University of Castilla-La Mancha

A publication of the

**Department of Computer Science**

# Automatic Translation from WS-CDL to Prioritized-Timed Colored Petri Nets by using WST Tool.

by

María Díaz Valentín Valero Hermenegilda Macià Juan José Pardo
María Emilia Cambronero Gregorio Díaz.

Technical Report **#DIAB-01-01-14** November 2011

DEPARTAMENTO DE SISTEMAS INFORMÁTICOS
ESCUELA SUPERIOR DE INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE CASTILLA-LA MANCHA
Campus Universitario s/n
Albacete - 02071 - Spain
Phone +34.967.599200, Fax +34.967.599224

**Abstract**

Web Services Translation (WST) Tool is an integrated environment for the development of composite web services, which is based on several translations.
Firstly, the user may introduce the system requirements by means of KAOS elicitation technique. Then, the design of the composite web service is done by UML 2.0 sequence diagrams, which are automatically translated into Web Services Choreography Description Language (WS-CDL) specification documents. These WS-CDL specifications are translated into a network of timed automata, now as a novelty in the tool, and we have in turn incorporated the translation from a WS-CDL specification to prioritized timed-colored Petri nets (PTCPN). The final step that can be made through this tool is the validation and verification of the system design, by using the interface that it provides to the UPPAAL tool.

## 1 Introduction

WST Tool [41] was initially developed to implement the transformation from UML into WS-CDL [38] and after that, the WS-CDL document was translated into timed automata. In this new extension of WST Tool, we have added the transformation from WS-CDL to PTCPN. Thus, we may use both formalisms, prioritized timed-colored Petri nets or timed automata, in order to analyse the system behaviour, so we can profit from the specific features of each of these formal models in order to have a better knowledge of the system under study.

Let us first introduce the main principles of composite web services and the main techniques and technologies involved in their development. A Web Service [8] can be defined as a self-describing, self-contained modular application that can be published, located and invoked over a network, usually the Internet. Web Services are therefore applications that provide services obtainable through the Internet. Web Services composition has arisen as a natural and elegant way of providing new value-added services in a combination of already established services. The different suppliers can then act together to provide a new service; they can be written in different languages and they can be executed on different platforms. In this work, we have only taken into account the choreographic view of the composition, modeled by using WS-CDL.

WS-CDL [39] is a W3C [37] standard for the description of composite Web services, which allows us to describe peer-to-peer collaborations regardless of the supporting platform or programming model. WS-CDL describes the collaborations among the parties involved by means of choreographies and activities. However, the WS-CDL standard does not include any comments about priorities among interactions, so that we have also proposed an extension of the standard to add this capability. Specifically, we associate priorities with interactions, taking into account that in a composite Web service it can be interesting to favor some interactions against others.

We use a prioritized-timed model of colored Petri nets [21] to capture the main WS-CDL elements, providing thus a formal framework to describe precisely the behavior of the parties involved in the choreography. This specific model is supported by the CPN Tools [17], which allow us to make simulations, verification and analysis of properties.

In this work we describe the translation from WS-CDL to PTCPNS, and the implementation of this translation in WST Tool. With this translation we can get two main benefits. On the one hand, we obtain a graphical representation in terms of prioritized-timed colored Petri nets, which can help the software designer to get a complete view of the composed Web Services and the interactions among the different participants. On the other hand, Petri nets are also a formal tool, in the sense that they describe not only a static vision of a system, but also its dynamic

```
<interaction  name="ncname"
channelVariable="qname"
...>
<participate relationshipType="qname"/>
<exchange name="ncname"...>
<send     .../>
<receive  .../>
<exchange/>*
<priority value="P1">
<timeout time-to-complete....>
</priority>
.....
</record>*
</interaction>
```

Figure 1: Priority syntax

behavior. We can then use the Petri net representation to verify and validate the composed Web Services.

We have structured this report as follows: in Section 2 we show how to introduce priorities in WS-CDL; in Section 3 we introduce the particular model of prioritized-timed colored Petri net, and the translation is presented in Section 4; In Section 5 we show how this translation has been implemented in WST tool, in particular, the XSLT architecture. A case study that illustrates the translation is shown in Section 6, and finally, in Section 7 the conclusions and some indications about our future work are formulated.

## 2 WS-CDL with Priorities

In many cases it is desirable to favor some interactions against others, i.e., in the composition of Web Services some parties can express their interest in the prioritization of certain interactions. We can think in a Web Service for selling or reserving items of some different sorts. Clients interact with the Web Server to buy or reserve items, but these interactions may have different levels of priority associated, depending on the kind of item or even depending on the client that makes the request.

We therefore introduce an extension of WS-CDL with priorities, by adding a "priority attribute"to each interaction (in Fig. 1 we show the extension of WS-CDL with priorities). Priorities are established as natural numbers, with the same interpretation as in CPN Tools, the greater number, the lower activity priority in the system. Then, the interpretation of this attribute is the following: in the event of conflict only those interactions with the maximum priority (lowest value) are allowed.

## 3 Prioritized-Timed Colored Petri Nets

In this section we introduce the specific model of Prioritized-Timed Colored Petri Nets (PTCPN) that we consider for the translation.

We use a well-known model supported by the CPN Tools, prioritized-timed colored Petri nets [20], which are a prioritized-timed extension of colored Petri nets. In this model, places have associated a color set (with a related data type). Then, each token has an attached data value (token color), which belongs to the color to which the token is associated.

There is also a discrete global clock that represents the total time elapsed in the system model. Arcs have also an inscription associated, which are called arc expressions, and are constructed using variables, constants, operators and functions. To evaluate an arc expression we need to bind the variables, which consist in assigning a value to the variables that appear in the arc inscription. These values are then used to select the token colors that must be removed or added when firing the corresponding transition.

Arc expressions can also have a time information associated, both for place-transition and transition-place arcs. Actually, we only need time inscriptions in output arcs, and even, when all the output arcs of a transition have the same time inscription, there is a shorthand notation in CPN Tools by which this time information is associated to the transition instead of the output arcs, so this is the specific model that we use in our WS-CDL semantics, we will only consider these time inscriptions in the transitions. We will not use, therefore, any time inscription in the arcs.

Thus, the time inscription that a transition has associated is used to specify the delay that must be added to the current value of the global clock for every token generated by the firing of the transition. Furthermore, transitions can have guards associated, which are Boolean expressions that may prevent their firing. Thus, when a transition has a guard, it must evaluate to true for the binding to be enabled, otherwise the binding is disabled and the transition cannot be fired.

**Definition 1** (Notation)
The following notation will be used henceforth:

- $\mathbb{N}$ will denote the set of natural numbers, $\mathbb{N} = \{0, 1, 2, \ldots\}$ and $\Sigma = \mathbb{N} \times \mathbb{N}$.
- Multisets are defined as functions $C : X \to \mathbb{N}$, providing us with the number of instances of each element $x \in X$. As usual, we will enumerate the elements of a multiset $C$ as follows: $C = \{r_1.x_1, \ldots, r_n.x_n\}$, meaning that $C(x_i) = r_i$, for all $i = 1, \ldots, n$, and $C(x) = 0$, for all $x \neq x_i$, $i = 1, \ldots, n$.

  The set of multisets over a set $X$ will be denoted by $\mathcal{B}(X)$. For any $x \in X$ and $C \in \mathcal{B}(X)$ we say that $x \in C$ if and only if $C(x) > 0$.
- For any $C_1, C_2 \in \mathcal{B}(X)$, we define:
  - $C_1 + C_2 \in \mathcal{B}(X)$, where $\forall x \in X : (C_1 + C_2)(x) = C_1(x) + C_2(x)$.
  - $C_1 \subseteq C_2$ if and only if $\forall x \in X : C_1(x) \leq C_2(x)$.
  - If $C_2 \subseteq C_1$ we can define the subtraction $C_1 - C_2 \in \mathcal{B}(X)$, where $\forall x \in X : (C_1 - C_2)(x) = C_1(x) - C_2(x)$.
- For any $C \in \mathcal{B}(\Sigma)$, we define the first projection $\Pi_1(C) \in \mathcal{B}(\mathbb{N})$, as follows: $\forall n \in \mathbb{N}, \Pi_1(C)(n) = \sum_{m \in \mathbb{N}} C(n, m)$.
- For any $C \in \mathcal{B}(\Sigma)$ and $n \in \mathbb{N}$ we define the second projection $\Pi_2(C, n)$ as the ordered list that consists of the elements $(m_1, m_2, \ldots, m_{\Pi_1(C)(n)})$, such that $(n, m_i) \in C$, $\forall i = 1, \ldots, \Pi_1(C)(n)$ and $m_i \leq m_{i+1}$, $\forall i = 1, \ldots, \Pi_1(C)(n) - 1$.

- For any $C_1, C_2 \in \mathcal{B}(\Sigma)$, we say that $C_1 \preceq C_2$ if and only if the following conditions hold:
  - $\Pi_1(C_1) \subseteq \Pi_1(C_2)$.
  - $\forall n \in \mathbb{N}$, taking $\Pi_2(C_1, n) = (m^1_1, \ldots, m^1_{\Pi_1(C_1)(n)})$ and $\Pi_2(C_2, n) = (m^2_1, \ldots, m^2_{\Pi_1(C_2)(n)})$, we must have $m^1_i \geq m^2_i$, $\forall i = 1, \ldots, \Pi_1(C_1)(n)$.

  These conditions state that for every $n$ the total number of elements $(n, m)$ (moving $m$) must be lesser in $C_1$ than in $C_2$, and for every element $(n, m)$ in $C_1$ there must be a corresponding (distinct element) $(n, m')$ in $C_2$, with $m \geq m'$.

