# MODELING BPEL WEB SERVICES FOR DIAGNOSIS: TOWARDS SELF-HEALING WEB SERVICES

Yingmin Li, Tarek Melliti and Philippe Dague
*LRI, Univ. Paris-Sud, CNRS, Parc Club Orsay Université, 4 rue Jacques Monod, bât G, Orsay, F-91893, France*

Keywords: Web service, diagnosis, BPEL, modeling, Petri net.

Abstract: An approach generating automatically the data dependency diagrams of the orchestrated complex Web services is presented. The method is derived from the Model-Based Reasoning paradigm, whose origin comes from Artificial Intelligence applied to engineered systems. It is achieved by modeling BPEL activities by Petri nets, enriched to represent data dependencies, and by proposing aggregation rules of these dependency relations. The algorithm to aggregate the basic enriched Petri nets, producing the data dependency diagram of the orchestrated Web service, is given. The model obtained can be directly exploited by a diagnosis algorithm.

## 1 INTRODUCTION

Generally speaking, a Web service is a software system designed to support interoperable machine-to-machine interaction over a network (Booth et al., 2004). Large distributed Web services systems for complex functions can be built, based on basic Web services. A Web service can thus be basic or composite, and the composite ones are made up of the basic ones. The components (basic or composite Web services) of Web services applications communicate each other with messages. All the inputs, results, and errors are encapsulated inside the messages and circulate between components. Composite Web services are built according to two kinds of structure: orchestrated or choreographed (Rosario et al., 2006). The main difference between both lies in the paths that the messages follow between the components. In an orchestration, a central process (which can be a Web service) takes the control of the involved Web services and coordinates the execution of their operations. So the message path is up-to-down. Only the central coordinator is aware of the process and execution order of its components, which it invokes by sending messages to them. In contrast, in a choreography, which is founded on a collaboration effort focusing on the exchange of messages in a public business process, there is no central coordinator and all the involved Web services need to be aware of the process, the operations to execute, the messages to exchange and their timing. So we can see that a Web services application is a data (messages) oriented system.

As Web services are software components they may dysfunction: some faults cannot be detected immediately and may provoke a dysfunction further in the composition. Thus it is important to enrich Web services framework in order to be able to trace the source of faults. Many faults can happen within a Web service: in the network, the data bases, the program, etc. Here we focus on data semantic faults (e.g., a fault caused by different interpretations of a date format: 06/03/2006 will be interpreted in English as June, 3, 2006, but in French as March, 6, 2006). To enrich Web services framework, we need a method to detect and explain such kind of faults, in a word to diagnose dysfunctions. There exists several diagnosis approaches like learning-based, model-based, etc. Compared to the learning based approaches, the model-based one presents the advantage in the context of Web services to be able to detect more effectively the unanticipated or hidden faults like semantic faults. As modeling is the most important step towards model-based diagnosis, we propose in the following a method to automatically generate a model for orchestrated composite Web services in order to diagnose data semantic faults.

## 2 THE CONTEXT: BASIC BPEL ACTIVITIES

BPEL (Business Process Execution Language) is a language for the formal specification of composite Web services, especially the orchestrated ones (Andrews et al., 2003). A BPEL process specifies the ex-



Li Y., Melliti T. and Dague P. (2007).
MODELING BPEL WEB SERVICES FOR DIAGNOSIS: TOWARDS SELF-HEALING WEB SERVICES.
In *Proceedings of the Third International Conference on Web Information Systems and Technologies - Internet Technology*, pages 297-304
DOI: 10.5220/0001290902970304

act order in which participating Web services should be invoked, either sequentially or in parallel. One can define loops, declare variables, copy and assign values, define fault handlers, and set conditions to control the process flow. So with all these constructions, one can define complex composite Web services to implement complicated business process controls.

In a typical scenario, the BPEL business process receives a request. To fulfill it, the process invokes the involved Web services and then responds to the original caller (Weerawarana and Curbera, 2002). A BPEL process consists of steps and each step is called an "activity".

BPEL is the starting point for modeling composite Web services. No matter they are orchestrated or choreographed, their components are BPEL services and have to be modeled first. So, we have to model the following primitive operations defined by BPEL:

- $Invoke(o,X,Y)$, that invokes another Web service operation $o$, taking the value of the variable $X$ as input and storing the output in the variable Y. Note that $type(X) = input(o)$ and $type(Y) = output(o)$.
- $Receive(o,X)$, that receives the input message of operation $o$, storing its value in the variable $X$.
- $Reply(o,X)$, that sends the response to the invoker of the BPEL process, storing the result in the variable $X$.
- $Assign(X,Y)$, that stores the value of the source variable X into the target variable Y. Note that the source can be a value.

To simplify the model, we do not develop the time activity (*wait*), the exception raising (*throw*), or the empty activity (*empty*).

BPEL supports also structured activities. So, to define the complex algorithms that specify exactly the business processes steps, the primitive or structured activities ($S_i$) are combined as follows:

- $Sequence(S_1,S_2)$ defines a set of activities that will be invoked in an ordered sequence.
- $Flow(\{S_i\}_{i\in I})$ defines a set of activities that will be invoked in parallel.
- $Switch(\{c_i(\overline{X_i},\overline{V_i}),S_i\}_{i\in I})$ defines a case-switch construction for implementing branching execution guarded by conditions $c_i$ defined over the variables and values vectors $\overline{X_i}$ and $\overline{V_i}$.
- $While(\{c(\overline{X},\overline{V}),S_1\})$ defines a loop execution of the activity $S_1$ guarded by the condition $c$.

