be shipped. Otherwise the customer is notified that the order cannot be completed (activity a_8). The bill is sent (activity a_7) only once the book has been shipped and the price has been calculated. The process then waits to receive a payment from the customer (activity a_{10}).



In the above diagram, the dotted box denotes a flow. The dotted arrows denote prefixing. The solid arrows denote links. All transition conditions are *true*. Those activities that have incoming links have a join condition. Its join condition is the disjunction of the incoming links.

Using the CWB, we can verify that the above process is free of deadlock. We can also check that it is possible to ship the book after replenishing stock. Furthermore, we can verify that at some point after the order has been received the stock is checked. All these properties can be expressed in the μ-calculus. We refer the reader to [27] for more details. Using a richer calculus, the above specification can be refined. For this more detailed process, several other interesting properties can be checked as described in [27].

5. CONCLUSION

Let us first summarize our contributions. We introduced a new calculus, named the BPE-calculus, that contains the main control flow constructs of BPEL4WS. We modelled our BPE-calculus by means of a structural operational semantics. The syntax and semantics of the BPE-calculus were used as input of the PAC to produce modules for the CWB to support the BPE-calculus. The modified CWB provides us with a powerful verification tool for BPE-processes.

5.1 Related Work

In the literature, a number of calculi to model business processes have been proposed. For example, in [10], Chessell et al. introduce the business process modelling language StAC. This language shows similarities with our BPE-calculus. The focus of StAC is on compensation handlers, an ingredient of BPEL4WS that is not considered in the BPE-calculus. Chessell et al. do not provide a formal semantics for StAC. The πt-calculus of Bocchi, Laneve and Zavattaro [7] is another calculus that includes compensation handlers. The πt-calculus is modelled by means of a reduction semantics, an approach closely related to the structural operational approach taken in this paper. In [41], Piccinelli and Williams present the calculus DySCo for business processes that is similar to CCS. They model DySCo by means of a structural operational semantics. In contrast to this paper, the above mentioned papers consider neither advanced synchronization patterns like DPE, nor verification.

One can distinguish three different approaches to verify programs. One can translate a program into the input language of an existing verification tool, one can develop a new tool that can handle the program directly, or one can use a combination these two approaches. For a discussion of the advantages and the disadvantages we refer the reader to, for example, [45]. Next, we discuss verification tools for web service orchestration that are closely related to ours. All use the first approach whereas we exploit the third approach.

In [43], Schroeder presents a translation of business processes into CCS. Subsequently, the CWB can be used for verification. The business process language that is studied in that paper is considerably simpler than our BPE-calculus. It is not clear to us if this approach is also applicable to BPEL4WS. In particular, it is not clear how to capture DPE in CCS (it can be done though, since CCS is Turing complete).

Nakajima [37] describes how to use the SPIN model checker [23] to verify web service orchestration. The language used

to compose web services is the Web Services Flow Language (WSFL) [29] which is one of BPEL4WS's predecessors. In order to do the verification using SPIN, business processes are first translated into Promela, the specification language provided by SPIN. SPIN only provides model checking and not preorder checking or equivalence checking.

In [26], Koehler, Kumaran and Tirenni model business processes as nondeterministic automata with state variables and transition guards. These automata are subsequently translated into the input language of the model checker NuSMV [12]. Koehler et al. show how NuSMV can be exploited to detect termination of business processes.

A similar approach is taken by Karamanolis, Giannakopoulou, Magee and Wheater in [25]. They translate business processes into FSP-processes and use the LTSA toolkit [31] for model checking. The LTSA toolkit allows the user to specify properties in terms of deterministic FSP-processes. In [19], Foster, Uchitel, Magee and Kramer describe a BPEL-4WS plug-in for the LTSA toolkit. They translate BPEL-4WS program into FSP-processes and subsequently use the LTSA toolkit to verify the FSP-processes. The LTSA toolkit only supports model checking. The set of properties that can be expressed as deterministic FSP-processes is smaller than the set of properties that can be captured by the μ-calculus.

In our study, we used a labelled transition system to model the BPE-calculus. Petri nets provide an alternative approach to model business processes. For an overview of modelling business processes by means of Petri nets we refer the reader to, for example, [3]. These Petri nets can also be used as a basis to develop verification tools. For examples, we refer the reader to, for example, the work of Van der Aalst [1], Martens [32], and Narayanan and McIlraith [38]. We believe that labelled transition systems are superior to Petri nets when it comes to modelling BPEL4WS, since, as also pointed out by Van der Aalst and Ter Hofstede in [4], advanced synchronization patterns like DPE cannot easily be captured by means of Petri nets (although it can be done, as shown by Martens [32]).

5.2 Future Work

We have already developed a tool that translates a BPEL-4WS program into a BPE-process. Using this tool, we have applied model checking, preorder checking and equivalence checking to a number of BPEL4WS programs. We were pleasantly surprised by the size of the labelled transition systems corresponding to BPE-processes that model BPEL4WS programs. For instance, using our tool we translated a realistic travel booking BPEL4WS program into the BPE-calculus. The so-obtained BPE-process gives rise to a labelled transition system with only 59 states. To put this in perspective, the CWB can handle labelled transition systems with millions of states. (It is not difficult to come up with a BPE-process the labelled transition of which consists of millions of states, but we have not encountered such BPE-processes in practice.) As a consequence, we are confident that BPEL4WS programs can be modelled in much more detail while still being verifiable by means of the CWB.