- For any $C_1, C_2 \in \mathcal{B}(\Sigma)$, with $C_1 \preceq C_2$, we define $C_2 \ominus C_1$ in the following (recursive) way:
  - For $C_1 = \emptyset$ we take $C_2 \ominus C_1 = C_2$.
  - For $C_1 \neq \emptyset$, let us consider that $C_2 = \{r^1_1.(n_1, m^1_1), \ldots, r^1_{i_{n_1}}.(n_1, m^1_{i_{n_1}}), \ldots, r^k_1.(n_k, m^k_1), \ldots, r^k_{i_{n_k}}.(n_k, m^k_{i_{n_k}})\}$, where $n_l \neq n_j$, $\forall l \neq j$, and $m^l_j < m^l_{j+1}$, $\forall l = 1, \ldots, k$ and $\forall j = 1, \ldots, i_{n_l}$.
    Since $C_1 \preceq C_2$, we can take one element $(n_l, m) \in C_1$, for some $l \in \{1, \ldots, k\}$, as well as the largest index $j$ for which $m^l_j \leq m$. We then define recursively:
    $C_2 \ominus C_1 = (\{r^1_1.(n_1, m^1_1), \ldots, r^1_{i_{n_1}}.(n_1, m^1_{i_{n_1}}), \ldots, r^l_1.(n_l, m^l_1), \ldots, (r^l_j - 1).(n_l, m^l_j), \ldots, r^l_{i_{n_l}}.(n_l, m^l_{i_{n_l}}), \ldots, r^k_1.(n_k, m^k_1), \ldots, r^k_{i_{n_k}}.(n_k, m^k_{i_{n_k}})\}) \ominus (C_1 - \{1.(n_l, m)\})$.
    Thus, $C_2 \ominus C_1$ is obtained by removing from $C_2$ elements $(n, m)$ that correspond to elements $(n, m')$ of $C_1$, such that $m$ is the largest value with $m \leq m'$.

    For instance, taking $C_1 = \{1.(2,3), 1.(2,5), 1.(1,4), 1.(7,6)\}$, and $C_2 = \{1.(2,0), 1.(2,1), 1.(2,2), 1.(1,3), 2.(7,6), 3.(3,3)\}$ it follows that $C_1 \preceq C_2$. Then, $C_2 \ominus C_1 = \{1.(2,0), 1.(7,6), 3.(3,3)\}$.

□

**Definition 2** (Prioritized-Timed Colored Petri Nets)
We define a prioritized-timed colored Petri net (PTCPN) as a tuple $(P, T, A, V, G, E, \lambda, D, \pi)$, where[1]:

- $P$ is a finite set of *places*, with colors in the set $\Sigma$. Thus, in our case, colors will be pairs $(n, x) \in \mathbb{N} \times \mathbb{N}$, where $n$ is the token value and $x$ its timestamp.
- $T$ is a finite set of *transitions* ($P \cap T = \emptyset$).
- $A \subseteq (P \times T) \cup (T \times P)$ is a set of directed *arcs*.
- $V$ is a finite set of *typed variables* in $\Sigma$, i.e. $\mathit{Type}(v) \in \Sigma$, for all $v \in V$.
- $G : T \longrightarrow \mathit{EXPR}_V$ is the *guard function*, which assigns a Boolean expression to each transition, i.e. $\mathit{Type}(G(t)) = \mathit{Bool}$.
- $E : A \longrightarrow \mathit{EXPR}_V$ is the *arc expression function*, which assigns an expression to each arc, such that $\mathit{Type}(E(a)) = \mathcal{B}(\mathbb{N} \times \{0\})$, which corresponds to untimed arcs, since, as mentioned above, we only attach time delays to transitions.

[1]We use the classical notation on Petri nets to denote the precondition and postcondition of both places and transitions:
$$\forall x \in P \cup T : {}^\bullet x = \{y \mid (y, x) \in A\} \qquad x^\bullet = \{y \mid (x, y) \in A\}$$

- $\lambda$ is the *labeling function*, defined both on places and transitions.
  - Places are labeled as *entry places*, *exit places*, *error places*, *internal places* and *variable places*, which, respectively, correspond to the following labels: $\{in, ok, er, i, rv\}$. In our specific model, a PTCPN will have an only *entry place* $p_{in}$, such that ${}^\bullet p_{in} = \emptyset$, which will be initially marked with a single token of color $(0, 0)$. There is also an only *exit place* $p_{ok}$, such that $p_{ok}^\bullet = \emptyset$, which will be marked with one token when the system finishes correctly. Each PTCPN has also a single *error place* $p_{er}$, such that $p_{er}^\bullet = \emptyset$, which will become marked with one token in the event of a failure. Variable places are denoted by $p_{rv}$, to mean that they capture the value of variable $v$ in role $r$. We will assume that a special value $e$ is used to denote that the variable has not yet been assigned. Finally, all the remaining places are considered as *internal*.
  - Transitions are labeled as follows: $\lambda(t) \in L \cup \{\emptyset\} \cup \{fail\}$, where $L$ is the set of basic activities, defined as follows:

$$L = \{time\_out, silent, noaction(r), assign(r, v, n), inter(r_1, r_2, v_1, v_2)\}$$

- $D : T \longrightarrow \mathbb{N} \times \mathbb{N}$, which is the *delay function*, which associates a time interval to each transition. For $D(t) = [d_1, d_2]$, this means that a uniform probability function will be used when $t$ is fired to select the specific discrete delay in that time interval.

- $\pi : T \longrightarrow \mathbb{N}$ is the *priority function*, which assigns a priority level to each transition.

In this definition, $EXPR_V$ denotes the expressions constructed using the variables in $V$, with the same syntax admitted by CPN Tools.

□

**Definition 3** (Markings)
Given a PTCPN $N = (P, T, A, V, G, E, \lambda, D, \pi)$, a marking $M$ is defined as a function $M : P \longrightarrow \mathcal{B}(\Sigma)$, which assigns a multiset of colors to each place (which can be empty).

A timed marking of a PTCPN $N$ is a pair $(M, x)$, where $M$ is a marking of $N$ and $x$ is the current system time instant. A marked prioritized-timed colored Petri net (MPTCPN) is then defined as a triple $(N, M, x)$, where $N$ is a PTCPN, and $(M, x)$ a timed marking of it.

□

We define the semantics for MPTCPNs in a similar way as in [21], now taking into account that transitions have associated priorities. We first introduce the notion of *binding*, then the *enabling condition* and finally the *firing rule* for MPTCPNs.

**Definition 4** (Bindings)
Let $N = (P, T, A, V, G, E, \lambda, D, \pi)$ be a PTCPN. A *binding* of a transition $t \in T$ is a function $b$ that maps each variable $v \in Var(t)$ into a value $b(v) \in \Sigma$, where $Var(t)$ is defined as the set of variables that appear both in the guard of $t$ and in the arc expressions of the arcs connected to $t$. We will denote by $B(t)$ the set of all possible bindings for $t \in T$.

Given an expression $e \in EXPR_V$, we will denote by $e\langle b\rangle$ the evaluation of $e$ for the binding $b$.

A *binding element* is then defined as a pair $(t, b)$, where $t \in T$ and $b \in B(t)$. The set of all binding elements is denoted by $BE$.

□

**Definition 5** (Enabling condition)
Let $N = (P, T, A, V, G, E, \lambda, D, \pi)$ be a PTCPN, and $(M, x)$ a timed marking of it. We say that a binding element $(t, b) \in BE$ is *enabled* at the time instant $x'$ in the timed marking $(M, x)$ if and only if the following conditions are fulfilled:

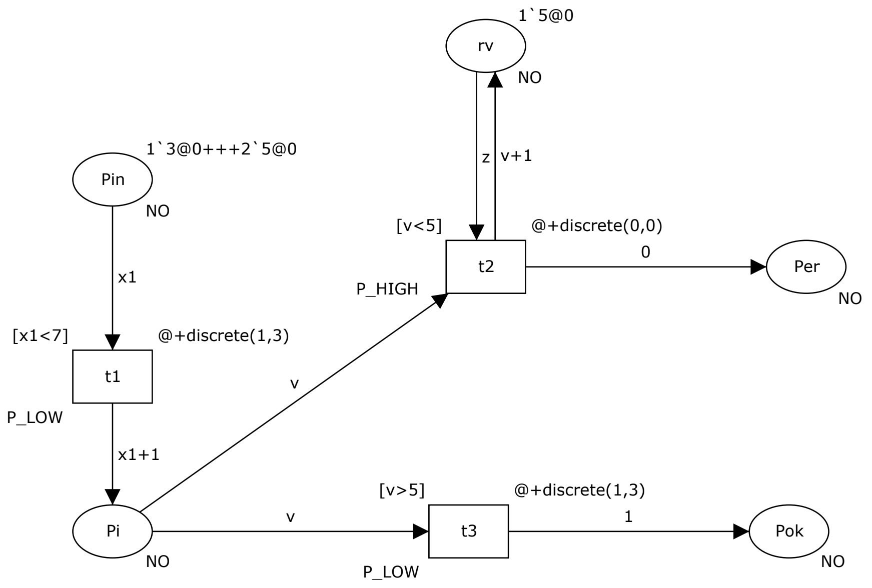


Figure 2: Graphical view of a PTCPN.

1. $x' \geq x$.
2. $G(t)\langle b\rangle = true$.
3. For all $p \in {}^{\bullet}t$, $E(p,t)\langle b\rangle_{x'} \preceq M(p)$, where $E(p,t)\langle b\rangle_{x'}$ consists of the same colors as $E(p,t)\langle b\rangle$, but replacing their timestamp (which was 0) by $x'$.
4. There is no other binding element $(t', b') \in BE$ fulfilling the previous conditions such that $\pi(t') < \pi(t)$.
5. $x'$ is the smallest time value for which there exists a binding element $(t, b)$ fulfilling these conditions.

□

**Definition 6** (Firing rule)
Let $N = (P, T, A, V, G, E, \lambda, D, \pi)$ be a PTCPN, $(M, x)$ a timed marking of $N$, and an enabled binding element $(t, b) \in BE$ at instant $x'$ in the timed marking $(M, x)$.

The firing of $(t, b)$ at instant $x'$ is non-deterministic, depending on the chosen delay $d \in \mathbb{N}$ for the transition. This delay is randomly selected in the interval given by $D(t)$. Thus, the new timed marking $(M', x')$ is:

$$\forall p \in P \,:\, M'(p) = M(p) \ominus E(p,t)\langle b\rangle_{x'} + E(t,p)\langle b\rangle_{d+x'}$$

□

**Example 1** Let us consider the marked PTCPN depicted in Figure 2, obtained from CPN Tools. Observe that timed color tokens in CPN Tools are drawn using the notation $n`v@x$, meaning that we have $n$ instances of a timed color token with color value $v$ and *timestamp* $x$, which correspond to $n.(v, x)$ according to our formal notation. Besides, the symbol '+++' is there used to represent the union of timed multisets.