The implementation of an operation $o$ takes place as a subpart of the BPEL code delimited by the corresponding *receive* activity and the associated *reply* activities.

## 2.1 The Approach

We want to model BPEL services for diagnosis. The basic Web services codes being unknown for outside except for the designers, we follow the work of (Ardissono et al., 2005b)(Ardissono et al., 2005a), that introduces, for modeling these services, three types of relations between their input and output messages:

- FW, when the output just forwards the value of an input message.
- SRC, when the service is the source of the output which is thus independent of the input messages.
- EL, when the output is elaborated from input messages.

In orchestrated BPEL services, the notion of operation plays a central role in the composition process and, unlike the basic services, the operations code is known for all. So we propose a method to model automatically the BPEL operations for diagnosis, that is, to deduce automatically (this is done by hand in the work quoted above) the data dependency relations (FW, SRC, EL) between the input and output messages of the BPEL services, by analyzing the BPEL code. And then to aggregate the models of the BPEL operations to deduce the data dependency between the receive activity of the operation and all its possible replies. For that, we propose a four-step method to deduce automatically the data dependency from the activity diagram of the operations:

1. Model the BPEL code using Petri nets to capture the possible orders of the activities, which relies on a simplification of the work of (Hamadi and Benatallah, 2003) (in section 3).
2. Enrich Petri nets with data dependency features, according to semantic rules (in section 4).
3. For each BPEL activity, propose a set of rules to deal with the sequential and alternative propagations of the three types of relations: FW, SRC, EL (in section 5).
4. Propose algorithms to aggregate the basic Petri nets and calculate the data dependency diagram (in section 6).

To diagnose a semantic fault of a Web service, we need to track the data dependency in the BPEL process in order to establish the responsibility of the service, that is, we must be able to illustrate how the service processes the data from input to output and, when a fault occurs, to deduce whether the service is the source of the fault. Precisely, the behavior of a basic Web service is captured by the relations (FW,

SRC, EL) that can be used to establish the responsibility of the service. For example, if a FW operation is executed normally, and the output is abnormal, the input must be abnormal. If a SRC operation is executed normally, and the output is abnormal, the operation must be the source of the fault.

To define now the data dependency graph of a composite BPEL service, we need to know the execution order of the activities. This is why we choose to model the BPEL process by using Petri nets.

# 3 MODELING BPEL USING PETRI NETS

## 3.1 Petri Nets

A Petri net is a formalism aimed at modeling discrete event concurrent systems. A Petri net is a bipartite graph (places and transitions).

**Def 1** *A Petri net is a tuple $\langle P,T,F\rangle$ where:*

- *$P$ is a set of labeled places*
- *$T$ is a set of labeled transitions*
- *$F \subseteq (P\times T \cup T\times P)$ is a set of arcs relating input places to transitions and transitions to output places*
- $\forall t \in T, \exists p,q, ((p,t)\in F \wedge (t,q)\in F)$
- $\forall t \in T, \forall p,q, ((p,t)\in F \wedge (t,q)\in F) \rightarrow p \neq q$

We represent by ${}^\bullet x$ and $x^\bullet$ respectively the input and output places or transitions of x:

$$\forall x \in P\cup T \begin{cases} {}^\bullet x = \{y \in P\cup T | (y,x)\in F\} \\ x^\bullet = \{y \in P\cup T | (x,y)\in F\} \end{cases}.$$

To illustrate the dynamics of a Petri net, we define its execution with the notion of marking and a set of rules of marking evolution (transition firing). A marking is a distribution of tokens on places.

**Def 2** *A marking $M$ of a net $N=\langle P,T,F\rangle$ is a subset of $P$, $M\subseteq P$.*

Markings are also called configurations. When we have a net with a marking, we have a net system.

**Def 3** *A net system $S$ is a couple of a net and an initial marking $M_{in}$, $S=\langle P,T,F,M_{in}\rangle$.*

The execution of a net system is based on the transition firing rules.

**Def 4** *Let $S=\langle P,T,F,M_{in}\rangle$ be a net system and $M\subseteq P$ be a marking of $S$. We say that a transition $t$ is enabled in $M$, if ${}^\bullet t \subseteq M$ and $t^\bullet \cap M = \emptyset$.*

When a transition is enabled, it can fire. The firing of a transition is the core concept of the execution of a Petri net.

**Def 5** *Let $S$ be a net system as above. Let $M1$ and $M2$, be two markings of $S$ (i.e., $M1\subseteq P$ and $M2\subseteq P$). We say that the transition $t$ fires from $M1$ to $M2$, if $t$ is enabled in $M1$, and $M2=(M1\setminus {}^\bullet t)\cup t^\bullet$.*

We write $M1 \xrightarrow{t} M2$, when $t$ fires from $M1$ to $M2$.

