

# Discrete Event Models for Web Service Processes

Yuhong Yan

Department of Computer Science and Software Engineering  
Concordia University, Montreal, Canada  
yuhong@encs.concordia.ca

## Abstract

Web service processes are business processes composed of individual Web services. Web service process description languages, both choreography and orchestration languages, are influenced by techniques from workflow modeling, formal methods and software engineering. Since they are software script languages to be executed by a process engine, their expressive power indeed is beyond classic discrete event system models, such as process algebra and Petri nets. This chapter analyzes and compares different Web service process description languages, discusses the issues in using discrete event system models to model Web service processes, and also compares Web service processes with workflow processes. This chapter also discusses the suitable methods based on the formal models for various computing tasks, such as verification and validation.

## 1 Introduction

Service computing views business entities as service providers and models business processes as compositions of multiple services. Service-Oriented Architecture (SOA) and Web services are two related concepts in Service computing. In W3C documents, SOA typically refers to Web services (Booth, Haas, McCabe, & et.al., 2004) and the two terms are interchangeable. Some other people tend to regard SOA as a more abstract concept that it is not bound to any specific technology, while Web services are an instantiation of SOA bound with W3C specifications SOAP (W3C, 2004a) and WSDL (W3C, 2004b). What people agree on is the roles and operations in the SOA/WS triangle (see Figure 1). The SOA/WS triangle represents the common principle of SOA and Web services. The service providers first publish their service descriptions into a registration server. The service requestors can look up the registration to choose suitable services. Then the service requestors can directly interact with the service providers by sending messages to them. The service description can also be stored by the service providers other than stored in a registration server.

Indeed Web services are the only available implementation of SOA, the discussion in this chapter is based on the current Web service techniques. A Web service is an autonomous, well-defined, standards-based component that can be accessed via established Web-based protocols (Arkin, Askary, Fordin, & et.al., 2002). Web services not

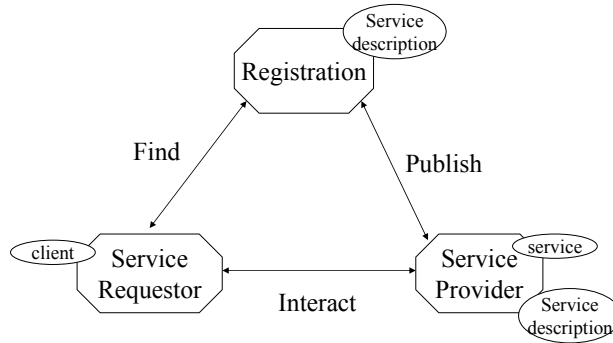


Figure 1: The SOA/WS triangle

only are used as middleware for application integration and invocation due to its interoperability, but also have the capacity of business process modeling and management.

Web service processes are business processes composed by individual Web services. W3C and OASIS have released several XML-based description languages to model Web service processes. These languages can be classified into orchestration languages and choreography languages. **Orchestration** languages hold the point of view of a service requestor (c.f. BPEL4WS). They model how the requestor calls external services and how to process the response internally. In another word, orchestration languages model the *internal* behavior within a service requestor. Complementarily, **choreography** languages model how the services interact with each other outside of the services. The behavior of a Web service is strictly bounded in the sense that it is owned and accessible within the corporation, but hidden from any other corporation. Therefore, the internal behavior of a Web service is not observable for another, except its emitting messages and its communication ports. The choreography languages describe the *observable* behaviors between a group of Web services. It can be from the individual service point of view that the global model is projected on a single service (c.f. WSCI), or from the global system point of view (c.f. WS-CDL). Both choreography and orchestration languages describe the *data flow* and the *control flow* of a business process. Control flow expresses the execution order of the actions in constructs of sequence, branching, parallel, synchronization etc. Data flow is about process relevant data and how these data are interchanged and manipulated by the actions. The two flows are intertwined such that the control flow is triggered by certain status of data and data is manipulated by the control flow.

As Web services interact with each other by passing messages to trigger certain behaviour, Web service processes can be considered to be *discrete event systems* whose logical *states* are the current values of the *variables*. A process composed by Web services can change its state by executing an *action*, such as an assignment of variables, receiving or emitting messages. The emitting messages can be a triggering *event* for another service to take an action. Discrete event system models, especially process algebra and Petri nets, are the original formal models used to design some of the Web services process description languages, while the resulted Web services process description languages have expression power beyond class formal models. This chapter introduces the Web service process description languages and the discrete event system models for modeling and computing Web service processes. This chapter also discusses the methods for model verification and validation. In another dimension, the Web service process description languages are influenced by workflow modeling techniques. Some other modeling tools from software engineering, e.g. UML activity diagrams, can also be used for Web service process modeling. Therefore, the second objective of this chapter is to determine the connections between Web service process modeling and the techniques from workflow modeling and UML activity diagrams.

## 2 Web Service Choreography and Orchestration Languages

This section introduces the basic knowledge about Web services and Web service (WS) process description languages.

### 2.1 The Technical Stack of Web Services

The technical stack in Figure 2 covers the supporting techniques for Web services. The transportation layer uses Internet protocols, such as HTTP, SMTP, FTP and BEEP. XML is the supporting technique for the messaging layer. SOAP messages are XML formatted and XML-RPC is the way to invoke a remote operation. The interoperability of Web services is based on the assumption that these Internet protocols and XML parsers are available for any commonly used platforms. WSDL is exploited at the interface layer to indicate the end point of a service and to describe the operations of a service and the types and number of parameters for invoking an operation. UDDI is a service discovery protocol that supports service registration and look-up. The process description layer contains several languages for describing the process of how the services interact with each other for a task. Process description languages are investigated in this paper.

WS process description languages can be classified as orchestration languages and choreography languages. *Orchestration* languages, such as BPEL4WS and BPML, describe an executable process inside a coordinator. The coordinator is the service requester for all the other services and the coordinator transfers communication between services. This process is the internal behavior of this coordinator and is owned by the coordinator. *Choreography languages*, such as WSCI and WS-CDL, describe a

complex task via the definition of the conversation that should be undertaken by each service. The process is the *observable* behaviors of the business process. It can be from the global system point of view (c.f. WS-CDL) that describes the interactions among several services, or from the individual service point of view, that the global model is projected on a single service (c.f. WSCI). Since a process in choreography language is not executable, sometimes it is called an *abstract process*. The following two subsections use some sample languages to demonstrate how the **data flow** and the **control flow** of a business process are described in these languages.

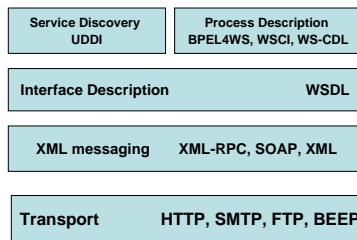


Figure 2: The technical stack for Web services

## 2.2 Orchestration Languages

**BPEL4WS** (Business Process Execution Language for Web services, BPEL for short) (Andrews, Curbera, Dholakia, Goland, & et.al., 2003) is supported by a consortium with BEA, IBM, Microsoft and others and is recognized by OASIS as an e-business standard. It is currently the most widely supported language. BPEL has two ancestors: XLANG from Microsoft, which is a  $\pi$ -calculus-based model, and WSFL from IBM, which is a Petri nets-based model. The two styles can still be found in BPEL that sometimes give alternative solutions to model one process (Wohed, Aalst, Dumas, & Hofstede, 2002).

The basic element in the BPEL process is called an *activity*. An activity is either a primitive or a structured activity. The set of *primitive activities* contains: *invoke*, invoking an operation of some Web service; *receive*, waiting for a message from an external source; *reply*, replying to an external source; *wait*, waiting for some time; *assign*, copying data from one place to another; *throw*, indicating errors in the execution; *terminate*, terminating the entire service instance; and *empty*, doing nothing.