Thus, $p_{in}$ is initially marked with one token of color $(3, 0)$, and two tokens of color $(5, 0)$, and the place $rv$ has one token with color $(5, 0)$. Transitions are labeled with their associated guard, time interval and priority information. Arcs are labeled with the corresponding expressions, in which no time delays appear, as we are considering that only transitions have associated time delays.

From the initial marking we can see that only transition $t_1$ can be fired (at instant 0), and any token of those in $p_{in}$ can be used for that purpose. Taking $(5, 0)$ we get the binding $x = 5$, which fulfills the transition guard. The firing of $t_1$ with this binding removes one instance of $(5, 0)$ from $p_{in}$, and produces a new token on $p_i$. The timestamp of this new token is a discrete value in the interval $[1, 3]$ (let us say 3). Thus, considering the output arc inscription we get a token $(6, 3)$ on $p_i$.

Now, transition $t_1$ must fire again twice (until $p_{in}$ becomes empty), due to the time constraints of this model. As a result we may obtain in $p_i$ the following marking $\{1.(4, 3), 1.(6, 1), 1.(6, 3)\}$ (the timestamp values depend on the values chosen from the interval $[1, 3]$). The only transition that can be fired at this marking is $t_3$, because due to the time constraints we must first use the token $(6, 1)$ and $t_2$ cannot be fired using this token. The firing of $t_3$ produces a new token on $p_{ok}$, whose color value must be 1, and the timestamp depends again on the chosen delay in the time interval $[1, 3]$. For instance, we could obtain the color token $(1, 4)$.

Only two tokens now remain on $p_i$, with colors $(4, 3)$ and $(6, 3)$, and $t_2$ becomes the only transition enabled (due to condition (4) of Def. 5). Its firing removes the token $(4, 3)$ from $p_i$, the token on the place $rv$ changes to $1.(5, 3)$, and creates a new token on $p_{er}$, with color $(0, 3)$. Finally, the remaining token $(6, 3)$ on $p_i$ only allows us to fire $t_3$, generating a new token on $p_{ok}$, with value 1 and a timestamp depending on the delay chosen for its firing.

$\square$

## 4 PTCPN Semantics for WS-CDL

In this section we provide a PTCPN semantics for the considered WS-CDL subset. Our goal is to obtain a PTCPN representation capturing the main aspects of Web service composition, and specially those related to data, time and priorities. This representation will then capture the visible behavior of the participants in a web service composition and their interactions.

The obtained PTCPNs will be 1-safe [34], which means that only one token can be at any place in any reachable marking. Moreover, when one of the initial or final places is marked, no other place can be marked at the same time, except places associated with variables or the dead tokens that remain in some places when the error place has been marked. Furthermore, all of the generated PTCPNs will have one initial place[2], which activates the PTCPN when it is marked, and two exit places, which do not have any postconditions and cannot be marked simultaneously. These exit places correspond to the correct or erroneous termination of the system represented by the PTCPN.

The starting point is a WS-CDL document with the syntax of interactions extended considering priorities. We assume that all the priority values in the WS-CDL document are greater than or equal to one, with the purpose of reserving the maximum priority value (0), which will be used in the translation of some WS-CDL structural elements.

[2]This does not mean that this is the only initially marked place.

We consider that we have only the root choreography, i.e. there is only one choreography in the document. The different elements of the document are thus translated as follows[3]:

- *RoleTypes*: These are used to enumerate the observable behavior of each party. Transitions can be given a label indicating the roletype involved in their execution.
- *RelationShipTypes* and *Channels*: Both elements are used in interactions, so they are (implicitly) considered in the translation provided for interaction activities.
- *Information types and Variables*: For simplicity, we only consider two variable types: date/time variables and integer variables. Date and time variables are used to establish the time constraints under which some activities can (or must) be performed. Their use is therefore restricted, in the sense that will be explained when we describe the translation of the WS-CDL structural elements in which they can appear. Integer variables are used to represent the commonly observable information in collaborations. These are translated by using the colored places labeled by *rv*, whose colored marking indicates the current value of the variable.

  Data variables can be assigned a value using the *assign* activity; they can be used in interaction activities, and also in the guards of the workunits.
- *Choreography*: This is, of course, the main element of the WS-CDL document. It describes the activities to be performed by the different participants, and may contain an exception block. Compositionally translating each one of these elements, we then have:

$$N_a = (P_a, T_a, A_a, V_a, G_a, E_a, \lambda_a, D_a, \pi_a) \text{ (PTCPN for the main activities)}$$
$$N_e = (P_e, T_e, A_e, V_e, G_e, E_e, \lambda_e, D_e, \pi_e) \text{ (PTCPN for the exception block)}$$

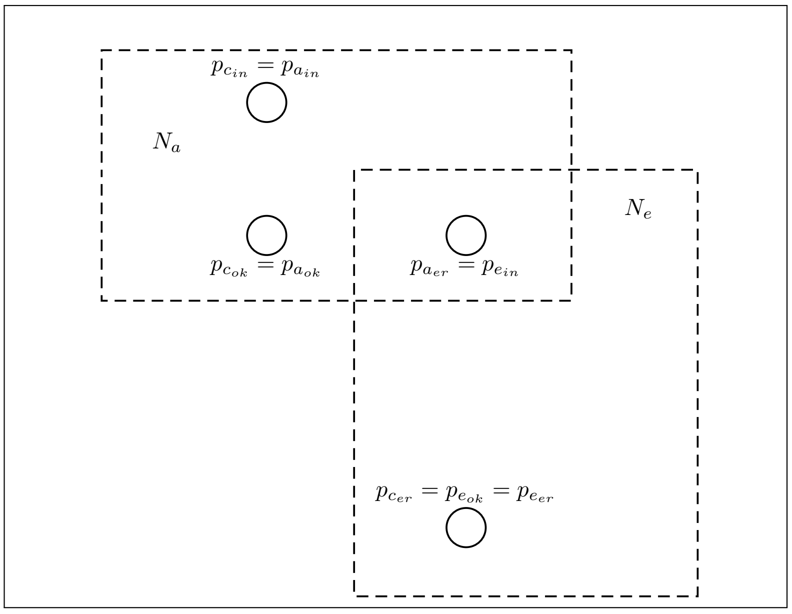

$p_{c_{in}} = p_{a_{in}}$
$N_a$
$N_e$
$p_{c_{ok}} = p_{a_{ok}}$
$p_{a_{er}} = p_{e_{in}}$
$p_{c_{er}} = p_{e_{ok}} = p_{e_{er}}$

Figure 3: Choreography Translation

[3]We omit the specific syntax of each element, which can be found in the WS-CDL description document [38], and we also omit the formal definitions of the PTCPNs obtained for each case, which can be easily deduced from the figures.

Let $p_{a_{in}}$ and $p_{e_{in}}$ be the initial places of $N_a$ and $N_e$ respectively; $p_{a_{ok}}$ and $p_{e_{ok}}$ their *correct* exit places, and $p_{a_{er}}$, $p_{e_{er}}$ their *erroneous* exit places. The PTCPN for the choreography is then constructed as indicated in Figure 3, where we are joining the following places:

$$\begin{array}{rcl} p_{c_{in}} & = & p_{a_{in}} \\ p_{c_{er}} & = & p_{e_{ok}} = p_{e_{er}} \\ p_{c_{ok}} & = & p_{a_{ok}} \\ p_{a_{er}} & = & p_{e_{in}} \end{array}$$

and the remaining places, transitions and edges are the same as in $N_a$ and $N_e$. The PTCPN is then activated by putting one token $(0, 0)$ on $p_{c_{in}}$. The other places, $p_{c_{ok}}, p_{a_{er}}$ and $p_{c_{er}}$ in this figure, as well as all the internal places, are initially unmarked. Notice, however, that we can have other marked places, specifically those associated with integer variables, whose initial marking is $1.(e, 0)$, where $e \in \mathbb{N}$ is a natural value reserved to represent that the variable has not yet been assigned.

- *Activities*: We may have basic activities, workunits and ordering activities. The translation for each one is shown in the following subsections.

## 4.1 Basic activities

As we are considering only a choreography (root), we do not need to consider either the basic activities *perform* or *finalize* (see [38] for a description of the different WS-CDL activities). For the remainder the translation works as follows:

- *Assign, Silent* and *Noaction* activities. These are translated as indicated in Fig. 4, by means of a single transition with the lowest priority (we call it P0, obtained by taking the highest priority numeric value used in the WS-CDL document plus one) labeled with the name of the corresponding activity.

  As we consider that the time required to execute *assign* and *noaction* is negligible, the corresponding transitions have a null delay associated, which means that they are immediately executed, once they become enabled, because their guard is true. Notice that for the *assign* activity translation we use a self loop between the transition and the place associated with the variable ($rv$) in order to replace its previous value by $n$.

  We associate a time argument $x$ to the silent activity, which captures the time required for its execution. The corresponding transition is then labeled with this delay (interval $[x, x]$) to enforce its execution after $x$ units of time, once it becomes enabled, because the guard of this transition is *true*. We also consider that these basic activities cannot finish abnormally.

- Interaction activities.

  We only consider one message exchange within each interaction activity, which takes a value from a source variable and assigns a target variable with that value. However, if the source variable has not yet been assigned an error occurs, and the interaction finishes abnormally.

  Interaction activities may also have an associated *time-out*$(x)$. In this case, if the *time-out* expires and the interaction has not been performed, it finishes abnormally. In addition, a priority attribute $(l)$ may have been indicated, and this value is used as the priority of the corresponding transition in the PTCPN representation, otherwise it has the minimum priority (P0).