**Def 6** *We extend the definition 5 to define a sequence of transitions:*

- *$M \xrightarrow{\lambda} M$, if $\lambda$ is the empty sequence*
- *$M \xrightarrow{\omega t} M'$, if $\exists M'', M \xrightarrow{\omega} M'' \wedge M'' \xrightarrow{t} M'$*

For a given net system $S=\langle P,T,F,M_{in}\rangle$, the set $RS(S)=\{M \mid \exists \omega, M_{in} \xrightarrow{\omega} M\}$ is the reachable set of $S$. $FS(S)=\{\omega \mid \exists M, M_{in} \xrightarrow{\omega} M\}$ is the set of transition occurrences sequences. We use also a variant of $FS(S)$, $FS_M(S)=\{\omega \mid M_{in} \xrightarrow{\omega} M\}$, to denote the set of transition occurrences sequences that lead to the marking $M$.

The idea of using Petri nets for modeling BPEL (Hinz et al., 2005) is to use the places to represent data and the transitions to represent activities. Each basic BPEL activity is thus a transition of Petri net, and there are two kinds of places: the data places represent the input and output messages of a BPEL activity and we create the control places, which we call transmission activation places, to represent the activation condition of this activity. The transmission activation place generated by a transition is the activation condition of the next activity. These control places will express the operational semantics of BPEL.

Petri nets show three kinds of advantages: first, provide a formal model to represent the operational code; second, express a notion of causality between places and transitions that will be very useful to extract dependencies between application data and control data; and last, offer a set of properties and associated analysis tools that will facilitate the analysis of the operational code.

## 3.2 Translating Bpel to Petri Net

First, we define the kind of Petri net that will be used to model BPEL processes.

**Def 7** *A BPEL Petri net is a tuple $N=\langle a^{in},a^{out},P,T,C,R\rangle$, where:*

- *$a^{in},a^{out}$ are the input and output activation places of the Petri net*
- *$P$ is the set of labeled (data and control) places*
- *$T$ is the set of labeled transitions*
- *$C\subseteq (P\cup\{a^{in}\})\times T \cup T\times (P\cup\{a^{out}\})$ is the set of normal arcs (solid arcs)*

- $R \subseteq P \times T$ *is the set of reading arcs (dashed arcs) with* $C \cap R = \emptyset$.

And $\forall t \in T$ $\begin{cases} {}^\bullet t = \{y \in P \cup \{a^{in}\} | (y,t) \in C\} \\ {}^\circ t = \{y \in P | (y,t) \in R\} \\ t^\bullet = \{y \in P \cup \{a^{out}\} | (t,y) \in C\} \end{cases}$

Similar to Petri net, a marking in a BPEL Petri net is a distribution of tokens on the places: $M \subseteq P \cup \{a^{in}, a^{out}\}$. A transition $t$ is enabled in $M$ if ${}^\bullet t \cup {}^\circ t \subseteq M$. If a transition $t$ is enabled in $M$, it can fire from $M$ to $M'$ with $M' = (M \setminus {}^\bullet t) \cup t^\bullet$.

The basic activities of the BPEL language consist of the communication activities (*receive*, *reply*), the data manipulation activities (*assign*, *compute*, and *condition* evaluation), and the *invoke* activity. These activities constitute the building blocks of the BPEL constructions and we will give now the BPEL Petri net model of each one.

As our aim in this behaviors modeling is to catch to the maximum the dependency between application data and control data, we decompose the type of the input messages or Xpath expression variables into their elementary parts, denoted by the *leaves* of their tree structure. For $X$ a variable of type $m$ (resp. a Xpath expression), we use $x_i$ to range over $Leaves(m)$ (resp. $Leaves(X)$) and we denote the $x_i$ part of $X$ by the couple $(X, x_i)$.

**Receive:** the receive activity $receive(o, X)$ is the first step of a BPEL service. It is activated when an input message of the operation $o$ is received by the service and the value in this message is assigned to the variable $X$. The corresponding BPEL Petri net is $N = \langle a^{in}, a^{out}, P, T, C, R \rangle$ with:

- $P = \{(X, x_i), (m, x_i)\}_{i \in I}$ and $T = \{t_{receive}\}$
- $C = \{(a^{in}, t_{receive}), (t_{receive}, a^{out})\} \cup \{(t_{receive}, (X, x_i))\}_{i \in I}$
- $R = \{((m, x_i), t_{receive})\}_{i \in I}$

**Reply:** the reply activity $reply(o, X)$ takes as input the operation name $o$ and the variable $X$ that will be used to construct the output response message. The corresponding BPEL Petri net is $N = \langle a^{in}, a^{out}, P, T, C, R \rangle$ with:

- $P = \{(X, x_i), (m, x_i)\}_{i \in I}$ and $T = \{t_{reply}\}$
- $C = \{(a^{in}, t_{reply}), (t_{reply}, a^{out})\} \cup \{(t_{reply}, (m, x_i))\}_{i \in I}$
- $R = \{((X, x_i), t_{reply})\}_{i \in I}$

Note that the variable $X$ places are related to the transition by reading arcs, which means that the activity does not change their values.