We are interested to extend the BPE-calculus so that it covers even more of BPEL4WS. Initially, we plan to tackle compensation handlers and fault handlers. Here, we can build on the work on compensation handlers of Bocchi, Laneve and Zavattaro [7] and Chessell et al. [10]. Next, we plan to incorporate time into our model. In this case, we can exploit the work of Bhat, Cleaveland and Lüttgen [6]. Finally, we plan introduce some data into our BPE-calculus. Here we can fruitfully exploit predicate abstraction, as first proposed by Graf and Saidi [22].

We are also interested to extend our calculus to model and verify multiple interacting BPEL4WS programs. Fu, Bultan and Su have addressed this problem in [20]. They translate BPEL4WS programs into guarded automata. These automata are subsequently translated into Promela and can be verified using the SPIN model checker. We plan to tackle this problem using the CWB.

These extensions would allow us to model web service orchestration described in BPEL4WS more accurately and it would form the basis of an even more powerful tool for the verification of web service orchestration.

6. ACKNOWLEDGEMENTS

The authors would like to thank Jon Bennett, Rance Cleaveland, Bill O'Farrell, Parke Godfrey, Yves Lesperance, Fatima Ashfaq Ramay and the referees.

7. REFERENCES

[1] W.M.P. van der Aalst. Verification of workflow nets. In P. Azéma and G. Balbo, editors, *Proceedings of the 18th International Conference on Applications and Theory in Petri Nets*, volume 1248 of *Lecture Notes in Computer Science*, pages 407–426, Toulouse, June 1997. Springer-Verlag.

[2] W.M.P. van der Aalst. Challenges in business process management: Verification of business processes using Petri nets. *Bulletin of the EATCS*, 80:174–199, June 2003.

[3] W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems*. The MIT Press, 2002.

[4] W.M.P. van der Aalst and A.H.M. ter Hofstede. Workflow patterns: on the expressive power of (Petri-net-based) workflow languages. In K. Jensen, editor, *Proceedings of the Fourth Workshop on the Practical Use of Coloured Petri Nets and CPN Tools*, volume 560 of *DAIMI PB series*, pages 1–20, Aarhus, August 2002. University of Aarhus.

[5] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business process execution language for web services, version 1.1. Available at `www.ibm.com/developerworks/webservices/library/ws-bpel`, May 2003.

[6] G. Bhat, R. Cleaveland, and G. Lüttgen. A practical approach to implementing real-time semantics. *Annals of Software Engineering*, 7(1–4):127–155, 1999.

[7] L. Bocchi, C. Laneve, and G. Zavattaro. A calculus for long running transactions. In E. Najm, U. Nestmann,

and P. Stevens, editors, *Proceedings of the 6th IFIP WG 6.1 International Conference on Formal Methods for Open Object-Based Distributed Systems*, volume 2884 of *Lecture Notes in Computer Science*, pages 124–138, Paris, December 2003. Springer-Verlag.

[8] F. van Breugel and M. Koshkina. Does dead-path-elimination have side effects? Technical Report CS-2003-04, York University, Toronto, April 2003.

[9] J. Chen. On verifying distributed multithreaded Java programs. In *Proceedings of the 33rd Hawaii International Conference on Systems Sciences*, pages 2930–2939, Hawaii, January 2000. IEEE.

[10] M. Chessell, C. Griffin, D. Vines, M. Butler, C. Ferreira, and P. Henderson. Extending the concept of transaction compensation. *IBM Systems Journal*, 41(4):743–758, 2002.

[11] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web services description language (WSDL) version 1.1. Available at `www.w3.org/TR/2001/NOTE-wsdl-20010315`, March 2001.

[12] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. NUSMV: a new symbolic model checker. *Software Tools for Technology Transfer*, 2(4):410–425, March 2000.

[13] E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. The MIT Press, 1999.

[14] R. Cleaveland, T. Li, and S. Sims. The Concurrency Workbench of the New Century user's manual. Available at `www.cs.sunysb.edu/~cwb`, July 2000.

[15] R. Cleaveland, G. Lüttgen, V. Natarajan, and S. Sims. Modeling and verifying distributed systems using priorities: a case study. *Software Concepts Tools*, 17(2):50–62, 1996.

[16] R. Cleaveland and S.T. Sims. The NCSU concurrency workbench. In R. Alur and T. Henzinger, editors, *Proceedings of the 8th Conference on Computer-Aided Verification*, volume 1102 of *Lecture Notes in Computer Science*, pages 394–397, New Brunswick, NJ, July 1996. Springer-Verlag.

[17] R. Cleaveland and S.T. Sims. Generic tools for verifying concurrent systems. *Science of Computer Programming*, 42(1):39–47, January 2002.