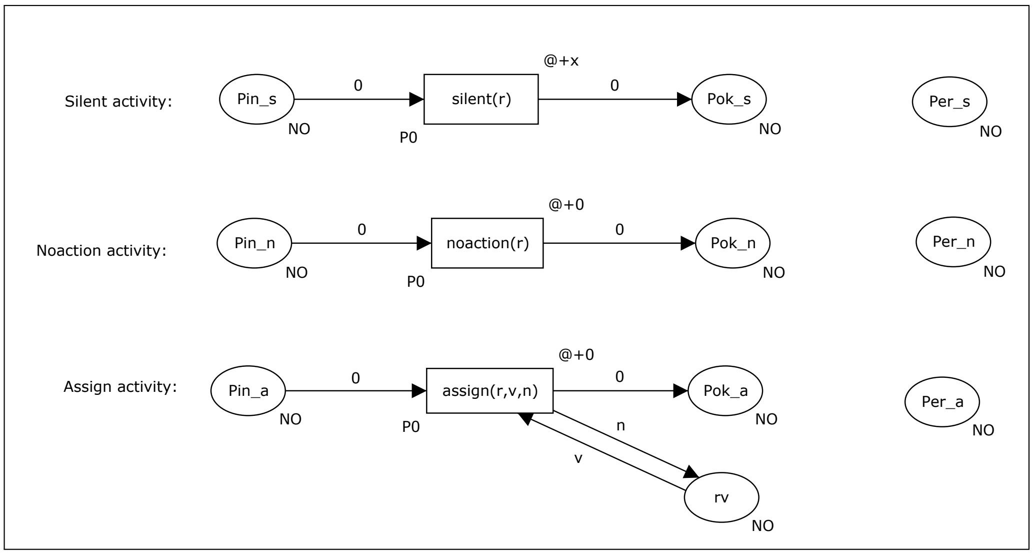


Figure 4: Basic Activity Translation

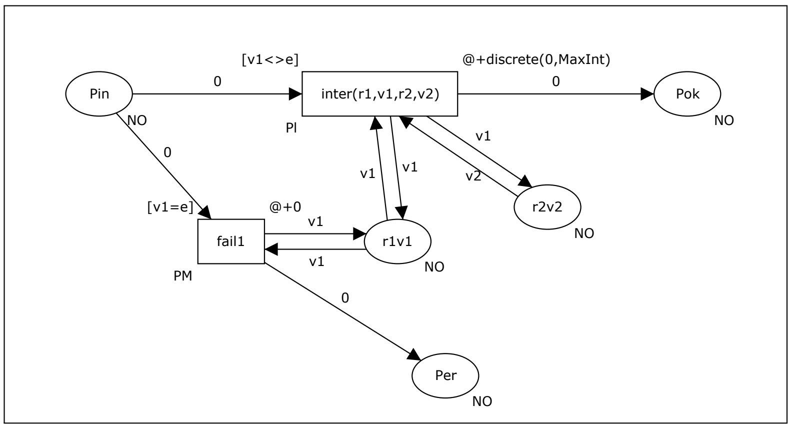


Figure 5: Translation of an interaction without time-out.

Figure 5 illustrates the translation of an interaction without an associated time-out. Transition *fail1* has been introduced to capture the abnormal termination that occurs when the source variable ($v1$) in the interaction is unassigned. Transition *fail1* is labeled with the *fail* action, it has the guard condition $v = e$ and maximum priority (PM is 0), so it is immediately fired when the source variable has not been assigned[4]. The firing of the transition *inter* corresponds to the execution of the interaction, it takes the value of *v1* from the token color on place *r1v1*, and changes the token color on *r2v2* with this value. This transition

[4]The value $e$ means that the variable remains unassigned.

can be fired at any moment, so its associated time interval is $[0, MaxInt]$, where *MaxInt* represents the maximum integer value supported by the tool.

The translation for an interaction with an associated time-out is depicted in Fig. 6, in which two additional transitions are included, *time_out* and *fail2*, with $\lambda(time\_out) = time\_out$, and $\lambda(fail2) = fail$. The firing of transition *time_out* represents the passage of $x + 1$ time units without performing the interaction. In this case, once *time_out* has been fired and $x+1$ time units have elapsed, we must immediately fire the transition *fail2*, which corresponds to the abnormal termination due to the expiration of the time-out. Transition *fail2* has again the maximum priority (PM), since exception conditions are immediately executed when they occur. We could have, for instance, an error condition in one branch of a parallel activity, the other branch must then be immediately aborted, and the whole parallel activity terminates abnormally.

Notice that the token generated by the transition *time_out* on its postcondition place *Pd* will only be available after $x + 1$ time units. In the meanwhile other simultaneous actions can take place. For instance, if we have an activity running in parallel, it can perform inner activities until this time-out expires and throws the exception (which occurs once the token on place *Pd* can be used for firing *fail2*).

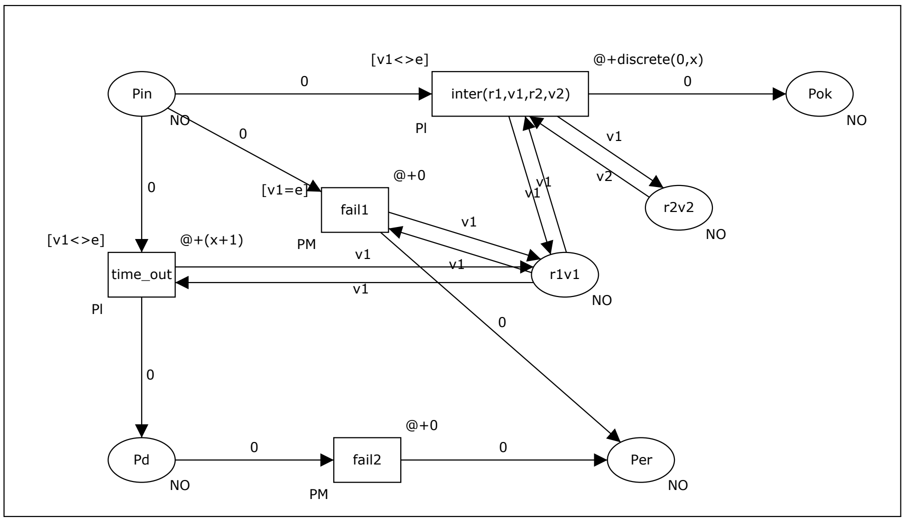


Figure 6: Translation of interaction with time-out.

## 4.2 Workunits

Figure 7 shows the syntax of workunits, where the main elements are the activity inside the workunit, the guard that allows the activation of the workunit, the guard that captures the repetition condition, and the Boolean attribute *block*, which specifies whether the workunit must wait until the activation condition becomes true or not. As stated above, we allow the use of date and time variables in WS-CDL to establish a time constraint for the execution of a workunit, although we restrict the use of these variables to simplify the translation. They can only be used in the workunit activation guards, and for this purpose only, i.e. to

```
<workunit  name="ncname"
      guard="xsd:boolean XPath-expression"?
      repeat="xsd:boolean XPath-expression"?
      block="true|false"? >
      ...
      Activity-Notation
      ...
</workunit>
```

Figure 7: Workunit syntax

```
<assign roleType="tns:T2">
    <copy name="MinDurationInfo">
        <source duration="2:00" />
        <target variable="cdl:getVariable
                   ('min','','','tns:T2')" />
    </copy>
</assign>
<assign roleType="tns:T2">
    <copy name="MaxDurationInfo">
        <source duration="5:00" />
        <target variable="cdl:getVariable
                   ('max','','','tns:T2')" />
    </copy>
</assign>
<workunit  name="WhileDeadlineAvailable"
           guard="hasDurationPassed('min',xsd:T2) and not(hasDurationPassed('max', xsd:T2))"
           repeat="cdl:getVariable('v2','','','tns:T2') = 2"
           block="true">
        <interaction name="acceptReservation" ... >
            <participate relationshipType="tns:T1-T2"
                         fromRoleTypeRef="tns:T1" toRoleTypeRef="tns:T2"/>
            <exchange name="response"
                      informationType="tns:SeatsInfoType" action="request">
                <send variable="cdl:getVariable('v1','','','tns:T1')" />
                <receive variable="cdl:getVariable('v2','','','tns:T2')"/>
            </exchange>
            <priority value="2">
            <timeout  time-to-complete="4:00"
        </interaction>
</workunit>
```

Figure 8: Delayed workunit example

establish time intervals for the workunit execution. No other variable can appear in the guards in this case, and the workunit block attribute must be true to enforce the delay.

A workunit may have other activation guards, in which some (integer) data variables from the different role types can be checked. For both cases we provide the corresponding translation.

– Delayed Workunit. A *delayed workunit* is a particular case of a workunit in which time variables are used in order to establish a time interval for the workunit execution. Notice that in order to enforce the delay the block attribute of a delayed workunit must always be true.

Figure 8 shows the WS-CDL syntax that can be used to specify a delayed workunit. In this case we use the function *hasDurationPassed* and two time variables, *min* and *max*, in order to establish the time interval in which the workunit can be executed. The same effect could also be obtained by using a time variable and the function *getCurrentDateTime*.

The corresponding translation is shown in Figure 9, in which $A_1$ is the activity inside the workunit and $N_{A_1}$ its corresponding PTCPN. There is a new transition *tt* connecting $p_{in}$ with $p_{in_{A_1}}$, with $\lambda(tt) = \emptyset$ and time interval $[x, y]$, where $x, y$ are, respectively, the values of the variables *min*, *max* in the WS-CDL specification.

In addition, we replicate every initial transition of $N_{A_1}$, i.e. for every $t1 \in p^{\bullet}_{in_{A_1}}$ we consider a new transition, *t1r*, with the same interval, label and priority as $t1$, and its guard is obtained as a conjunction of the guard of $t1$ (*gt1* in the Figure) and the repetition condition of the workunit ($g'$). Then, *t1r* is connected as follows: ${}^{\bullet}t1r = \{p_{okA_1}\} \cup ({}^{\bullet}t1 \setminus \{p_{in_{A_1}}\}) \cup \{p_{r_iv_i} \,|\, v_i \text{ appears in } g'\}$, and $t1r^{\bullet} = t1^{\bullet} \cup \{p_{r_iv_i} \,|\, v_i \text{ appears in } g'\}$. For any variable $v_i$ appearing in $g'$, if there is already a self-loop arc connecting *t1r* with $p_{r_iv_i}$, we keep the existing label in both arcs. Otherwise, both arc expressions are $v_i$. The purpose of these transitions *t1r* is therefore to perform $A_1$ again when it has been correctly terminated and $g'$ is evaluated to true.

There is also a new transition $t$, with $\lambda(t) = \emptyset$ and maximum priority, whose guard is the negation of the workunit repetition condition and puts one token on $p_{ok}$ when $g'$ is false.