To enable the presentation of complex constructs, the following *structured activities* are defined: *sequence*, for defining an execution order; *switch*, for conditional routing; *while*, for looping; *pick*, for race conditions based on timing or external triggers; *flow*, for parallel routing; and *scope*, for grouping activities to be treated by the same fault handler. Structured activities can be nested and combined in arbitrary ways. Within the scope of *flow*, the execution order can further be controlled by

links. The activity at the source end of a link will be executed before the activity at the target end of a link. Triggering conditions can be associated with a link. When an activity is at the target ends of multiple links, joint conditions can be associated with the activity.

Error handling in BPEL is done by fault handlers and compensation handlers. A compensation handler associates with a scope of activities and is available for invocation only when the scope completes normally. Within a compensation handler, a set of activities are predefined to reverse the effects of the activities in the scope. A fault handler also associates with a scope of activities. The difference is that a fault handler is invoked when the scope cannot complete normally and exceptions are thrown. A fault handler executes predefined activities to recover the business process. The recovery actions can be the invocation of compensation handlers, informing business partners, charging the responsible business partner a penalty for failure to complete an activity, or simply rethrowing the exceptions. One can see that a compensation handler is solely to reverse the effects of activities, while a fault handler can define more complicated business logic including to invoke compensation handlers.

*Variables* are the elements for BPEL data flow. Variables in BPEL are actually the SOAP messages defined in WSDL. Therefore, the variables in BPEL are compound objects that contain several properties. Each property corresponds to a part of the SOAP message defined in WSDL. Variables and their properties are used in the triggering and joint conditions of links, in activity assign, and in correlation definition. A set of correlation tokens is defined as a set of properties shared by the messages referring the same correlation set. For example, client id can be in a correlation set. Any messages referring to this correlation set will use the same client id.

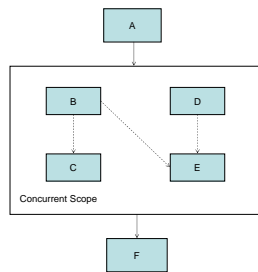


Figure 3: A business process with six activities

**Example 1.** Figure 3 represents a process with six activities. The first activity A is a receive activity. A concurrent scope flow follows A, containing four invoke activities B, C, D, E. Because links (dashed arrow lines) modify the concurrent execution order, the actual execution order would be C after B and E after D; E also has to wait until B finishes. The last activity F is a reply. The BPEL process is presented below in a simplified format. The underlined letters, e.g. a and y, are compound

variables. In particular, each of a to f represents a tuple of {partnerlink, portType, operation}. They are the information about the services. v is the received message by A, and w is the replied message by F. b-in and b-out, so as the other -ins and -outs, are input and output messages for B to E respectively. <corr. > is short for correlation. p is a correlation set.

```

<sequence >
  <receive a v ><corr. >p <\corr. ><\receive >
  <flow >
    <invoke b b-in b-out >
      <source linkname=b.to.c >
      <source linkname= b.to.e >
    <\invoke >
    <invoke c c-in c-out >
      <target linkname=b.to.c >
    <\invoke >
    <invoke d d-in d-out >
      <source linkname=d.to.e >
    <\invoke >
    <invoke e e-in e-out >
      <target linkname=b.to.e >
      <target linkname= d.to.e >
    <\invoke >
  <\flow >
  <reply f w ><corr. >p <\corr. ><\reply >
<\sequence >

```

**BPML** (Business Process Modeling Language) (Shapiro, 2002) is a standard developed and promoted by BPMI.org (the Business Process Management Initiative). BPMI.org was supported by several organizations, including Intalio, SAP, Sun, and Versata. BPMI.org recently merged with OMG. BPML is a rival of BPEL for expressing executable business processes. The main ingredients of BPML are: activities, processes, contexts, properties, and signals. *Activities* are components performing specific functions. *Simple activities* are atomic while *complex activities* are decomposed into smaller activities. A detailed list of these activities is similar to WSCI in the next section for the two specifications are all from BPMI.org. A *process* is a complex activity which can be invoked by other processes. Processes can be nested. Two types of special processes are exception processes and compensation processes. *Context* in BPML defines an environment for the execution of related activities. The context can be used to exchange information and coordinate execution. A context contains local definitions that only apply within the scope of that context. Local definitions include properties, processes, and signals, among others. A *property* definition has a name and a type, while each property instance has a value in the range of the type of the corresponding definition (similar to *properties* of variables in BPEL). *Signals* are used to coordinate the execution of activities executing within a common context other than through basic routing constructs such as sequence (similar to *links* in BPEL). One can think of signals as messages, and there are constructs to send such a message (raise signal) and wait for a message (synchronize signal).

## 2.3 Choreography Languages

**WSCI** (Web Service Choreography Interface) (Arkin et al., 2002) was submitted in 2002 to the W3C by BEA Systems, BPMI.org, Commerce One, Fujitsu Limited, Intalio, IONA, Oracle Corporation, SAP AG, SeeBeyond Technology Corporation, and Sun Microsystems. WSCI aims to describe the flow of messages exchanged by a Web service participating in choreographed interactions with other services. In contrast to BPEL, WSCI describes the observable behavior of a Web service. WSCI does not address the definition and the implementation of the internal processes that actually drive the message exchange, while BPEL describes the internal actions which are not observable from outside. WSDL and WSCI both concern the observable interfaces of a service. WSDL is a static interface that does not describe the execution orders of operations, nor does it describe correlation aspects. In contrast, WSCI describes the execution orders of the actions. The actions are indeed defined in WSDL as operations.

`Action` is the basic atomic activity mapping to an operation in WSDL. Some atomic activities are: `delay`, expressing a time interval; `empty`, doing nothing; and `fault`, triggering a fault. Other atomic activities are for `process` (a kind of complex activity): `call`, instantiating a process and waiting for it to complete; `spawn`, instantiating a process; and `join`, waiting for instances of the spawned process to complete. `Process` is a basic complex activity containing other activities. Other complex activities define control flow structures: `all`, for parallel activities; `choice`, for reacting to different messages and exception events; `foreach`, for iteration for each item of a list; `sequence`, for sequential order actions; `switch`, for conditional routing; `until`, for iteration evaluating the truth value of a condition after a loop; and `while`, for iteration evaluating the truth value of a condition before a loop. WSCI defines the message event handler, timeout event handler and fault handlers. The message event handler can be used inside `choice` and the other two can be used inside activities. WSCI defines transaction scopes and compensations for rolling back failed transactions. WSCI does not have anything equivalent to `Link` in BPEL.

Data flow of WSCI has some similar features as BPEL. WSCI does not use variables such as BPEL to reference to SOAP messages. Instead, WSCI uses `selectors` to refer to the parts of messages whose mechanism is based on XPath. Correlation is the same as in BPEL that associates the properties from the different messages. Unique to WSCI is its global model. The WSCI global model connects the operation that generates a message and the operation that consumes the message. It shows globally the “conversation” paths. It is interesting to point out that WSCI does not contain information of the endpoint of a service.

**Example 2.** Use of WSCI to represent the process in Example 1. Since only the two operations of receiving and replying messages are the observable behaviors, WSCI has only two actions. Here `a` and `f` represent tuples of {action name, role name, portType/operation}.

```
<process >
  <sequence >
    <action a v ><corr. >p <\corr. ><\action >
    <action f w ><corr. >p <\corr. ><\action >
  <\sequence >
```

<\process >

Our example is a simple one. Think about a travel agent who has many operations to deal with, such as clients and airlines. WSCI can define the execution orders of these operations. For example, WSCI can define a loop to deal with a request for several tickets, and WSCI can define that the operation of `Booking_ticket` has to follow the operation of `Checking_availability`.