[18] W.M. Elseaidy, R. Cleaveland, and J.W. Baugh Jr. Modeling and verifying active structural control systems. *Science of Computer Programming*, 29(1–2):99–122, July 1997.

[19] H. Foster, S. Uchitel, J. Magee, and J. Kramer. Model-based verification of web service composition. In *Proceedings the 18th IEEE International Conference on Automated Software Engineering*, pages 152–161, Montreal, October 2003. IEEE.

[20] X. Fu, T. Bultan, and J. Su. Analysis of interacting BPEL web services. In *Proceedings of the 13th International World Wide Web Conference*, New York, May 2004. ACM.

[21] S. Gradara, A. Santone, M.L. Villani, and G. Vaglini. Model checking multithreaded programs by means of reduced models. In *Proceedings of the 4th Workshop on Language Descriptions, Tools and Applications*, Electronic Notes in Theoretical Computer Science, Barcelona, April 2004. Elsevier.

[22] S. Graf and H. Saidi. Construction of abstract state graphs with PVS. In O. Grumberg, editor, *Proceedings of the 9th International Conference on Computer Aided Verification*, volume 1254 of *Lecture Notes in Computer Science*, pages 72–83, Haifa, June 1997. Springer-Verlag.

[23] G.J. Holzmann. *The Spin Model Checker: Primer and Reference Manual*. Addison-Wesley, 2003.

[24] H.K. Kapoor and M.B. Josephs. Modelling and verification of delay-insensitive circuits using CCS and the Concurrency Workbench. To appear in *Information Processing Letters*. Available at `myweb.lsbu.ac.uk/~kapoorhk`, 2003.

[25] C. Karamanolis, D. Giannakopoulou, J. Magee, and S.M. Wheater. Model checking of workflow schemas. In *Proceedings of the 4th International Enterprise Distributed Object Computing Conference*, pages 170–179, Makuhari, Japan, September 2000. IEEE.

[26] J. Koehler, G. Tirenni, and S. Kumaran. From business process model to consistent implementation: a case study for formal verification methods. In *Proceedings of the 6th International Enterprise Distributed Object Computing Conference*, pages 96–106, Lausanne, September 2002. IEEE.

[27] M. Koshkina. Verification of business processes for web services. Master's thesis, York University, Toronto, October 2003. Available at `www.cs.yorku.ca/~franck/students`.

[28] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[29] F. Leymann. Web services flow language. Available at `www.ibm.com/software/solutions/webservices/pdf/WSFL.pdf`, May 2001.

[30] F. Leymann and W. Altenhuber. Managing business processes as an information resource. *IBM Systems Journal*, 33(2):326–348, 1994.

[31] J. Magee and J. Kramer. *Concurrency: State Models and Java Programs*. John Wiley and Sons, 1999.

[32] A. Martens. *Distributed Business Processes — Modeling and Verification by help of Web Services*. PhD thesis, Humboldt-Universität zu Berlin, July 2003. Available at `www.informatik.hu-berlin.de/top/download/documents/pdf/Mar03.pdf`.

[33] L.G. Meredith and S. Bjorg. Contracts and types. *Communications of the ACM*, 46(10):41–47, October 2003.

[34] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, 1980.

[35] R. Milner. *Communication and Concurrency*. Prentice Hall International, 1989.

[36] S. Nakajima. On verifying web service flows. In *Proceedings of the Symposium on Applications and the Internet*, pages 223–224, Nara City, Japan, January/February 2002. IEEE.

[37] S. Nakajima. Verification of web service flows with model-checking techniques. In *Proceedings of the 1st International Symposium on Cyber Worlds*, pages 378–386, Tokyo, November 2002. IEEE.

[38] S. Narayanan and S.A. McIlraith. Simulation, verification and automated composition of web services. In *Proceedings of the 11th International World Wide Web Conference*, pages 77–88, Honolulu, May 2002. ACM.

[39] R. De Nicola and M. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

[40] D. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *Proceedings of 5th GI-Conference on Theoretical Computer Science*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183, Karlsruhe, March 1981. Springer-Verlag.

[41] G. Piccinelli and S.L. Williams. Workflow: A language for composing web services. In W.M.P. van der Aalst, A.H.M. ter Hofstede, and M. Weske, editors, *Proceedings of the International Conference on Business Process Management*, volume 2678 of *Lecture Notes in Computer Science*, pages 1–12, Eindhoven, June 2003. Springer-Verlag.

[42] G.D. Plotkin. A structural approach to operational semantics. Report DAIMI FN-19, Aarhus University, Aarhus, September 1981.

[43] M. Schroeder. Verification of business processes for a correspondence handling center using CCS. In A.I. Vermesan and F. Coenen, editors, *Proceedings of European Symposium on Validation and Verification of Knowledge Based Systems and Components*, pages 1–15, Oslo, June 1999. Kluwer.

[44] S. Sims. The Process Algebra Compiler user's manual. Available at `www.reactive-systems.com/pac`, November 1999.

[45] W. Visser, K. Havelund, G. Brat, S. Spark, and F. Lerda. Model checking programs. *Automated Software Engineering*, 10(2):203–232, April 2003.