– Data Workunit.
  We now consider the case of a workunit with an activation guard in which we may check the value of some data variables. We now distinguish two cases, according to the block attribute value:

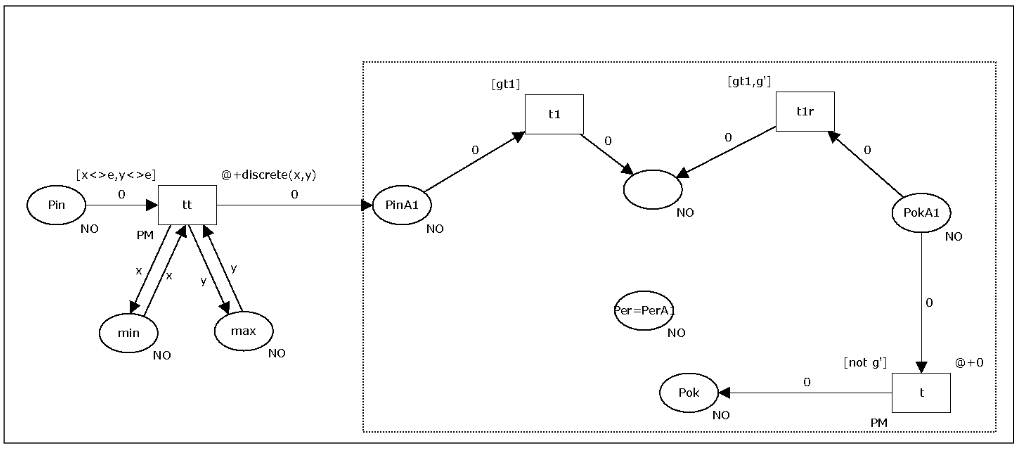


Figure 9: Delayed Workunit Translation

– ***Block = true*** (Figure 10).

In this case every initial transition $t_1 \in p^{\bullet}_{in_{A_1}}$ of $N_{A_1}$ (the PTCPN corresponding to the activity inside the workunit) is replaced by two new transitions, $t_1'$ and $t_1''$, connected as follows:

$$\begin{aligned}
&{}^{\bullet}t_1' = {}^{\bullet}t_1 \cup \{p_{r_iv_i} \,|\, v_i \text{ appears in } g\}\\
&{}^{\bullet}t_1'' = \{p_{ok_{A_1}}\} \cup ({}^{\bullet}t_1 \setminus \{p_{in_{A_1}}\}) \cup \{p_{r_iv_i} \,|\, v_i \text{ appears in } g'\},\\
&t_1'^{\bullet} = t_1^{\bullet} \cup \{p_{r_iv_i} \,|\, v_i \text{ appears in } g\}\\
&t_1''^{\bullet} = t_1^{\bullet} \cup \{p_{r_iv_i} \,|\, v_i \text{ appears in } g'\},
\end{aligned}$$

For any variable $v_i$ appearing in $g$ (resp. $g'$), if there was already a self-loop arc connecting *t1'* (resp. $t_1''$) with $p_{r_iv_i}$, we keep the existing label in both arcs. Otherwise, both arc expressions are $v_i$.

These transitions have the same interval, label and priority as $t_1$, and their guards are obtained as follows:

- For $t'_1$ we take the conjunction of the guard of $t_1$ (*gt1*) with the activation guard ($g$) of the workunit.
- For $t''_1$ we take the conjunction of the guard of $t_1$ (*gt1*) with the repetition guard ($g'$) of the workunit.

We also have a new transition $t$ with maximum priority, $\lambda(t) = \emptyset$, and its guard is the negation of the repetition condition of the workunit. This transition will be fired when the repetition condition is false, thus generating one token on $p_{ok}$.

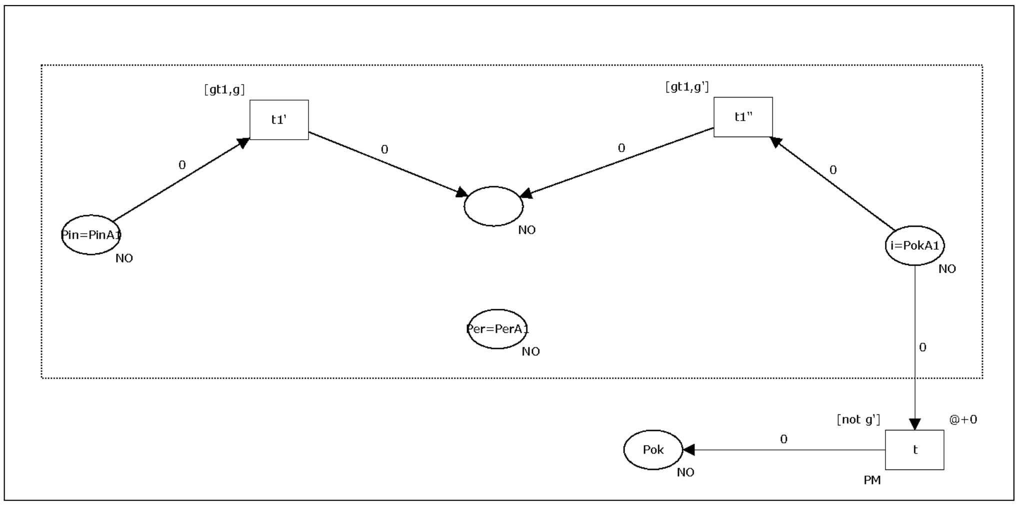


Figure 10: Data Workunit Translation for *block = true.*

- ***Block = false*** (Figure 11).

  The only difference with the previous case is the new transition *tb* with maximum priority, $\lambda(tb) = \emptyset$ and guard $\neg g$ (activation guard). Thus, when the guard condition is false, transition *tb* is immediately fired and the workunit is skipped.

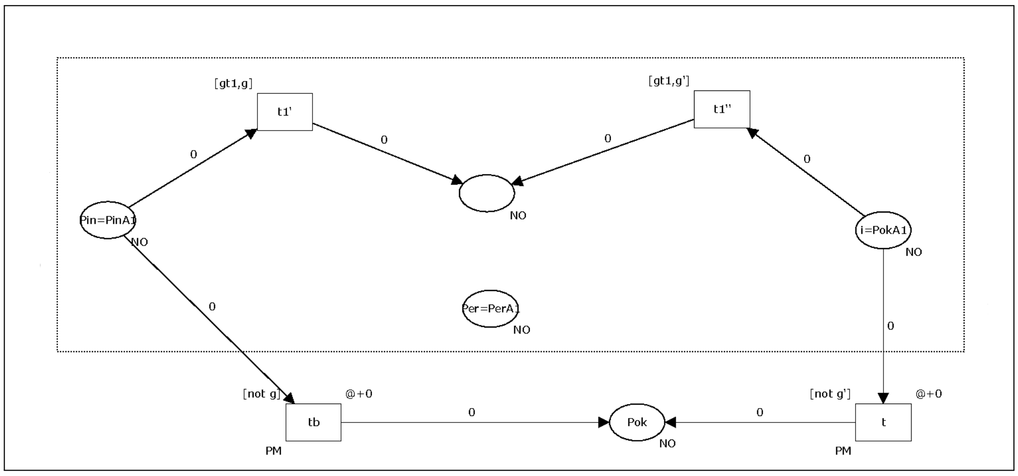


Figure 11: Data Workunit Translation for *block = false.*

## 4.3 Ordering structures

These are used to combine activities in a nested structure that uses the sequence, parallel and choice constructs. For all these cases we provide the translation by only considering two activities. However, the generalization to a greater number of activities is straightforward in all of them.

- Sequence: A sequence of two activities (with PTCPNs $N_{A_1}$ and $N_{A_2}$, respectively) is translated in a simple way (Figure 12), by just collapsing in a single place (this will be an internal place of the new PTCPN) the *correct exit* place of the $N_{A_1}$ and the *entry* place of $N_{A_2}$. The *entry* place of the new PTCPN will be the *entry* place of $N_{A_1}$. The *correct exit* place of the new PTCPN will be the *correct exit* place of $N_{A_2}$, and we also join the *error* places.

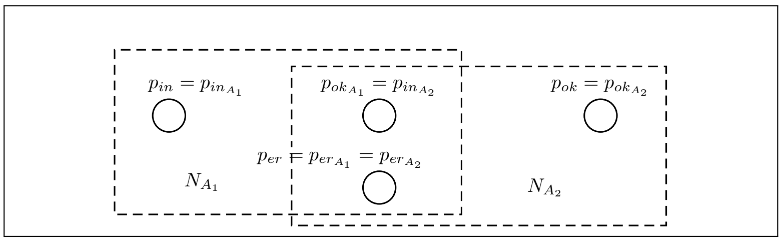

$p_{in} = p_{in_{A_1}}$
$p_{ok_{A_1}} = p_{in_{A_2}}$
$p_{ok} = p_{ok_{A_2}}$
$p_{er} = p_{er_{A_1}} = p_{er_{A_2}}$
$N_{A_1}$
$N_{A_2}$

Figure 12: Sequence Translation

- Parallel: The translation for a parallel activity is depicted in Figure 13, which includes two new transitions $t1$ and $t2$. The first to fork both parallel activities and the second to join them when correctly terminated. Both transitions have label $\emptyset$ and maximum priority to avoid other transitions being delayed (or not executed) due to their presence. We could have, for instance, an initial transition in $N_{A_1}$ with high priority, but as its activation depends on the execution of $t1$, another transition of another parallel activity (with lower priority) could be executed first if $t1$ is not executed immediately (as an action with the maximum priority in the model).

  Transition $t1$ thus puts one token on the initial places of both PTCPNs, $N_{A_1}$ and $N_{A_2}$, in order to activate them, and also puts one token on a new place, $p_c$, which is used to stop the execution of one branch when the other has failed. This place is therefore a precondition of every transition in both PTCPNs, and it is also a postcondition of the non-failing transitions. However, in the event of a failure, the corresponding *fail* transition will not put the token back on $p_c$, thus arresting the other parallel activity.