**WS-CDL** (Web Service Choreography Description Language) (Kavantzias, Burdett, & Ritzinger, 2004) is a choreography language in evolving. The difference between WS-CDL and WSCI is that WS-CDL describes all interdependencies among the different interactions between roles. WS-CDL is a global model of the conversation among services, while WSCI describes the behavior of a single participant when it plays a role within a complex conversation. More specifically, WSCI is a projection of the entire conversation of WS-CDL to a considered participant. One important concept of WS-CDL is the definition of channels. A channel is the information about the endpoint of a service. WS-CDL explicitly points out the possibility to pass the channel information among services. For example, the Buyer can send the channel information to the Seller who then forwards it to the Shipper. The Shipper could then send delivery information directly to the Buyer using the channel information originally supplied by the Buyer. Current business processes are defined in a static way that the business partners and the end points of their services cannot be changed at run time. WS-CDL brings the possibility to select business partners at run time. Dynamic business process composition at run time is a bold promise of Web services. WS-CDL defines three structures of *sequence*, *parallel*, and *choice* for execution ordering. *Variables* in WS-CDL are similar to variables in BPEL. These features are not so significant, compared to other languages. We can see that WS-CDL is still evolving, with more features expected to be added.

**ebXML BPSS** (Business Process Specification Schema) is not a Web service process description language, but a standard to describe business processes from OASIS. Since ebXML adopts WS standards for invoking service too, BPSS is frequently mentioned in WS-related places. If it is compared to WS process description languages, it can be considered as a kind of choreography language that describes how the business partners interact with each other via message passing. BPSS is an e-business specification in that its partners are not limited to Web services.

## 2.4 Relationship to Workflow Management

One of the tasks of workflow management studies business process modeling. Therefore, research in this domain greatly influences WS process modeling. Some people call WS process as workflow. Restrictively, they are not the same. Workflow management comes from Computer Supported Cooperative Work (CSCW). It supports the working environment that contains both computer applications and human beings. Therefore, the activities in Workflow Management Systems (WFMS) are a mixture of automatic activities and human activities. This is very different from Web service systems where all activities are automatic.

Managing this kind of hybrid system is the job of a workflow engine. A workflow

engine routes the workitems (which are the instances of activities in the process templates) to the real actors (which can be human beings or computer applications) for executing the workitems. The workflow engine takes care of the following things: 1) the state of the case (which is an instance of the process); 2) the queue of input events; 3) the case attributes and their values; and 4) the scheduled time-outs and the value of the global clock. The input events are from the environment. Therefore a WFMS is a reactive system that runs in parallel with the environment and responds or reacts to input events by creating certain desirable effects in the environment (Eshuis & Dehnert, 2003). This point of view influences the modeling issues in workflow management. (Eshuis & Dehnert, 2003) proposes two levels to model workflows:

At the *requirement* level, the workflow process contains only business logic. Since workflow processes model the internal behavior of a system, *a workflow process can be seen as an orchestration process at the requirement level*. In general, three important dimensions of workflows are the control flow dimension, the resource dimension and the data dimension. The control flow dimension concerns the ordering of activities, which is equivalent to the control flow in WS processes. The resource dimension concerns the system and database resources and human resources for activities. Many resources in workflow, such as systems and applications, are viewed as services from a WS point of view. That is exactly the concept of Service Oriented Architecture. All the WS process languages mentioned above do not model resources. The so called WS-Resource Framework is actually a specification to manage persistent data. The data dimension is workflow-relevant data. They can be considered as data in WS processes. XPDL (XML Process Definition Language) (WfMC, 2005) is an XML-based description language for workflow process definition. It describes a workflow process from the above three dimensions, so it appears similar to WS process languages. XPDL is not in the scope of this paper.

At the *implementation* level, the behaviors of WFMS itself, e.g. how to start a task, how to waiting for a task to complete, and how to rout the cases, are modeled. The WSFM behavior model is very important for implementing a Workflow engine. This research reflects the point of view that a WFMS system is a reactive system. In WS research, not much attention is on how to implement a WS process engine, though it could be implemented similarly as a workflow engine. It also means that current WS engines are not as sophisticated as workflow engines.

## **2.5 The Expressive Power of Web Services Process Description Languages**

From subsections 2.2 and 2.3, one can see that the representations of control flow and data flow in both orchestration and choreography languages are quite similar. The two orchestration languages, BPEL and BPML, have link to modify the executing order of activities, giving the two languages more flexible expressive power. Despite of this, since all the description languages can model most of the workflow patterns (see below), we consider that *the expressive powers of WS description languages are roughly compatible*.

W.M.P van der Aalst proposed 20 control flow patterns in workflow (Aalst, Hof-

stede, Kiepuszewski, & Barros, 2003). They are considered to cover most of the scenarios in business processes. We can see the influence of this study on WS process modeling. The study of mapping workflow patterns to WS process languages, such as BPEL, BPML and WSCI etc., shows that WS process languages can express most of the important workflow patterns (Aalst, Arthur, Hofstede, & Wohed, 2005) (Wohed et al., 2002). Some patterns are not directly supported by WS process languages. Some of the non-supported patterns have workarounds. Others seem not to be the concern of WS processes. For example, the patterns of multiple instances merge are not expressible by WS process languages. But multiple instances are not in the focus of WS process languages for now. Therefore, the WS process description languages are regarded to have nearly equivalent expressive power as Workflow patterns in control flow.

For data flow, most WS process languages model SOAP messages as *variables*. Variables contain properties for the parts of the messages. Correlation is used to associate different variables or properties. Most WS process languages can manipulate the values of the properties. WS process languages are considered to provide fairly sufficient representation and manipulation operations for data flow.

Although some of the WS process description languages claim to be rooted in the formal models, for example, XLANG and WS-CDL in process algebra, and WSFL in Petri nets, the WS process description languages are beyond what the classic formal models can express. The classic models need to be extended to model all the features in the control flow and data flow. Generally all the formal methods can model control flow easily. But the following features in the WS process description languages need to pay special attentions:

1)**Links** in BPEL change the execution order of activities. The activity at the target end of the link has to wait until the activity at the source end finishes. When an activity is the target end of several links, joint conditions are applied. Petri nets can model links easily, but process algebra cannot. This is because the communication defined in process algebra is synchronized, which means the source end activity cannot evolve until the event is consumed by the target end activity. Petri nets can model this by adding a place to contain a token. Actually, this defines asynchronized communication. See the next section for more details.

2)**Data and data manipulation** means variable should be included. Variable assignment and variable correlation are the ways to manipulate the variables. Variables are also used in conditions to trigger activities.

3)**Endpoint** can be a part of data flow. We want to specially point out its importance in reconfiguration tasks because one important feature of reconfiguration is to change partners at run time. If endpoints are included in the model and rewriting rules are defined, it is possible to reason about reconfiguration tasks. The future study of reconfiguration is expected be undertaken by both industry and academia.

### 3 The Discrete Event System Models for Web Service Process Modeling

WS processes can be considered as *discrete event systems* in that the actions occur at some moment in time, and different actions are separated in time. Among all the models for discrete event systems, automata, process algebra and Petri nets are the most popular to model WS processes.

Process algebra and Petri nets are popular because both are designed for reasoning about *concurrency*. Petri nets have an intuitive graphical representation, which is easy to understand for non-experts. A Petri net model describes both the states and the actions of the system under consideration. There are many analysis techniques available to investigate properties about the states as well as the dynamic behavior of a Petri-net model. Process algebra is purely symbolic formalism which is particularly well suited for computer manipulation. A process algebraic description of a system focuses on its dynamic behavior. It usually has no explicit representation of system states.  $\pi$ -calculus, a variant of process algebra, is especially attractive because it is one of the models to model communication channels, which implies the possibility of changing partners at run time (c.f. subsection 3.1). Proof techniques in process algebra are generally aimed at showing the equality of behavioral descriptions. Consequently, process algebra is useful to compare behavioral descriptions of concurrent systems. A comparison of process algebra and Petri nets can be found in (T. Baeten, 1998). Automata are also used to model WS processes. Though classic automata do not model concurrency, new developments extend classic models for concurrency or avoid computing global states, therefore making it suitable to use automata in WS process modeling.