**Assign:** the assign activity $assign(X, Y)$ allows the data transmission between local variables. It can be applied to variables if they are type-compatible on parts of their data values. It takes as input the source variable $X$, and assigns the source value to the target variable $Y$. As for the receive and reply activities, the variable $X$ can be either a variable or an Xpath expression of an existing variable. The corresponding BPEL Petri net is $N = \langle a^{in}, a^{out}, P, T, C, R \rangle$ with:

- $P = \{(X, x_i), (Y, y_i)\}_{i \in I}$ and $T = \{t_{assign}\}$
- $C = \{(a^{in}, t_{assign}), (t_{assign}, a^{out})\} \cup \{(t_{assign}, (Y, y_i))\}_{i \in I}$
- $R = \{((X, x_i), t_{assign})\}_{i \in I}$

Note that $x_i \in Leaves(Type(X))$ and $y_i \in Leaves(type(Y))$. The index $I$ is the same here because $type(X) = type(Y)$. Note also that the input places of the source variable are related to the transition by reading arcs, which means that the activity does not change them. In the case of assigning values to variables, $assign(v, Y)$, the $(X, x_i)$ are suppressed and the $(Y, y_i)$ are replaced by $Y$.

**Invoke:** the invoke activity $invoke(o, X, Y)$ is used to call an operation offered by a partner. It takes as input the operation name $o$ and a variable $X$ that $type(X) = input(o)$ and as output a variable $Y$ that $type(Y) = output(o)$. The variable $X$ contains the value of the input message of $o$ and $Y$ receive its response. The operation $o$ is extendable and we suppose that we only know its input-output dependency model $a = (X_a, Y_a, F_a)$, where $X_a$ and $Y_a$ are the input and output vectors and each $f_i$ in $F_a \in \{\text{FW,SRC,EL}\}$. The corresponding BPEL Petri net is $N = \langle a^{in}, a^{out}, P, T, C, R \rangle$ with:

- $P = \{(X, x_i), (Y, y_i)\}_{i \in I}$ and $T = \{t_{invoke}\}$
- $C = \{(a^{in}, t_{invoke}), (t_{invoke}, a^{out})\} \cup \{(t_{invoke}, (Y, y_i))\}_{i \in I}$
- $R = \{((X, x_i), t_{invoke})\}_{i \in I}$

Notice that $x_i \in X_a$ and $y_i \in Y_a$.

The figure 1 illustrates the BPEL Petri net models of the basic activities.

We present now the BPEL Petri net models of the structured activities by composing (in a way similar to (Hamadi and Benatallah, 2003), but simplified) the models of the basic activities they are made up of.

**Sequence operator:** the sequence operator $sequence(S_1, S_2)$ is used to connect different activities, and the execution order of these activities is the same as their appearance order in the constructor. We associate to each activity $S_i$ its BPEL Petri net model $N_i = \langle a_i^{in}, a_i^{out}, P_i, T_i, C_i, R_i \rangle$. The BPEL Petri net of the resulting sequence is $N = \langle a_1^{in}, a_2^{out}, P, T, C, R \rangle$ with:

- $P = P_1 \cup P_2 \cup \{a\}$

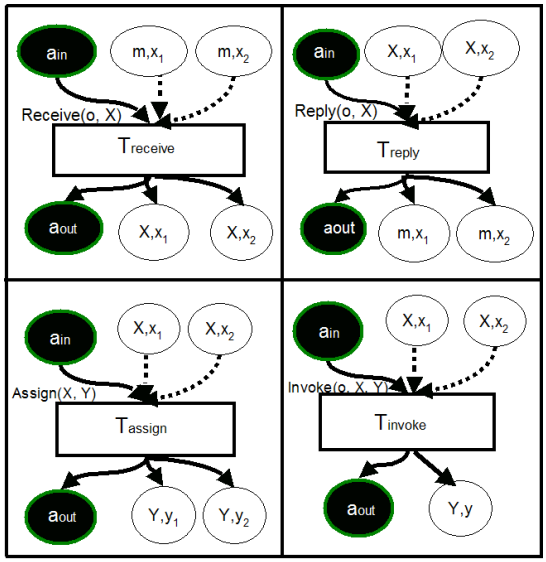


Figure 1: BPEL Petri net models of the basic activities.

- $T = T_1 \cup T_2$
- $C = (C_1 \cup C_2 \cup \{(t_i, a)\}_{t_i \in {}^\bullet a_1^{out}} \cup \{(a, t_j)\}_{t_j \in a_2^{in\bullet}}) \setminus (\{(t_i, a_1^{out})\}_{t_i \in {}^\bullet a_1^{out}} \cup \{(a_2^{in}, t_j)\}_{t_j \in a_2^{in\bullet}})$
- $R = R_1 \cup R_2$

Notice that we introduced an inside activation place $a$ from $T_1$ to $T_2$.