  Notice also that the *error* places of $N_{A_1}$ and $N_{A_2}$ have been joined in a single error place ($p_{er}$), which becomes marked with one token on the firing of one *fail* transition. In this case, the other activity cannot execute any more actions ($p_c$ is empty), so some useless tokens would remain permanently on some places in the PTCPN. However, it should be noticed that these tokens cannot cause any damage, since the control flow has been transferred to the exception block PTCPN, once the place $p_{er}$ has become marked.

- Choice: We now impose a syntactical restriction: no parallel operator can appear at the first level of the arguments of a choice. This restriction is introduced for technical reasons: the translation of a parallel activity creates an immediate initial transition with maximum priority, so if we allow a parallel activity as argument of a choice, according to the translation depicted in Figure 14 this transition would be fired immediately, due to its maximum priority, i.e. we would not actually have a choice.

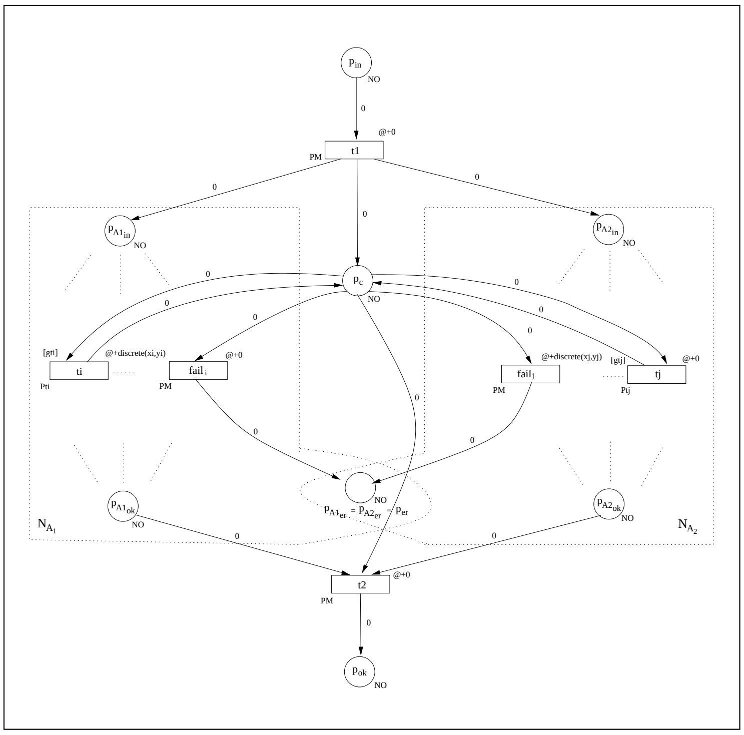


Figure 13: Parallel Activity Translation.

We can see in Figure 14 that the translation of a choice of two activities (with PTCPNs $N_{A_1}$ and $N_{A_2}$) is made by joining the *entry*, *error* and *correct termination* places of both PTCPNs. The structure of both PTCPNs is maintained, except for the following cases:

– When at most one of the arguments (let us say $A_1$) has one initial *fail* transition ($t \in p_{in_{A_1}}^{\bullet}$, $\lambda(t) = fail$), then, we remove this initial *fail* transition of $N_{A_1}$, as well as the arcs connected with it. Observe that as a consequence of the compositional construction there cannot be any other initial *fail* transitions in $N_{A_1}$. The choice therefore cannot fail when only one of the argument activities can fail.

– In the event of both PTCPNs having initial *fail* transitions, these are joined in a single *fail* transition, with maximum priority, delay 0, and its guard is the conjunction of the guards of both *fail* transitions. Thus, the choice can only fail when both activities are able to fail.

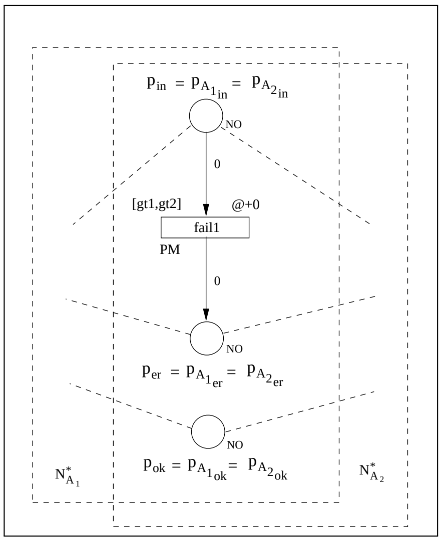

$p_{in} = p_{A_{1_{in}}} = p_{A_{2_{in}}}$
NO
0
[gt1,gt2]
@+0
fail1
PM
0
NO
$p_{er} = p_{A_{1_{er}}} = p_{A_{2_{er}}}$
NO
$p_{ok} = p_{A_{1_{ok}}} = p_{A_{2_{ok}}}$
$N^*_{A_1}$
$N^*_{A_2}$

Figure 14: Choice Activity Translation.

```
<parallel>
    <sequence>
        <assign roleType="tns:T2">
            <copy name="MinDurationInfo">
                <source duration="2:00" />
                <target variable="cdl:getVariable
                           ('min','','','tns:r2')" />
            </copy>
        </assign>
        <assign roleType="tns:T2">
            <copy name="MaxDurationInfo">
                <source time="5:00" />
                <target variable="cdl:getVariable
                           ('max','','','tns:r2')" />
            </copy>
        </assign>
        <workunit  name="WhileDeadlineAvailable"
                   guard="hasDurationPassed('min',xsd:r2)
                                 and not(hasDurationPassed('max', xsd:r2))"
                   repeat="cdl:getVariable('v2','','','tns:r2') = 2"
                   block="true">
            <interaction name="acceptReservation" ... >
                <participate relationshipType="tns:r1-r2"
                           fromRoleTypeRef="tns:r1" toRoleTypeRef="tns:r2"/>
                <exchange name="response"
                        informationType="tns:SeatsInfoType" action="request">
                    <send variable="cdl:getVariable('v1','','','tns:r1')" />
                    <receive variable="cdl:getVariable('v2','','','tns:r2')"/>
                </exchange>
                <priority value="2"/>
            </interaction>
        </workunit>
    </sequence>
    <choice>
        <assign roleType="tns:r2">
            <copy name="v2">
                <source value="2" />
                <target variable="cdl:getVariable
                        ('v2','','','tns:r2')" />
            </copy>
        </assign>
        <noAction roleType="tns:r2" />
    </choice>
</parallel>
```

Figure 15: Illustration of the WS-CDL ordering structures

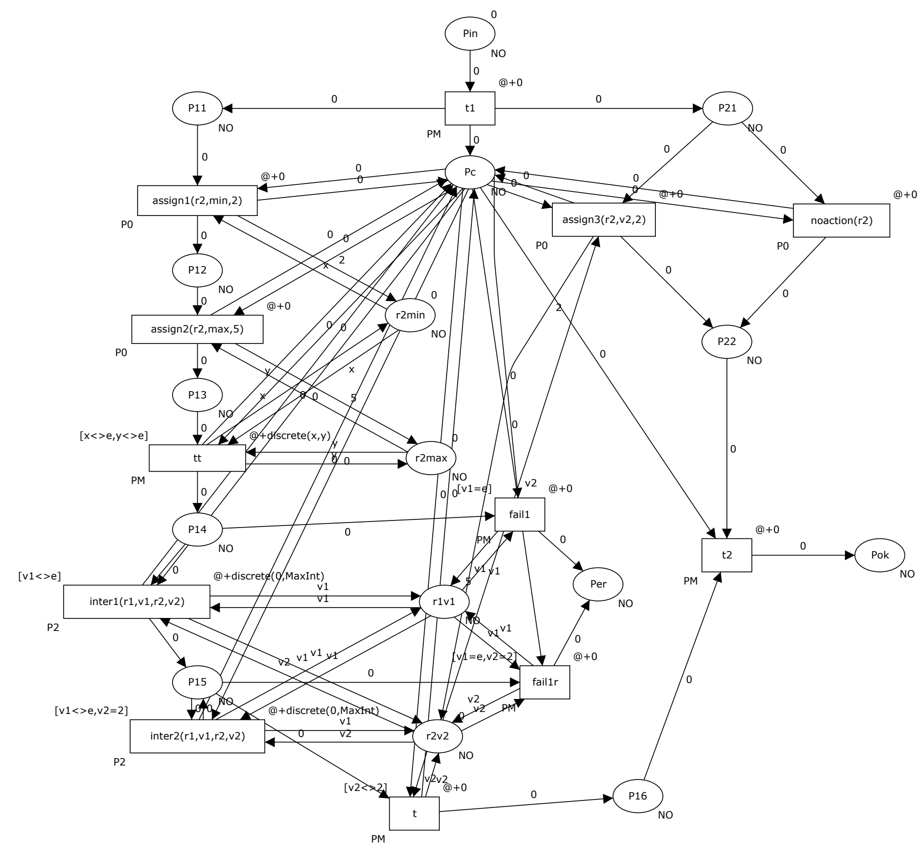


Figure 16: PTCPN corresponding to Example 2.

Notice that the initial *time_out* transitions of both PTCPNs are preserved by this construction, which means that the highest priority *time_out* transition whose guard is true will fix the time-out associated with the choice.

**Example 2** Figure 15 shows part of a WS-CDL document illustrating the use of the ordering structures and a delayed workunit. Its corresponding PTCPN is depicted in Figure 16.

□

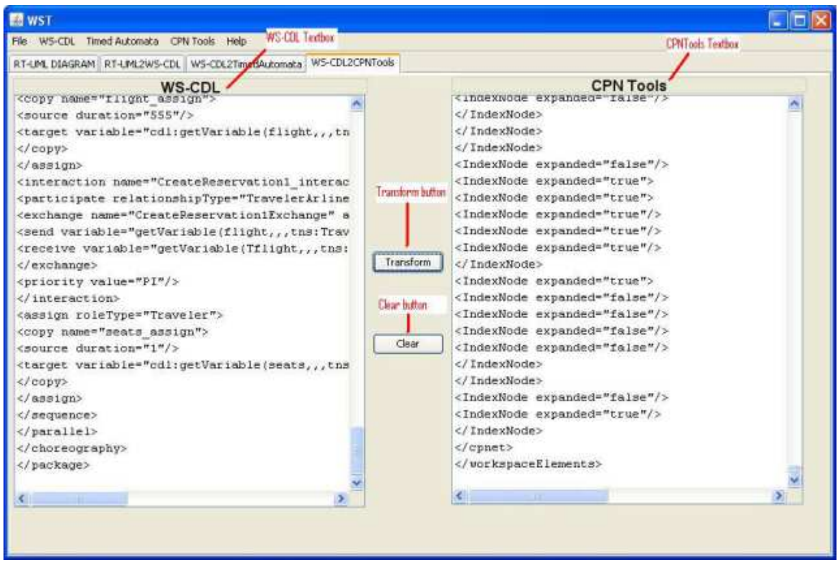
## 4.4 Exception Blocks

Choreographies may have one exception block. The exception block consists of some (possibly guarded) workunits, only one of which can be finally executed (the first one whose guard evaluates to true). For simplicity we can assume that only one non-guarded workunit is defined in the exception block (the so-called *default exception workunit*). Exception workunits cannot be repetitive and their *block* attribute must be false, so that the translation of the default exception workunit is that of the activity inside it.