Figure 4 shows the expressive power of formal models. It represents a popular view. The techniques developed for one model can be used in other models, which blurs the boundaries between the models. For example, (T. Baeten, 1998) combines Petri nets and process algebra where Petri nets are used to model system components and process algebra is used to specify and verify the behavior of these components.

Model	Compositionality	Parallelism	Resources
PA (CCS)	✓	✓	✗
$\pi$	✓	✓	✓
PN	✗	✓	✓
Automaton	✓	✗	✗

Figure 4: The expressive power of formal models.

This paper discusses how these models are used for WS process modeling rather

than presenting research on formal models. It needs to be pointed out that, although some of the WS process description languages claim to be rooted in the formal models, for example XLANG and WS-CDL in process algebra, and WSFL in Petri nets, the present WS process description languages are very powerful and go beyond what the classic formal models can express. This paper seeks to determine the expression boundaries of the formal models within the scope of Web services description languages. The expressive power is about describing control flow and data flow. This paper is not concerned with time and probability issues in the formal models because currently, the WS process description languages do not cover these features. The sections below show how these formal models can model the features of WS process description languages. It does not mean the more features a method covers, the better this method would be. Depending on the computing tasks, a model just needs to cover necessary and sufficient features. This paper does not aim to select a best method, but rather to show the capacities of different methods. Generally speaking, control flow constructs are easy to model, meaning the mapping is straight forward. The discussion below focuses on the difficulties listed in subsection 2.5.

### 3.1 Process Algebra

With process algebras, a complex system is defined by a set of equations. The Calculus of Communication Systems (CCS) (Milner, 1989) defines processes in the form below:

**Definition 1** *The processes are given by*

$$P ::= 0 \mid a.P \mid P + P' \mid P \parallel_S P' \mid P \setminus \{a\} \mid P[f] \mid !P$$

The following are the intended interpretation of the definition:

- the *inaction*  $0$ : a process that does nothing;
- the *prefix*  $a.P$ : the process  $P$  cannot proceed until action  $a$  is completed.  $a$  is in action set  $Act = \{a, b, c, \dots\} \cup \{\tau\}$ .  $\tau$  is non-observable action. The basic actions in CCS are to *receive* a message (this is noted by simply writing its name) or to *emit* a message (this is written by its name prefixed by the quote symbol, e.g.  $'a$ ).
- *nondeterministic choice*  $P + P'$ :  $P$  and  $P'$  are mutually exclusive, only one of them can proceed;
- *parallel composition*  $P \parallel_S P'$ :  $P$  and  $P'$  are parallel, they can proceed independently and can interact via actions  $S$ . By varying the synchronization set  $S$ , parallel composition ranges from complete parallelism ( $\parallel_\phi$ ) to full synchrony ( $\parallel_{Act}$ ).
- *restriction*  $P \setminus \{a\}$ : action  $a$  is within the scope of  $P$ . The external environment observes a  $\tau$  action.
- $P[f]$  behaves like  $P$ , but with the actions relabeled by the function  $f : Act \rightarrow Act$ .

- *replication*  $!P$ : an infinite composition of  $P|P|...$

The definitions of operational semantics are omitted. Other well-known process algebras are the Communication Sequential Processes (CSP) (Hoare, 1985) and the Algebra of Communicating Processes (ACP) (J. Baeten, 2005). Extensions of the basic model include data, time and stochastic features.  $\pi$ -calculus is one extension of process algebra.  $\pi$ -calculus is used to model the processes that are mobile (in the sense that configuration of communication links is dynamic). For Web services,  $\pi$ -calculus is attractive because it models the channels and is able to model configuration of channels through communication, which means it is possible to change partners by forwarding channel information through communication. The actions of  $\pi$ -calculus are given by the following definition.

**Definition 2** *Actions in  $\pi$ -calculus:*

$\alpha ::= c\langle x \rangle$	<i>send the name <math>x</math> via the channel <math>c</math></i>
$c(x)$	<i>receive any name via channel <math>c</math>, binding the name received to <math>x</math></i>
$\tau$	<i>unobservable action</i>

The operators of  $\pi$ -calculus are the same as process algebra, except the restriction rule is sometimes written as  $(\nu x)P$ , instead of  $P \setminus \{x\}$ . Channels in Web services are information about endpoints. In Petri nets, the graph is static, which means there is no way to change a process at execution time.

### 3.1.1 Process Algebra for Web Service Process Modeling

Process algebra is used to model both choreography and orchestration languages. People usually start with notations of a basic process algebra language, such as CCS, ACP or CSP, and define some customized operations to map to some activities in the Web service process description languages and then define the operational semantics, or in another words, rewriting rules, to manipulate the control flow and data flow.

Viroli uses process algebra to model BPEL (Viroli, 2004). This work is so far the most complete mapping towards all the features of BPEL. At the syntax level, he introduces meta-variables that contain a series of variables or properties. This makes the formalization simple and clean. He introduces opaque identifier to model non-deterministic values (that will be given at execution time). Basic activity  $\langle \text{reply } \underline{l} \ \underline{v} \rangle$  and the correlation set within its scope  $\langle \text{corr. } \underline{p} \rangle$  is mapped to construct  $\text{send}(\bar{l}, \bar{v}, \bar{p})$ <sup>1</sup>. Similar mappings are defined for receive and assign. Structural activities sequence and flow are mapped to standard operators “;” and “||”.  $\langle \text{while condition}=\underline{e} \rangle \ \underline{a} \ \langle \backslash \text{while} \rangle$  is mapped to  $\text{while}(e) \ \{a\}$ . Similar mappings are defined for switch and pick. For manipulating data, a term  $\bar{w} \mapsto \bar{u}$  is defined to show variable or property  $w_i$  is associated to value  $u_i$ . For example,

<sup>1</sup>in (Viroli, 2004), it is  $\text{send}(l, \bar{v}, \bar{w})$ . We prefer to use  $\bar{l}$ .

$\bar{w} \mapsto \langle 180, false, 3 \rangle$  means values of 180, *false* and 3 are assigned to  $w_1, w_2, w_3$  respectively. Operational semantics is similar to standard process algebra with extension of variables. For example, the rewriting rule of `flow` is as following:

$$(\bar{w} \mapsto \bar{u})(s_0 \parallel s_1) \xrightarrow{\alpha} (\bar{w}' \mapsto \bar{u}')(s'_0 \parallel s_1) \text{ if } (\bar{w} \mapsto \bar{u})s_0 \xrightarrow{\alpha} (\bar{w}' \mapsto \bar{u}')s'_0$$

`link` is modeled by two operations  $source(\lambda)$  and  $target(\lambda)$ , with  $\lambda$  is the name of the link. The rewriting rule for `link` is:

$$(\bar{w} \mapsto \bar{u})(source(\lambda); s \parallel target(\lambda); s') \xrightarrow{\tau} (\bar{w} \mapsto \bar{u})(s \parallel s')$$

**Example 3.** Use of Viroli's method (Viroli, 2004) to represent the process in Example 1.

```

recv( $\bar{a}, \bar{v}, \bar{p}$ );
{ { invoke( $\bar{b}, \bar{b}_{in}, \bar{b}_{out}$ ); ((source(b_to_c); target(b_to_c);
invoke( $\bar{c}, \bar{c}_{in}, \bar{c}_{out}$ )) || source(b_to_e)) } ||
{ ((invoke( $\bar{d}, \bar{d}_{in}, \bar{d}_{out}$ ); source(d_to_e);
target(d_to_e)) || target(b_to_e));
invoke( $\bar{e}, \bar{e}_{in}, \bar{e}_{out}$ )) } };
send( $\bar{f}, \bar{w}, \bar{p}$ )