**Conditional operator:** the switch operator $switch(\{(c_i(\overline{X_i}, \overline{V_i}), S_i)\}_{i \in I})$ represents an alternative execution of the activities $S_i$ under the conditions $c_i(\overline{X_i}, \overline{V})$ where $\overline{X_i}$ is the vector of the free variables $x_{ij}$ of the condition and $\overline{V_i}$ is the vector of values $v_{ij}$. Let $N_i = \langle a_i^{in}, a_i^{out}, P_i, T_i, C_i, R_i \rangle$ be the BPEL Petri net model of the activity $S_i$. The BPEL Petri net model of the resulting activity is $N = \langle a^{in}, a^{out}, P, T, C, R \rangle$ with:

- $P = \bigcup_{i \in I} (P_i \cup \{a_i^{in}\} \cup \{x_{ij}\}_j \cup \{v_{ij}\}_j)$
- $T = \bigcup_{i \in I} (T_i \cup \{t_{c_i}\})$
- $C = \bigcup_{i \in I} (C_i \cup \{(a^{in}, t_{c_i}), (t_{c_i}, a_i^{in})\} \cup \{(t_{ij}, a^{out})\}_{t_{ij} \in {}^\bullet a_i^{out}}) \setminus \{(t_{ij}, a_i^{out})\}_{t_{ij} \in {}^\bullet a_i^{out}}$
- $R = \bigcup_{i \in I} (R_i \cup \{(x_{ij}, t_{c_i})\}_j \cup \{(v_{ij}, t_{c_i})\}_j)$

**Iterative operator:** the while operator $while(c(\overline{X}, \overline{V}), S_1)$ represents an activity that iterates the activity $S_1$ execution until the breaking off of the conditions $c(\overline{X})$ (where $\overline{X}$ is a vector of the free variables $x_i$ and $\overline{V}$ is a vector of values $v_i$). Let $N_1 = \langle a_1^{in}, a_1^{out}, P_1, T_1, C_1, R_1 \rangle$ be the BPEL Petri net model of the activity $S_1$. The BPEL Petri net model of the while activity is $N = \langle a^{in}, a_1^{out}, P, T, C, R \rangle$ with:

- $P = P_1 \cup \{a_1^{in}\} \cup \{x_i\}_i \cup \{v_i\}_i$
- $T = T_1 \cup \{t_c, t_{\overline{c}}\}$
- $C = (C_1 \cup \{(a^{in}, t_c), (a^{in}, t_{\overline{c}}), (t_c, a_1^{in}), (t_{\overline{c}}, a_1^{out})\} \cup \{(t_j, a^{in})\}_{t_j \in {}^\bullet a_1^{out}}) \setminus \{(t_j, a_1^{out})\}_{t_j \in {}^\bullet a_1^{out}}$
- $R = R_1 \cup \{(x_i, t_c), (v_i, t_c), (x_i, t_{\overline{c}}), (v_i, t_{\overline{c}})\}_i$

Notice that we introduced $t_c$ to represent the transition if condition $c$ is true and $t_{\overline{c}}$ to represent the transition if condition $c$ is false.

**Parallel operator:** the flow operator $flow(\{S_i\}_{i \in I})$ executes the activities $S_i$ in parallel. It terminates when all the activities are finished (fork-join). Let $N_i = \langle a_i^{in}, a_i^{out}, P_i, T_i, C_i, R_i \rangle$ be the BPEL Petri net model of the activity $S_i$. The BPEL Petri net model of the parallel activity is $N = \langle a^{in}, a^{out}, P, T, C, R \rangle$ with:

- $P = \bigcup_{i \in I} (P_i \cup \{a_i^{in}, a_i^{out}\})$
- $T = \bigcup_{i \in I} T_i \cup \{t^{in}, t^{out}\}$
- $C = \bigcup_{i \in I} C_i \cup \{(a^{in}, t^{in}), (t^{out}, a^{out}\} \cup \{(t^{in}, a_i^{in})\}_{i \in I} \cup \{(a_i^{out}, t^{out})\}_{i \in I}$
- $R = \bigcup_{i \in I} R_i$

Notice that we introduced $t^{in}$ and $t^{out}$ to represent the initial and final transitions of the parallel activity.

See the figure 2 for graphical illustration.

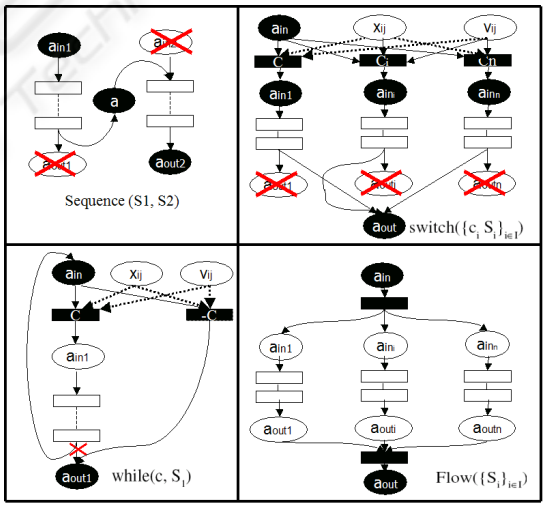


Figure 2: BPEL Petri net models of the control operators.

## 4 ENRICHING BPEL PETRI NET MODEL FOR DATA DEPENDENCY

The semantics of the Petri net model and its marking express the operational dependency between the tran-

sitions executions. But, in addition to this operational dependency, we want to capture the nature of the dependency between data. For all that, the idea is to enrich each transition of the BPEL Petri net with a set of dependency relations (*FW*, *SRC* or *EL*) between its input and output places. This explicit data causality together with the Petri net causality properties provide a rich model to analyze the data dependency in the BPEL process definition. If $P_1$ and $P_2$ are two sets and $F$ is a set of relations labels between elements of $P_1$ and elements of $P_2$ we will denote by $[P_1 \rightarrow P_2]_F$ the set of all possible mappings $f$ from $P_1$ to $P_2$ such that $\forall p \in P_1, f : p \rightarrow f(p) \in F$.