# 5 WST Tool

## 5.1 Previous work and new additions

WST Tool is an integrated environment for the development of composite web services, in which we have added a new function, which is the automatic translation from WS-CDL specifications



```
<copy name="flight_assign">
<source duration="555"/>
<target variable="cdl:getVariable(flight,,,tn
</copy>
</assign>
<interaction name="CreateReservation1_interac
<participate relationshipType="TravelerArline
<exchange name="CreateReservation1Exchange" a
<send variable="getVariable(flight,,,tns:Trav
<receive variable="getVariable(Tflight,,,tns:
</exchange>
<priority value="PI"/>
</interaction>
<assign roleType="Traveler">
<copy name="seats_assign">
<source duration="1"/>
<target variable="cdl:getVariable(seats,,,tns
</copy>
</assign>
</sequence>
</parallel>
</choreography>
</package>
```

```
<IndexNode expanded="false"/>
</IndexNode>
</IndexNode>
</IndexNode>
<IndexNode expanded="false"/>
<IndexNode expanded="true">
<IndexNode expanded="true">
<IndexNode expanded="true"/>
<IndexNode expanded="true"/>
<IndexNode expanded="true"/>
</IndexNode>
<IndexNode expanded="true">
<IndexNode expanded="false"/>
<IndexNode expanded="false"/>
<IndexNode expanded="false"/>
<IndexNode expanded="false"/>
</IndexNode>
</IndexNode>
<IndexNode expanded="false"/>
<IndexNode expanded="true"/>
</IndexNode>
</cpnet>
</workspaceElements>
```

Figure 17: WST Tools

to PTCPNs. As WS-CDL is an XML-based language, and PTCPNs are supported by CPNTools and are also represented by XML files, we have used XSLT stylesheets to transform the WS-CDL document to obtain another XML document representing the PTCPN in a format supported by CPNTools. These XSL stylesheets are created using the XSLT editor. The obtained XML document can be visualized, simulated and verified with CPNTools.

WST Tool is available for Windows/Linux systems under virtual machine Java, at *http://www.dsi.uclm.es/retics/WST/* .

WST Tool has a very simple and intuitive interface, in which the different translations supported are clearly visible by their corresponding tabs. Thus, the user only needs to click on the appropriate tab in order to perform a specific translation. Thus, we now have a new tab, called "WS-CDL2CPNTools", which is used to transform the WS-CDL specification into a PTCPN. Figure 17 shows a picture of the WST Tool and the transformation from a WS-CDL document to a PTCPN.

Let us now see a brief description of the new elements that have been added:

- In the **WS-CDL Textbox**, we can find the corresponding WS-CDL specification that can be introduced by hand or by using the automatic transformation from UML. This specification will be loaded by using the option "Open WS-CDL File" in the File menu. A dialog window will be shown to the user asking him to select the document to be opened. If the file is not valid, an error message will be displayed on the screen.
- In the **CPNTools Textbox**, after clicking on the button "Transform", the corresponding Petri Net XML specification is shown. To save this specification, the user must click on the "Save Colored Petri Net File" option in the CPN Tools menu. A dialog window will be shown to the user to choose the destination folder.

In brief, we have two new buttons on the screen:

- **The Transform button** allows the user to generate the corresponding PTCPN. The result will be automatically displayed in the CPN Tools Textbox after a few seconds. If the WS-CDL Textbox is empty, pressing the Transform button will have no effect.
- **The Clear button** allows the user to clean the contents of both text boxes, the WS-CDL Textbox and the PTCPN Textbox. If both text boxes are empty, pressing on this button will have no effect.

## 5.2 XSLT Architecture

The XSLT transformation sheets (*eXtensible Stylesheets Language/Transform*) are a W3C declarative language to transform XML documents into other XML documents or to some other kind of documents. The XSLT stylesheets are widely used, as an easy way to apply transformation rules to a source document in order to obtain the corresponding output documents. Nowadays, XSLT is widely recommended in web edition area, due to its ability to generate HTML or XHTML sheets.

For making the transformation of XML documents, we have used XSLT stylesheets, which allow the programmer to convert the original one in two ways: On the one hand, the programmer can manipulate the contents of the document to organize them without changing the document format, whereas, on the other hand, the programmer can use XSLT sheets to transform the contents into other different formats (see Figure 18).

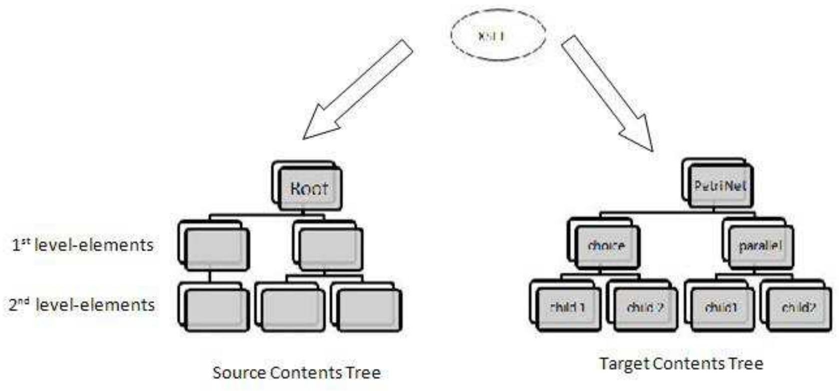


Figure 18: XML transformation by using XSLT

We have then defined a number of rules to extract the PTCPN elements from the choreography defined in the well-known choreography language, WS-CDL. Thus, our tool, WST, is used to carry out this transformation in an automatic way, presenting to the user a *.cpn file*, which can be opened with CPNTools. After doing this, the user can analyze and verify the model by using the features of the CPNTools (see Figure 19).

In Figure 20, we can see the Java code that we have used to transform the input XML document into the corresponding XML output document, by using four XSLT templates called *cpn1, cpn2, cpn3 and cpn4*, and the output document is a file called "final.xml".

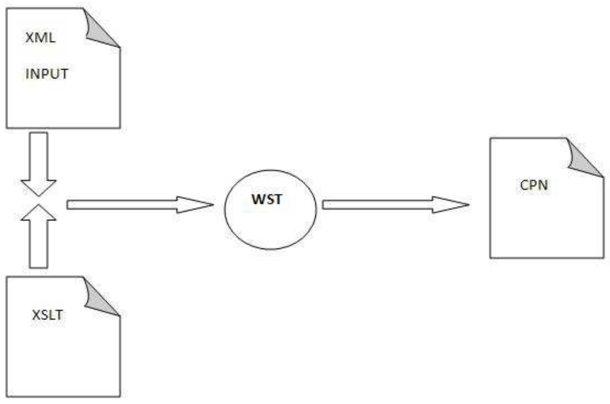


Figure 19: Application of XSL rules to obtain the cpn file

```
TransformerFactory tFactory = TransformerFactory.newInstance();
Transformer transformer1 =
                              tFactory.newTransformer(new StreamSource(
                              "XSLFiles/cpn1.xsl"));

transformer1.transform(new StreamSource(fichero),
                                       new StreamResult(new FileOutputStream(
                             "XSLFiles/salida1.xml")));

TransformerFactory t2Factory = TransformerFactory.newInstance();
Transformer transformer2 =
                        t2Factory.newTransformer(new StreamSource(
                        "XSLFiles/cpn2.xsl"));
transformer2.transform(new StreamSource("XSLFiles/salida1.xml"),
                                       new StreamResult(new
                        FileOutputStream("XSLFiles/salida2.xml")));
TransformerFactory t3Factory = TransformerFactory.newInstance();
Transformer transformer3 =
                        t3Factory.newTransformer(new StreamSource(
                        "XSLFiles/cpn3.xsl"));
transformer3.transform(new StreamSource("XSLFiles/salida2.xml"),
                                       new StreamResult(new
                        FileOutputStream("XSLFiles/final.xml")));

FileInputStream fr = new FileInputStream("XSLFiles/final.xml");
```

Figure 20: Example of the XSLT template

## 5.3 Rules

From a Document Type Definition (DTD) file, one can describe the XML document structure: labels, data types, and so on. The XML document must be validated to check if it satisfies the DTD structure defined previously. Besides, the systems that exchange XML documents must agree in this DTD structure and validate DTD received documents based on the agreed structure.

The technology used in this work is Documents Object Model (DOM). This specification is a W3C standard independent of programming language. This technology allows us to manipulate multiple sections of a document at the same time, and it is not necessary to read all the document again to work on a particular area of the document. This technology also transforms XML documents in a hierarchical tree. Thus, DOM allows us to manipulate the XML document, for instance, to add items, to delete items, etc. The library that we have used in this work is org.w3c.dom.*;

As mentioned in Section 5.2, the templates used to generate the output XML document have been XSLT. A stylesheet XSLT must be a well-formed XML document stored with the extension *.xsl*.

### 5.3.1 The structure of the stylesheet

The document starts with the instruction $\langle$ *?xml versión* = '1.0'?$\rangle$ . The element root is a stylesheet, which contains all other elements. In an XSLT stylesheet, the name of reserved elements by the specification comes from the same namespace, so they must be written preceded by the appropriate alias that must point to the URL: http://www.w3c.org/1999/XSL/Transform. In Figure 21 we show the structure of the XSLT document.
Once we have located the initial and final mark of the root element "xsl:stylesheet", we define the transformation rules:

- Each rule is defined by an "xsl:template".
- In the rules, we indicate those elements of the XML document that will be transformed.
- The rules also indicate how each element must be transformed.
- Each rule is applied to all elements of the XML document.
- In the XSLT rules, between their initial and final marks, one can include:
  - Text to be written literally in the output document.
  - Marks that are added to the XML output document.
  - Reserved elements to perform an action such as retrieving the value of an item, sorting results, calling other rules of the stylesheet, etc.