```

Because `links b_to_c` and `d_to_e` define basic sequence order, we remove the two links from the above formula and show only the function of link `b_to_e`:

```

recv( $\bar{a}, \bar{v}, \bar{p}$ );
{ { invoke( $\bar{b}, \bar{b}_{in}, \bar{b}_{out}$ ); (
invoke( $\bar{c}, \bar{c}_{in}, \bar{c}_{out}$ )) || source(b_to_e)) } ||
{ (invoke( $\bar{d}, \bar{d}_{in}, \bar{d}_{out}$ )) || target(b_to_e));
invoke( $\bar{e}, \bar{e}_{in}, \bar{e}_{out}$ )) } };
send( $\bar{f}, \bar{w}, \bar{p}$ )

```

In (Viroli, 2004), two activities,  $send(\bar{c}, \bar{c}_{in})$  and  $recv(\bar{c}, \bar{c}_{out})$ , are used for  $invoke(\bar{c}, \bar{c}_{in}, \bar{c}_{out})$ . Here we use  $invoke$  to make the presentation more brief.

Viroli does not model the triggering conditions for the *source* of a link and the joint conditions for the *targets* of multiple links. Since these conditions are critical to model arbitrary logic for control flow, it is better to model these conditions. Here is one solution to model the conditions. *Source* is extended to  $source(\lambda, f)$ , where  $f$  is a boolean expression in the sense of XPath. The activity that is the target of multiple links needs to model the joint condition. For example,  $invoke(\bar{e}, \bar{e}_{in}, \bar{e}_{out})$  in the last example can be extended to  $invoke(\bar{e}, \bar{e}_{in}, \bar{e}_{out}, \lambda_{b\_to\_e}, \lambda_{d\_to\_e}, f)$ . The changes to the rewriting rules are not discussed in this paper due to the limit of length of this paper.

There is some other work that uses process algebra. (Salaün, Bordeaux, & Schaerf, 2004) discusses the possibilities of using process algebra to model both orchestration and choreography processes. The answer is positive. At the end of the paper, the author discusses the mapping between CSS operators and BPEL activities. This mapping is rough in that it does not consider features such as correlation, links, variables, and triggering conditions.

Ferrara maps BPEL into LOTOS (Ferrara, 2004), because LOTOS allows one to model data. The features of the control flow and data flow are well modeled by LOTOS

language. It is interesting to see that `links` are modeled as dummy actions. As an example, we use Ferrara's method to model the control flow and `link b_to_e` of example 1. The other two links are ignored because they represent trivial sequence order. A dummy activity  $G$  is added to represent the link `b_to_e`. Then the control flow of example 1 is:  $A; \{(B; (G||C))||((D||G); E)\}; F$ . We can see the equivalence of the execution trace by removing the effect of the dummy actions. This work does not model endpoints. This work also skips correlation.

(Brogi, Canal, Pimentel, & Vallecillo, 2004) uses process algebra to model WSCI. The translation is quite straightforward. (Busi, Gorrieri, Guidi, Lucchi, & Zavattaro, 2005) proposes a formalization for a new choreography language. Actually this formalization can be used for WS-CDL.

Right now, most of work using process algebra, such as (Salaün et al., 2004) and (Viroli, 2004), do not use  $\pi$ -calculus. But if the endpoint is modeled, it is possible to reason about changing partners. It would be a future direction.

There are supporting tools designed for process algebra for the following tasks (Ferrara, 2004):

**Control flow verification:**

- temporal logic model checking to prove properties of services: liveness, safety, and request-response (a request is always satisfied, also for infinite behaviors). If the property is not satisfied, a counterexample is returned.
- bisimulation, to check whether the behaviors of two services or two versions of the same service are equivalent.
- simulation, to check whether the behavior of a service is included with the behavior of other interacting services.
- execution traces of the service (manually or randomly guided), to understand the behavior of the service.

**Data flow verification:**

- data type checking, in the case of LOTOS and other process algebras allowing data handling.
- black box testing: for a class of input values, some properties are satisfied.

**Reconfiguration modeling:** Compositionality is a feature of Web services. Composition at run time means choosing partners at run time and/or changing the business logic of the processes. The models to support this have to include endpoints and variable manipulation capacity. This is a promising direction for future study. It seems that though  $\pi$ -calculus is a natural model for reconfiguration, a model which models endpoints and variable manipulation can also do the job.

**Process monitoring and fault diagnosis** is a new demand for Web service process management. It is important to know the causes of a failed business process and identify the responsible business partners. Process algebra, as well as the other formal methods below, is used for system monitoring and fault diagnosis. The principle of

diagnosis can be found in (Hamscher, Console, & de Kleer, 1992). Roughly, a system is considered to be composed by components. The diagnosis task is to find which components are responsible for observed system behavior which has discrepancy to the expected right behavior. (Console, Picardi, & Ribaud, 2002) uses process algebra for diagnosis. After the right system behavior, as well as known faulty behavior, is modeled in process algebra, all the possible traces of actions that are consistent to the observations are found out. If some faulty actions are in the traces, they are the diagnosed faults. Monitoring mechanism needs to be developed to record the necessary information for fault diagnosis. The condition of diagnosing faults is that sufficient information is recorded. The study of monitoring mechanism is to determine which kind of information is necessary for which kind of faults (also called diagnosability analysis).

Process algebra are also used in some tasks that are unique for Web service processes. For example, (Breugel & Mariya, 2005) analyzes Dead-Path-Elimination in BPEL with process algebra, and (Haddad, Melliti, Moreaux, & Rampacek, 2004) verifies the ambiguity (non usability) of a Web service behavior.

## 3.2 Petri Nets

P/T net is a simplest model in Petri nets and is the most commonly used model in business process modeling.

**Definition 3** A Petri Net (*P/T Net*) is a triple  $(P, T, F)$ , where

- $P$  is a finite set of places,
- $T$  is a finite set of transitions,  $(P \cap T = \emptyset)$
- $F \subseteq (P \times T) \cup (T \times P)$  is a finite set of arcs, the flow relations.

(Aalst et al., 2003) uses Petri nets to study workflow patterns. (Aalst et al., 2003) shows Petri nets can easily model the control flows. The standard way to incorporate data into Petri nets is to use colored tokens. Colored tokens are tokens that have attributed values. These attributed values are modified by transitions. Another way is to interpret places as predicates, but then instances of the predicates can be seen as tokens that can change values when a transition consumes them (Eshuis & Wieringa, 2003). So in both approaches, tokens carry data. In practice, sometimes people associate places with variables and triggering rules with transitions to model data flow.

**Example 4.** Using Petri nets to model the control flow of the process in example 1 can be as simple as in Figure 5. The boxes are the transitions of the Petri net, which represent the activities in BPEL. The `link` is very easy to model using Petri nets. In Figure 5, the transition `S` is a dummy transition that is not mapped to any activities in BPEL, but is used to join the two parallel branches.