**Def 8** *The extended BPEL Petri net of the net* $N = \langle a^{in}, a^{out}, P, T, C, R\rangle$ *is a tuple* $EN = \langle N, D\rangle$ *with*

- $D : T \rightarrow 2^{[P \rightarrow P]_{\{FW,SRC,EL\}}}$ *such that* $\forall t \in T, D(t) \subseteq [{}^{\circ}t \cup t^{\bullet} \rightarrow t^{\bullet}]_{\{FW,SRC,EL\}}$

*D is a function that associates to each transition of the net a set of FW, SRC and EL functions between the input and output places of the transition.*

In the following, we reconsider the BPEL Petri net model of each BPEL activity and extend it by defining its *D* function.

**Receive:** $receive(o,X)$ receives an input message and forwards the value contained directly to the variable $X$, so:

$D(t_{receive}) = \{FW((m,x_i),(X,x_i)), FW(a^{in},a^{out})\}$.

**Reply:** $reply(o,X)$ just forwards the value of variable $X$ to the output message, so: $D(t_{reply}) = \{FW((X,x_i),(m,x_i)), FW(a^{in},a^{out})\}$. **Assign:** $assign(X,Y)$ forwards the data of the source variable $X$ to the target variable $Y$ and $assign(v,Y)$ gives the value $v$ to the variable $Y$. In the case of assigning a variable to a variable, the assign activity only forwards the variable value and the activation status from input to output. And in the case of assigning a value to a variable, this (variable,value) pair is considered as created by the service. So: $D(t_{assign}) = \begin{cases} \text{assign(X,Y):} & \{FW(X,Y), FW(a^{in},a^{out})\}, \\ \text{assign(v,Y):} & \{SRC(Y), FW(a^{in},a^{out})\}. \end{cases}$

**Invoke:** $invoke(o,X,Y)$ is used to call a partner's operation *o*. *o* is known by its intput-output dependency model $a = (X_a, Y_a, F_a)$. All that we have to do is to extend the invoke transition by this model, by replacing in $F_a$ each $x_i$ by $(X,x_i)$ and each $y_i$ by $(Y,y_i)$:

$D(t_{invoke}) = F_a[x_i/(X,x_i), y_i/(Y,y_i)] \cup \{FW(a^{in},a^{out})\}$.

The extended nets of the structured activities are obtained by composing the extended nets of the activities in parameters.

**Sequence operator:** $sequence(S_1,S_2)$ does no add new transition, and it only unifies the input activation of $S_2$ with the output activation of $S_1$ by a new place labeled $a$. The data dependency relation does not change but we relabel the two unified places by $a$:

$D = D_1[a_1^{out}/a] \cup D_2[a_2^{in}/a]$.

**Conditional operator:** the BPEL Petri net model of $switch(\{(c_i(\overline{X_i},\overline{V_i}),S_i)\}_{i\in I})$ has been obtained by adding a transition $t_{ci}$ for each condition, an input activation place for the resulting choice, a place for each variable (or value) involved in each condition $c_i$, and by unifying the output activations of all the activities $S_i$. The switch operator is activated when the input activation is transmitted to the activity $S_i$ whose condition is evaluated to be true. So it exists an elaboration relation between the variables and values ($x_{ij}$ and $v_{ij}$), which are used for the $c_i$ condition evaluation, and the input activation of $S_i$. Each condition transition is thus extended by an elaboration relation between its input places (input activation, variables and values) and its output place:

$D = \bigcup_{i\in I}(D_i[a_i^{out}/a^{out}] \cup D(t_{ci}))$

with $D(t_{ci}) = \{EL(\{a^{in},x_{ij},v_{ij}\},a_i^{in})\}$.

**Iterative operator:** the BPEL Petri net model of $while(c(\overline{X},\overline{V}),S_1)$ has been obtained by adding two transitions $t_c$ and $t_{\overline{c}}$ for the iteration condition $c$ and its complementary, an input activation place for the resulting choice between them, and a place for each variable (or value) involved in the condition. As for the switch operator, the activation transmission through one or the other transition is elaborated from the variables $x_i$ and values $v_i$ used in the condition:

$D = D_1[a_1^{out}/a^{in}] \cup D(t_c) \cup D(t_{\overline{c}})$,

with $D(t_c) = \{EL(\{a^{in},x_i,v_i\},a_1^{in})\}$

and $D(t_{\overline{c}}) = \{EL(\{a^{in},x_i,v_i\},a_1^{out})\}$.

**Parallel operator:** $flow(\{S_i\}_{i\in I})$ transmits the activation to each of the activities $S_i$. Its BPEL Petri net model contains two additional transitions: one that forwards the parallel activation to each activity and the other that gathers the terminations of each activity for ending the parallel activation. So:

$D = \bigcup_{i\in I}(D_i \cup \{FW(a^{in},a_i^{in})\} \cup \{FW(a_i^{out},a^{out})\})$

With the enriched BPEL Petri nets defined in this section, we can capture the data dependency of the BPEL service operations. But for constructing the diagnosis models for BPEL services, we need to define the propagation rules to aggregate the BPEL service operations together.