(Ouyang et al., 2005) models control flow of BPEL with Petri nets and presents formal semantics for the control flow structures. (Ouyang et al., 2005) claims to be the most complete mapping that covers all control flow structures, `links` and event handlers. (Ouyang et al., 2005) considers “skip” behavior of a basic activity. “Skip”

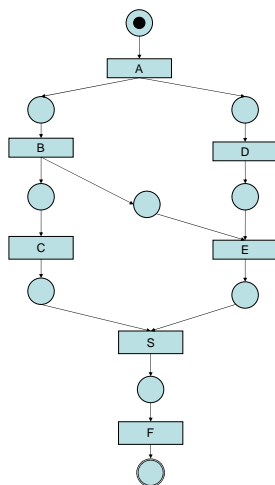


Figure 5: The Petri net for the control of the process in Figure 3.

behavior is that when an activity cannot be executed because the joint conditions of the links pointing to it are not satisfied, this activity passes a value to the links leaving from it so that the process can continue. It is so called “dead-path elimination”. With the “skip” behavior, a basic activity  $X$  is as complex as in Figure 6. The normal processing of  $X$  is drawn in solid lines, and the skip behaviors are drawn in dashed lines. In the normal processing part, place  $r_X$  (“ready”) models an initial state when it is ready to start activity  $X$  before checking the status of all control links coming into  $X$ , and place  $f_X$  (“finished”) indicates a final state when both  $X$  completes and the status of all control links leaving from  $X$  have been determined. The transition labeled  $X$  models the action to be performed. Transition  $X$  has an input place  $s_X$  (“started”) for the state when activity  $X$  has started, and an output place  $c_X$  (“completed”) for the state when  $X$  is completed. Two transitions are drawn as solid bars model internal actions for checking pre-conditions or evaluating post-conditions for activities. The skip path is at the left side. Note that the “to skip” and “skipped” places are respectively decorated by two patterns (a letter  $Y$  and its upside-down image) so that they can be graphically identified.

(Ouyang et al., 2005) does not model data flow, but it includes propositional variables for boolean conditions. An automatic tool has been developed to detect unreachable activities and detect conflicts of parallel activities. (Ouyang et al., 2005) does not say how the analysis tool is implemented, supposing state space analysis (c.f. below) can do this.

(Schmidt & Stahl, 2004) presents another semantics based on Petri nets for BPEL. (Schmidt & Stahl, 2004) models almost all features of BPEL. (Schmidt & Stahl, 2004) considers activity cancellation and failure behaviors within a basic activity. Therefore,

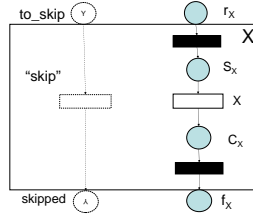


Figure 6: The basic activity as in (Ouyang et al., 2005).

the model of a basic activity is complicated. Data are modeled as objects that have to feed in the transitions. Correlations are also modeled as objects to be used by the transitions. The activities connected with links have to include the behavior of links. This work does not model endpoints.

Simulation and verification are two methods used for Petri nets analysis. Simulation is a method to analyze Petri Nets. It works in a similar way to program testing and, unless the system is trivial, cannot be used to prove or verify properties about the behavior of the process. State space analysis can verify many properties for Petri nets. The basic idea behind state space is to construct a directed graph which has a node for each reachable marking and an arc for each occurring binding element. State spaces are also called occurrence graphs or reachability graphs/trees. From state analysis, properties such as liveness, reachability and fairness can be verified. The semantics of Petri nets can be expressed in a process algebra expression (T. Baeten, 1998). Therefore, the analysis methods used for process algebra, such as model checking and bisimulation mentioned in the above subsection, can be applied for Petri nets.

Some papers use Petri nets for composition and verification tasks (Mecella, Presicce, & Pernici, 2002) (Hamadi & Benatallah, 2003), though they do not map any WS process languages into Petri nets.

Petri nets are also used for fault diagnosis (Boubour, Jard, Aghasaryan, Fabre, & Benveniste, 1997). The method is to unfold the trajectory of system evolution on observations. If a faulty state is in the trajectory, this fault is a diagnosis.

### 3.3 Automata

Finite automata are also called finite state machines ((Sipser, 1997), p31). An automaton has a number of *states* and a number of *transitions*, going from one state to another. A transition denotes the execution of an action. Besides, there is an initial state (there can be more than one) and a number of final states ((Sipser, 1997), p35):

**Definition 4** A finite automaton  $\Gamma$  is a 5-tuple  $\Gamma = (X, \Sigma, T, I, F)$ , where:

- $X$  is a finite set of states;
- $\Sigma$  is a finite set of events;
- $T \subseteq X \times \Sigma \times X$  is a finite set of transitions;

- $I \subseteq X$  is a finite set of initial states;
- $F \subseteq X$  is a finite set of final states.

A behavior is a run, i.e. a path from initial state to final state. Classic automata do not model *concurrency*. It is the study of concurrency which leads to the birth of Petri nets and process algebra. When modeling the interactions of several systems, automata have the space explosion problem caused by enumerating all the reachable states of the composed system. But as a basic model, some reasoning techniques for Petri nets and process algebra are based on automata. For example, it is a foundation for a model-checker to compute the expression language and reachability. Recent studies explore the capacity of using automata for concurrent systems. The concurrent behavior is modeled as pieces of synchronizing automata. If computation of the synchronization can be delayed until the final results have to be presented, the state explosion problem can be walked around (Pencolé & Cordier, 2005). *Concurrent automata* (Deussen, 1998) label the concurrent behaviors on the transitions which gives a way to express concurrent behavior with automata.

The advantages of using automata in modeling WS processes are that: 1) they can easily include actions, events, variables and rules into the basic model; 2) they are clearer than Petri nets and process algebra for representing the system states and flows; and 3) rich techniques are automata-based for various computing tasks.

Yan *et al.* maps BPEL into automata (Yan, Pencolé, Cordier, & Grastien, 2005). The control flows can be easily mapped to different structures of automata. Concurrent branches in `flow` are modeled as pieces of synchronizing automata. To represent data flow, state variables are defined and mapped to variables in BPEL. In addition, transition rules containing state variables are defined to model the triggering conditions in control flow. (Yan et al., 2005) does not show how to model `links`. Since (Yan et al., 2005) uses pieces of synchronizing automata, `links` can be modeled as synchronizing events. It shows automata have the full expressive power for WS process modeling. The objective of (Yan et al., 2005) is to monitor and diagnose WS processes modeled as automata.

**Example 5.** Use of Yan *et al.*'s method (Yan et al., 2005) to model the process in example 1. The nodes represent the states and the arcs represent the activities. In Figure 7, the nodes on the lines are duplicated copies of a state. For example, "Start\_flow" has three duplicated copies, so do "Start\_link" and "End\_flow". The lines are the borders of synchronizing scope. The synchronizing events are the labels in the square boxes. This process is composed of several pieces of automata with sequential activities.

(Fisteus, Fernández, & Kloos, 2004) (Fu, Bultan, & Su, 2004) and (Foster, Uchitel, Magee, & Kramer, 2003) use automata to model BPEL for verification. (Fisteus et al., 2004) considers cancelation and fault handling mechanisms inside a basic activity. Therefore a basic activity looks like an automaton with several states. (Fisteus et al., 2004) does not say how to handle concurrency and `link` for control flow. Neither is data flow being modeled. The graphical model is translated into a script language. The script language file is fed into a SPIN model checker. Currently it can verify invariants, reachability, and transition pre- and post-conditions, etc.

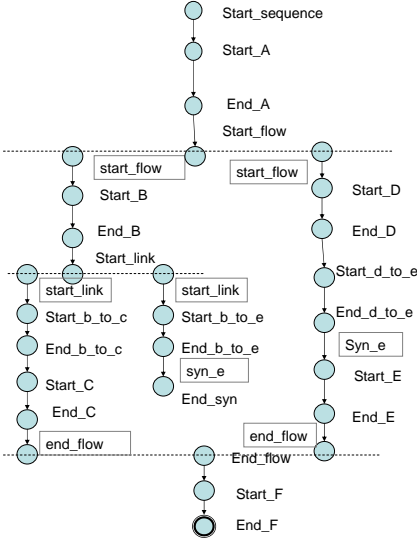


Figure 7: The pieces of synchronizing automata for the process in Figure 3

(Fu et al., 2004) models a WS process  $p$  as a guarded automaton  $(M^{in}, M^{out}, L, T, s, F, \Delta)$ , where  $M^{in}$  ( $M^{out}$ ) are incoming (outgoing) message types for  $p$ ,  $L$  is the set of local variables for  $p$ , and  $T, s, F$  are the set of states, the initial state, and the set of final states, respectively.  $\Delta$  is the set of transitions. Each transition  $\tau \in \Delta$  has a source state  $q_1 \in T$  and a destination state  $q_2 \in T$ . For example, a receive transition is  $\tau = (q_1, ?a, q_2)$ , where  $a \in M^{in}$  (? is to show the message  $a$  is an incoming message). (Fu et al., 2004) does not say how to model correlation and `link`. A BPEL process is translated into the script language Promela and sent to a SPIN model checker. (Fu et al., 2004) studies the synchronous property for communication for a composite Web service, i.e. either the communication is synchronous or with bounded queues.

(Foster et al., 2003) translates BPEL into a script language Finite State Process (FSP). FSP has a graphic representation like automaton and a symbolic syntax similar to process algebra. The translation is very straightforward. It does not show the mapping of correlation, data and `link`. The LTSA model checker is used to verify if BPEL implementation is consistent with specifications.

As mentioned above, automata are used for Web service process monitoring and diagnosis (Yan et al., 2005). Because it is a nature way to model system behavior, especially dynamic behavior, using automata, there are rich literatures on using automata for monitoring and diagnosing systems (Baroni, Lamperti, Pogliano, & Zanella, 1999)(Pencol  & Cordier, 2005). The major method is trajectory unfolding. When the system has concurrent actions, such as Web service processes, state explosion has to be avoided by some methods mentioned above.

## 4 Other Models and Related Studies

**Abstract State Machine** (ASM) specifies a pseudo-code program in which the variables and functions are built of terms. The transitions in ASM obey to inductive rules, i.e the variables and functions of states are built by applying rules on previous states. The rigorous mathematical definition of ASM is not within the scope of this paper. (Roozbeh, Uwe, & Mona, 2005) and (Fahland, 2004) provide a clear and complete formal representation of BPEL. (Fahland, 2004) follows the work in (Roozbeh et al., 2005) and covers more complete features of BPEL than (Roozbeh et al., 2005). Both works use the notion of agent in order to describe the semantics of each activity in BPEL. An agent is described by its own abstract state machine and its interface (allowing agent composition). Both works differentiate between the concept of an activity (generic interface) and their instances involved in process construction, thus allowing them to easily model the notion of process instances. `Correlation` and `link` are also handled using, in natural way, the precondition of the rules (acting as guards in the ASM transition system). The power of the ASM is that it represents different aspects of the language (functional, operational and general requirements) by using the same mechanism, but the work efficiency depends strongly on the function interpretation. Moreover, it is difficult to perform model checking with such models (Winter, 2000).

Though the following two models do not belong to formal models, they are related to WS process modeling.

**UML activity diagrams** can be used to model business processes. Actually, they are used in workflow modeling (Dumas & Hofstede, 1998). Activity diagrams are graphical. Rectangles and ovals are used to represent states. A bar represents a fork (more than one outgoing edge) or a join (more than one incoming edge). A diamond represents a choice (more than one outgoing edge) or a merging of different choices (more than one incoming edge). A WS process can be represented by an activity diagram.

**BPMN** (Business Process Modeling Notation) from BPMI.org is also a graphical diagram to model business processes. The symbols and the expressive power of BPMN are very similar to UML activity diagrams. There are studies that use BPMN to model workflow patterns (White, 2005a) and map BPMN processes to BPEL (White, 2005b).

## 5 Discussions and Conclusions

The notion of composability is essential to the success of any distributed formalism in this Internet age. Web service process description languages are script languages used by software developers for modeling business processes composed by Web services. Due to the lack of formal semantics for these languages (their semantics are defined using English prose), validation and reasoning about their process behaviors cannot be performed directly with these languages. The principle approach discussed in this paper is to translate a process described in a WS process description language into a mathematically well-founded model.

For **the mapping tasks**, the most supported and updated WS process languages - namely BPEL, WSCI and WS-CDL - have been translated into Petri nets, process

algebra and automata. BPEL is the most studied language, which means most current computing tasks are for orchestration. Though not all the combinations of the description languages and formal models exist up to now, it is not difficult to provide one. It is possible to extend classic formal models to cover the features of the control flow and data flow of the WS process description languages. Even very specific features such as `link` and `correlation` can be modeled. It needs more work on validating the mappings to see if the formal models are equivalent to the semantics described in WS process languages. There is some discussion that process algebra cannot handle certain control flow (Aalst, 2005). The example in (Aalst, 2005) is similar to the semantics of `link` in BPEL. In this paper, we see people have developed several solutions with process algebra. Whether or not they are truly equivalent to the semantics described in WS process languages is not within the scope of this paper. Although some existing works cover more features of WS process languages than others, it does not mean that the more features that are included in a model, the better the model is. People just need to include necessary and sufficient information for the computing tasks. Therefore, what to model depends on the computing tasks (see below).

Various **computing tasks** concern different information in the models. People study more the properties of control flow. For example, model checking considers mainly the properties of control flow, such as liveness and safety. Bisimulation is also a technique at the control flow level for verifying the equivalence of behaviors of two processes. We think that data validation is especially interesting for Web services because most errors that cause a process to stop in the middle are due to data type mismatching and data format invalidation. Some tasks concern both control flow and data flow, for example, insuring a correct implementation of an orchestration execution engine. We can see that the integration of verification, validation, monitoring and diagnosis, and reconfiguration leads to the concept of self-manageable Web service processes. Self-manageable Web service processes can automatically compose Web services to meet the business goals and verify the properties of the processes. When the process fails, the monitoring and diagnosing mechanisms can be triggered to find out responsible business partners, replace them and continue the process to fulfill its commitments. We believe that building self-manageable Web services is the ultimate goal for future study.

## Acknowledgement

Prof. Dr. Philippe Dague at Laboratory of Informatics in the University of Paris 11 (Orsay) reviewed the manuscript and gave suggestions of improvement previous to submission of this paper.

## References

Aalst, W. van der. (2005). Pi calculus versus petri nets: Let us eat humble pie rather than further inflate the pi hype. *BPTrends*.