# 5 PROPAGATING DATA DEPENDENCIES

As said above, a BPEL process is a partial order execution schema on basic activities. As our aim is to

aggregate the BPEL activities models from the basic ones that compose its code, we are led to compose extended BPEL Petri nets and thus to define how input-output dependency relations are aggregated according to the "sequential", "alternative", or hierarchical (related to the XML description) composition.

Let $a = \langle\{x\},\{y\},F_a\rangle$ and $b = \langle\{z\},\{t\},F_b\rangle$ (where $x,y,z$ and $t$ are four data parameters) be the dependency models of two given activities.

We would like to deduce the dependency model of the composed activity from:

1. the values of $F_a$ and $F_b$,
2. the unification between parameters,
3. and the execution order model of $a$ and $b$.

So, we have to axiomatize how dependency relations $f \in F_a$ and $g \in F_b$ are composed, according to properties such as "transitivity" and "maximality".

**Sequential aggregation rules ($f,g$):** we suppose here a sequential composition $a,b$ of $a$ and $b$ and we consider the transitivity axioms. In table 1, we give the sequential propagation of the three relations $FW, SRC, EL$.

Table 1: Sequential aggregation rules.

| |
|---|
| $FW(x,y)\ FW(z,t)\backslash z=y \equiv FW(x,t)$ |
| $EL(x,y)\ EL(z,t)\backslash z=y \equiv EL(x,t)$ |
| $FW(x,y)\ EL(z,t)\backslash z=y \equiv EL(x,t)$ |
| $EL(x,y)\ FW(z,t)\backslash z=y \equiv EL(x,t)$ |
| $SRC(y)\ FW(z,t)\backslash z=y \equiv SRC(t)$ |
| $SRC(y)\ EL(z,t)\backslash z=y \equiv SRC(t)$ |

**Alternative aggregation rules ($f \oplus g$):** we suppose here an alternative composition of the dependency models in case of a guarded choice execution. If $c = \langle\{z\},\{t\},F_c\rangle$ is a third activity dependency model, we consider the activity model $d = b \oplus c$ where either $b$ or $c$ is executed (e.g. following in sequence the execution of $a$). The table 2 shows the alternative composition rules of the three relations $FW, SRC, EL$ used to express the relation between $z$ and $t$ in the resulted activity $d$.

Table 2: Alternative aggregation rules.

| |
|---|
| $FW_b(z,t) \oplus FW_c(z,t) \equiv FW_d(z,t)$ |
| $EL_b(z,t) \oplus FW_c(z,t) \equiv EL_d(z,t)$ |
| $FW_b(z,t) \oplus EL_c(z,t) \equiv EL_d(z,t)$ |
| $EL_b(z,t) \oplus EL_c(z,t) \equiv EL_d(z,t)$ |
| $SRC_b(t) \oplus SRC_c(t) \equiv SRC_d(t)$ |
| $SRC_b(t) \oplus FW_c(z,t) \equiv EL_d(z,t)$ |
| $SRC_b(t) \oplus EL_c(z,t) \equiv EL_d(z,t)$ |
| $FW_b(z,t) \oplus SRC_c(t) \equiv EL_d(z,t)$ |
| $EL_b(z,t) \oplus SRC_c(t) \equiv EL_d(z,t)$ |

**Hierarchical propagation rules through data structures:** the input and output parameters are XML messages and we need, for diagnosis purpose, to obtain the finest description of the BPEL operations and thus propagation rules at the level of the messages substructures. We introduce thus the following decomposition axioms (a message type being a tree of labels, for any node label $l$ we denote by $Xpath(l)$ the set of all the paths of the subtree of root $l$):

- $FW(l_1,l_2) \equiv \wedge FW(l_i,l_j)$ with $l_i \in Xpath(l_1)$, $l_j \in Xpath(l_2)$ and $l_i \approx l_j$, which means that a forward relation between two data structures gives rise to forward relations between all the equivalent (same type) substructures.
- $SRC(l) \equiv \wedge SRC(l_j)$ with $l_j \in Xpath(l)$, which means that each time a data structure is created, all its substructures are also created.
- $EL(l_1,l_2) \equiv \wedge EL(l_1,l_j)$ with $l_j \in Xpath(l_2)$, which means that if a data structure is elaborated from another structure, each of its substructures is also elaborated from this structure.

# 6 IMPLEMENTING THE BPEL DIAGNOSIS MODEL

In section 3 and 4, we presented a Petri net modeling of a BPEL operation and in section 5, we proposed data dependencies aggregation rules. We can now define an algorithm that builds the diagnosis model of a BPEL operation from its extended Petri net. The idea of the algorithm is based on the following steps:

- Generate all the traces of the Petri net based on an initial marking and a valid final marking.
- Apply the sequential aggregation rules to each trace.
- Apply the alternative aggregation rules to the obtained set of aggregated traces.
- Apply the hierarchical propagation rules to data in order to optimize the operation description.