- Aalst, W. van der, Arthur, M. D., Hofstede, H. ter, & Wohed, P. (2005). *Pattern based analysis of bpml (and wsci)*. (QUT Technical report, FIT-TR-2002-05, Queensland University of Technology, Brisbane, [citeseer.ist.psu.edu/554490.html](http://citeseer.ist.psu.edu/554490.html))
- Aalst, W. van der, Hofstede, A. ter, Kiepuszewski, B., & Barros, A. (2003). Workflow patterns. *Distributed and Parallel Databases*, 14(1), 5-51.
- Andrews, T., Curbera, F., Dholakia, H., Golland, Y., & et.al. (2003). *Business process execution language for web services (BPEL4WS) 1.1*. (<ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf>, accessed on April 10, 2005)
- Arkin, A., Askary, S., Fordin, S., & et.al. (2002). *Web service choreography interface (WSCI) 1.0*. (available at [www.w3.org/TR/wsci](http://www.w3.org/TR/wsci), accessed on April 10, 2005)
- Baeten, J. (2005). A brief history of process algebra. *Theoretical Computer Science*, 335(2-3), 131-146.
- Baeten, T. (1998). In terms of nets: System design with petri net and process algebra. *Ph.D. thesis at Eindhoven University, the Netherland*.
- Baroni, P., Lamperti, G., Pogliano, P., & Zanella, M. (1999). Diagnosis of large active syste. *Artificial Intelligence*, 110(1), 135-183.
- Booth, D., Haas, H., McCabe, F., & et.al. (2004). *Web service architecture*. (available at [www.w3.org/TR/ws-arch/](http://www.w3.org/TR/ws-arch/), accessed on April 10, 2005)
- Boubour, R., Jard, C., Aghasaryan, A., Fabre, E., & Benveniste, A. (1997). A petri net approach to fault detection and diagnosis in distributed systems. In *Proc. of the 36th conference on decision and control* (p. 720-731).
- Breugel, F. van, & Mariya, K. (2005). Dead-path-elimination in bpel4ws. In *5th international conference on application of concurrency to system design* (p. 192-201). St Malo, France.
- Brogi, A., Canal, C., Pimentel, E., & Vallecillo, A. (2004). Formalizing web service choreographies. In *Electronic notes in theoretical computer science* (p. 73-94).
- Busi, N., Gorrieri, R., Guidi, C., Lucchi, R., & Zavattaro, G. (2005). Towards a formal framework for choreography. In *Proc. of international workshop on distributed and mobile collaboration (dmc 2005)*, *iee computer society press* (p. 107-112).
- Console, L., Picardi, C., & Ribauda, M. (2002). Process algebras for systems diagnosis. *Artificial Intelligence*, 142(1), 19-51.
- Deussen, P. (1998). Algorithmic aspects of concurrent automata. In H.-D. Burkhard, L. Czaja, & P. Starke (Eds.), *Proc. workshop on concurrency, specification and programming* (p. 39-70). Informatik-Bericht Nr. 110, Humbolt Univ. at Berlin.
- Dumas, M., & Hofstede, A. ter. (1998). UML activity diagrams as a workflow specification language. In M. Gogolla & C. Kobryn (Eds.), *Proc. of the uml'2001 conference* (p. 76-90). Springer.
- Eshuis, R., & Dehnert, J. (2003). Reactive petri net for workflow modeling. In W. van der Aalst & E. Best (Eds.), *Proc. 24th international conference on the applications and theory of petri nets (icatpn)* (p. 296-315). Springer.
- Eshuis, R., & Wieringa, R. (2003). Comparing petri net and activity diagram variants for workflow modelling - quest for reactive petri nets. In H. Ehrig, W. Reisig, G. Rozenberg, & H. Weber (Eds.), *Petri net technology for communication based systems* (p. 321-351). Springer.
- Fahland, D. (2004). *Complete abstract operational semantics for the web service busi-*

- ness process execution language (Technical Report No. SFU-CMPT-TR-2004-03). Humboldt-Universität zu Berlin, Institut für Informatik.
- Ferrara, A. (2004). Web services: a process algebra approach. In *Proceedings of the 2nd international conference on service oriented computing (icsoc)* (p. 242-251). New York, NY, USA: ACM Press.
- Fisteus, J., Fernández, L., & Kloos, C. (2004). Formal verification of bpel4ws business collaborations. In K. Bauknecht, M. Bichler, & B. Prll (Eds.), *Proc. of 5th international conference e-commerce and web technologies (ec-web)* (p. 76-85). Springer.
- Foster, H., Uchitel, S., Magee, J., & Kramer, J. (2003). Model-based verification of web service compositions. In *Proc. of eighteenth ieee international conference on automated software engineering (ase03)* (p. 152-161).
- Fu, X., Bultan, T., & Su, J. (2004). Analysis of interacting bpel web services. In *Proc. of the 13th international world wide web conference (www'04)*. ACM Press.
- Haddad, S., Melliti, T., Moreaux, P., & Rampacek, S. (2004). Modelling web services interoperability. In *Proceedings of the sixth international conference on enterprise information systems* (p. 287-295).
- Hamadi, R., & Benatallah, B. (2003). A petri net-based model for web service composition. In *Proceedings of the fourteenth australian database conference (adc)*.
- Hamscher, W., Console, L., & de Kleer, J. (Eds.). (1992). *Readings in model-based diagnosis*. Morgan Kaufmann.
- Hoare, C. A. R. (1985). *Communicating sequential processes*. Prentice Hall.
- Kavantzias, N., Burdett, D., & Ritzinger, G. (2004). *Web services choreography description language (ws-cdl) 1.0*. (available at [www.w3.org/TR/2004/WD-ws-cdl-10-20040427/](http://www.w3.org/TR/2004/WD-ws-cdl-10-20040427/), accessed on April 10, 2005)
- Mecella, M., Presicce, F., & Pernici, B. (2002). Modeling e-service orchestration through petri nets. In A. B. et al (Ed.), *Technology for e-services (tes)* (p. 38-47).
- Milner, R. (1989). *Communication and concurrency*. Prentice Hall. (International Series in Computer Science)
- Ouyang, C., Aalst, W. van der, Breutel, S., Dumas, M., Hofstede, A. ter, & Verbeek, H. (2005). *Formal semantics and analysis of control flow in ws-bpel* (Tech. Rep.). BPM Center Report BPM-05-13. (BPMcenter.org)
- Pencolé, Y., & Cordier, M.-O. (2005). A formal framework for the decentralised diagnosis of large scale discrete event systems and its application to telecommunication networks. *Artificial Intelligence Journal*, 164(1-2), 121-170.
- Roosbeh, F., Uwe, G., & Mona, V. (2005). *Abstract operational semantics of the business process execution language for web services* (Technical Report No. 11-10046740). Simon Fraser University, Burnaby B.C. Canada.
- Salaün, G., Bordeaux, L., & Schaerf, M. (2004). Describing and reasoning on web services using process algebra. In *Proc. of 2nd inter. conf. on web services (icws04)*.
- Schmidt, K., & Stahl, C. (2004). A petri net semantic for bpel: validation and application. In *Proc. of 11th workshop on algorithms and tools for petri nets (awpn 04)* (p. 1-6).
- Shapiro, R. (2002). *A comparison of xpdl, bpml and bpel4ws*. (available at [xml.coverpages.org/Shapiro-XPDL.pdf](http://xml.coverpages.org/Shapiro-XPDL.pdf), accessed on April 10, 2005)

- Sipser, M. (1997). *Introduction to the theory of computation*. PWS publishing company.
- Viroli, M. (2004). Towards a formal foundation to orchestration languages. In *Electronic notes in theoretical computer science 105* (p. 51-71). Elsevier.
- W3C. (2004a). *Soap specification*. (available at [www.w3.org/TR/soap12-part1/](http://www.w3.org/TR/soap12-part1/), accessed on April 10, 2005)
- W3C. (2004b). *WSDL specification*. (available at [www.w3.org/TR/wsdl](http://www.w3.org/TR/wsdl), accessed on April 10, 2005)
- WfMC. (2005). *Xml process definition language (xpdL) 2.0*. (available at [www.wfmc.org/standards/XPDL.htm](http://www.wfmc.org/standards/XPDL.htm), accessed on April 10, 2005)
- White, S. (2005a). *Process modeling notations and workflow patterns*. (available at [www.bpmn.org/](http://www.bpmn.org/), accessed on April 10, 2005)
- White, S. (2005b). *Using bpmn to model a bpel process*. (available at [www.bpmn.org/](http://www.bpmn.org/), accessed on April 10, 2005)
- Winter, K. (2000, March). Towards a methodology for model checking asm: Lessons learned from the flash case study. In S. F. I. of Technology (ETH) Zurich (Ed.), *International workshop on abstract state machines*. Monte Verita, Switzerland.
- Wohed, P., Aalst, W. van der, Dumas, M., & Hofstede, A. H. ter. (2002). *Pattern based analysis of bpel4ws*. ([citeseer.ist.psu.edu/556822.html](http://citeseer.ist.psu.edu/556822.html))
- Yan, Y., Pencolé, Y., Cordier, M.-O., & Grastien, A. (2005). Monitoring web service networks in a model-based approach. In *3rd IEEE European conference on web services (ecows05)*. Växjö, Sweden: IEEE Computer Society.