Let $o[m_1,m_2]$ be a BPEL operation modeled by the extended BPEL Petri net $N = \langle a^{in}, a^{out}, P, T, C, R, D\rangle$. Let $M_{in} = \{a^{in},(m_1,x_i),v_j\}$ and $M = \{a^{out},(m_1,x_i),v_j\}$ be the initial and final markings of $N$. The set of transitions sequences $FS_M(S)$, where $S$ is the net system $\langle N, M_{in}\rangle$, represents all the execution paths of $N$ from $M_{in}$ to $M$. Note that, according to the translating rules, the Petri net of the BPEL code is finite, but its execution can be infinite. But such infinite execution can be expressed using a regular expression. In fact, in our

case, due to the properties of aggregation rules, each infinite sequence is equivalent to a finite set of finite sequences. More precisely, taking into account that any data dependency relation $F \in \{FW, SRC, EL\}$ verifies $F^2 \equiv F$, we have the equivalences:
$F^* \equiv \{\varepsilon, F\}$ ($\varepsilon$ is the empty sequence) and $F^+ \equiv \{F\}$.

The diagnosis model of $o$, $R \subseteq [{}^\bullet t_{receive} \cup^\circ t_{receive} \rightarrow \bigcup_i t^\bullet_{reply_i}]_{\{FW,SRC,EL\}}$, is thus given by the algorithm:

**Alg 1** *Input S: BPEL net system of o, M: final marking. Output R: set of dependency relations*
*RF: list of sets of dependency relations*
$RF = \emptyset$
*for each* $\sigma_i = (t_{i1}, t_{i2}, ..., t_{ik_i}) \in FS_M(S)$ *do*
  $R = D(t_{i1})$;
  *for j=2 to* $k_i$ *do*
    $R = \circ_{Agg}(R, D(t_{ij}))$
  *endfor;*
  $RF = RF \cup \{R\}$;
*endfor;*
*while* ($|RF| > 1$) *do*
  $RF[1] = \oplus_{Agg}(RF[1], RF[2])$;
  *remove(RF,RF[2]);*
*endwhile;*
$R = RF[1]$;

## 7 CONCLUSION

In this paper, following model-based diagnosis approach, we used Petri nets to construct the diagnosis model of a BPEL service. By analyzing its BPEL code, we decomposed the BPEL process into basic operations (receive, reply, assign, and invoke) and operators (sequence, switch, while, flow). Each generic basic operation was then modeled by an extended BPEL Petri net (normal Petri net enriched by activation places and reading arcs and extended by data dependency relations attached to each transition) and each operator by an extended BPEL Petri net constructor, leading to an automatic extended BPEL Petri net modeling of the process. Finally, we gave the propagation rules for data dependencies corresponding to several composition modes (sequential, alternative, hierarchical through data structures) and the algorithm for calculating the diagnosis model of a BPEL service. Such a model can now be used as input to model-based diagnosis algorithms, such as the one proposed by (Ardissono et al., 2005a).

Further work will be carried out along two steps.

First, achieving decentralized model-based diagnosis for composite Web services. All our current work is in a centralized environment, but in reality, decentralization and distribution are the trends of software application. Centralized diagnosis is not adequate for decentralized composite Web services. Some progress was done in this direction by (Ardissono et al., 2005a), but limited to orchestrated Web services with a diagnosis supervisor.

Second, realizing auto-repairable and auto-reconfigurable Web services. For that, we need to define first the repair and reconfiguration rules for basic Web service activities (operations or operators), and then the actions planner and scheduler for the repair and reconfiguration supervisor of the Web services (centralized or decentralized).

## REFERENCES

Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Thatte, S., Trickovic, I., and Weerawarana, S. (2003). Business process execution language for web services, version 1.1. BEA Systems, IBM Corp., Microsoft Corp., SAP AG, Siebel Systems. http://www-128.ibm.com/developerworks/library/specification/ws-bpel.

Ardissono, L., Console, L., Goy, A., Petrone, G., Picardi, C., Segnan, M., and Dupré, D. T. (2005a). Cooperative model-based diagnosis of web services. Proc. of IFIP/IEEE Int. Workshop on Self-Managed Systems Services (SELFMAN 2005), Nice, France.

Ardissono, L., Console, L., Goy, A., Petrone, G., Picardi, C., Segnan, M., and Dupré, D. T. (2005b). Enhancing web services with diagnostic capabilities. Proc. of European Conference on Web Services (ECOWS-05), pp. 182-191, Vaxjo, Sweden, IEEE.

Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., and Orchard, D. (2004). Web services architecture, technical report, w3c. Technical report. http://www.w3.org/TR/ws-arch/.

Hamadi, R. and Benatallah, B. (2003). A petri net-based model for web service composition. Proc. of the 14th Australasian database conference, Adelaide, Australia, ACM.

Hinz, S., Schmidt, H., and Stahl, C. (2005). Trasforminf bpel to petri nets. Proc. of the 3rd Int. Conference on Business Process Management (BPM 2005), Nancy, France, LNCS 3649, pages 220-235.

Rosario, S., Benviniste, A., and S. Haar, C. J. (2006). Net systems semantics of web services orchestrations modeled in orc. Research report, IRISA, 1780.

Weerawarana, S. and Curbera, F. (2002). Business process with bpel4ws: Understanding bpel4ws, part 1. Technical report. http://www-128.ibm.com/developerworks/webservices/library/ws-bpelcol1/.