In Figure 22, we depict what happens when we find a label whose name is "interaction" in an XML input document.

```
 <?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output indent="yes" />
  - <xsl:template match="/">
 <workspaceElements>
   <generator tool="CPN Tools" version="2.9.12" format="6" />
  <cpnet>
  ...
<page id="ID6">
<template>
<xsl:for-each select="//choreography">
<xsl:for-each select="child::*">
<xsl:if test="(name()='noAction')">
   <xsl:call-template name="noAct" />
   </xsl:if>
<xsl:if test="(name()='fail')">
   <xsl:call-template name="fail" />
   </xsl:if>
  <xsl:if test="(name()='interaction')">
<xsl:call-template name="int">
  <xsl:with-param name="nombre_proceso">
   <xsl:value-of select="$nombre_proceso"/>
   </xsl:with-param>
   </xsl:call-template>
   </xsl:if>
<xsl:if test="(name()='workunit')">
  <xsl:call-template name="work">
  <xsl:with-param name="nombre_proceso">
   <xsl:value-of select="$nombre_proceso" />
   </xsl:with-param>
   </xsl:call-template>
   </xsl:if>
<xsl:for-each select="//choreography">
<xsl:for-each select="child::*">
  <xsl:if test="(name()='noAction')">
   <xsl:call-template name="noAct_trans" />
   </xsl:if>
<xsl:if test="(name()='fail')">
   <xsl:call-template name="fail_trans" />
   </xsl:if>
   <xsl:if test="(name()='interaction')">
   <xsl:call-template name="inter_trans">
   <xsl:with-param name="nombre_proceso">
   <xsl:value-of select="$nombre_proceso" />
   </xsl:with-param>
   </xsl:call-template>
   </xsl:if>
   <xsl:if test="name()='workunit'">
   <xsl:call-template name="work_trans">
   <xsl:with-param name="nombre_proceso">
   <xsl:value-of select="$nombre_proceso" />
   </xsl:with-param>
   </xsl:call-template>
   </xsl:if>
   ....
   </xsl:for-each>
   </template>
  </page>
- <fin>
....
</fin>
....
</cpnet>
 </workspaceElements>
```

Figure 21: Illustration of an XSLT template

```
<xsl:template name="int">
   <xsl:if test="descendant::record">
<xsl:call-template name="assign">
  <xsl:with-param name="nombre_assign">
   <xsl:value-of select="record/target/@variable" />
   </xsl:with-param>
  <xsl:with-param name="num_assign">
   <xsl:value-of select="count(preceding::assign)" />
   </xsl:with-param>
   </xsl:call-template>
    </xsl:if>
   <!--Add initial place-->
<xsl:element name="place">
   </xsl:element>
 <!-- If there is timeout-->
 <xsl:if test="priority/timeout">
   <xsl:call-template name="aux1" />
   </xsl:if>
   <xsl:if test="not(priority/timeout)">
     <!--Add transition of interaction-->
    <xsl:element name="trans">
   </xsl:element>
   </xsl:if>
    <!--Add transition timeout-->
  <xsl:element name="trans">
   </xsl:element>
   <!--Add transition fail-->
  <xsl:element name="trans">
   </xsl:element>
   <!--Add place v1r1-->
  <xsl:element name="place">
  </xsl:element>
   <!--Add place v2r2-->
  <xsl:element name="place">
  </xsl:element>
   <!--Add error place-->
    <xsl:element name="place">
    </xsl:element>
     <!-- Add final place-->
    <xsl:element name="place">
    </xsl:element>
  </xsl:template>
```

Figure 22: Basic Interaction XSLT rule

### 5.3.2 Specific rules

Next, we define some specific rules we have developed in order to carry out the transformation between WS-CDL and PTCPN, specifically for the activities: workunit, choice and parallel:

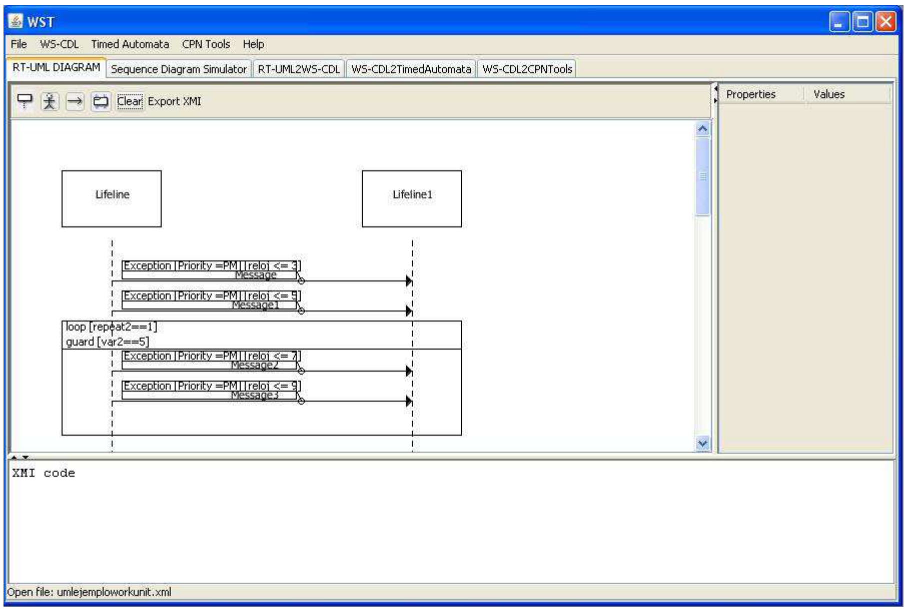

WST
File WS-CDL Timed Automata CPN Tools Help
RT-UML DIAGRAM
Sequence Diagram Simulator
RT-UML2WS-CDL
WS-CDL2TimedAutomata
WS-CDL2CPNTools
Clear
Export XMI
Properties
Values
Lifeline
Lifeline1
Exception |Priority =PM||reloj <= 3|
Message
Exception |Priority =PM||reloj <= 5|
Message1
loop [repeat2==1]
guard [var2==5]
Exception |Priority =PM||reloj <= 7|
Message2
Exception |Priority =PM||reloj <= 9|
Message3
XMI code
Open file: umlejemploworkunit.xml

Figure 23: UML Diagram for a Workunit activity

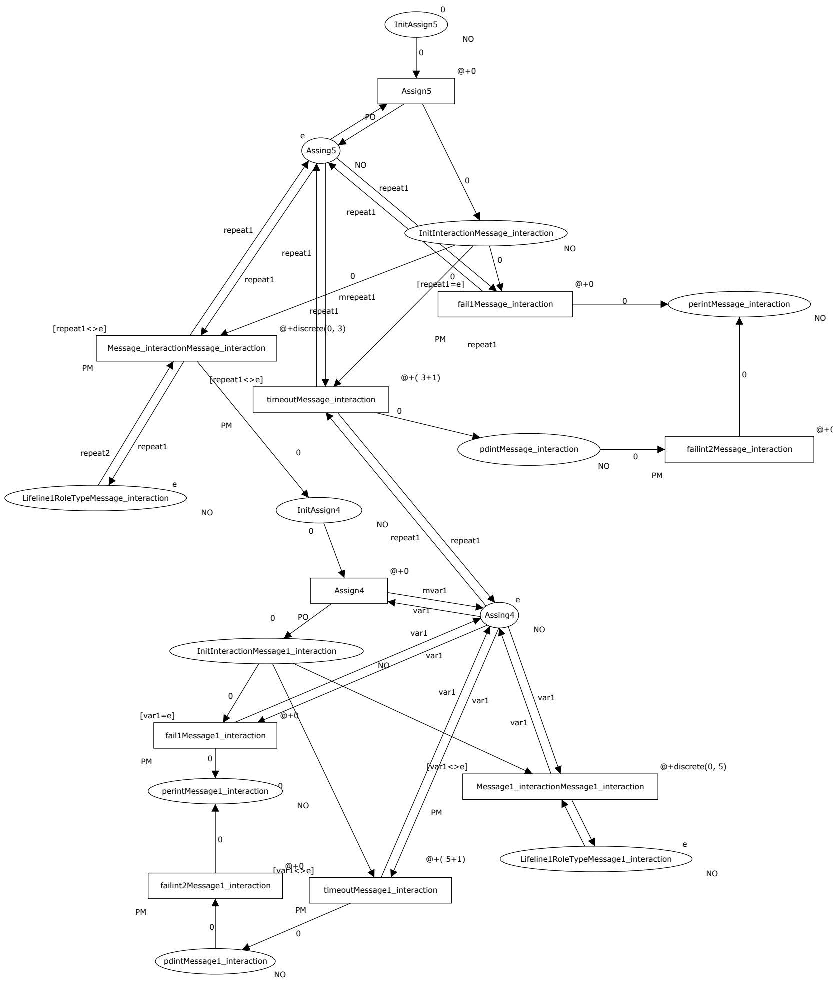

InitAssign5
Assign5
Assing5
InitInteractionMessage_interaction
fail1Message_interaction
perintMessage_interaction
Message_interactionMessage_interaction
timeoutMessage_interaction
pdintMessage_interaction
failint2Message_interaction
Lifeline1RoleTypeMessage_interaction
InitAssign4
Assign4
Assing4
InitInteractionMessage1_interaction
fail1Message1_interaction
perintMessage1_interaction
Message1_interactionMessage1_interaction
Lifeline1RoleTypeMessage1_interaction
failint2Message1_interaction
timeoutMessage1_interaction
pdintMessage1_interaction
repeat1
repeat2
[repeat1<>e]
[repeat1=e]
mrepeat1
@+discrete(0, 3)
@+( 3+1)
@+0
mvar1
var1
[var1=e]
[var1<>e]
@+discrete(0, 5)
@+( 5+1)
NO
PM
PO
0
e

Figure 24: Workunit ParI

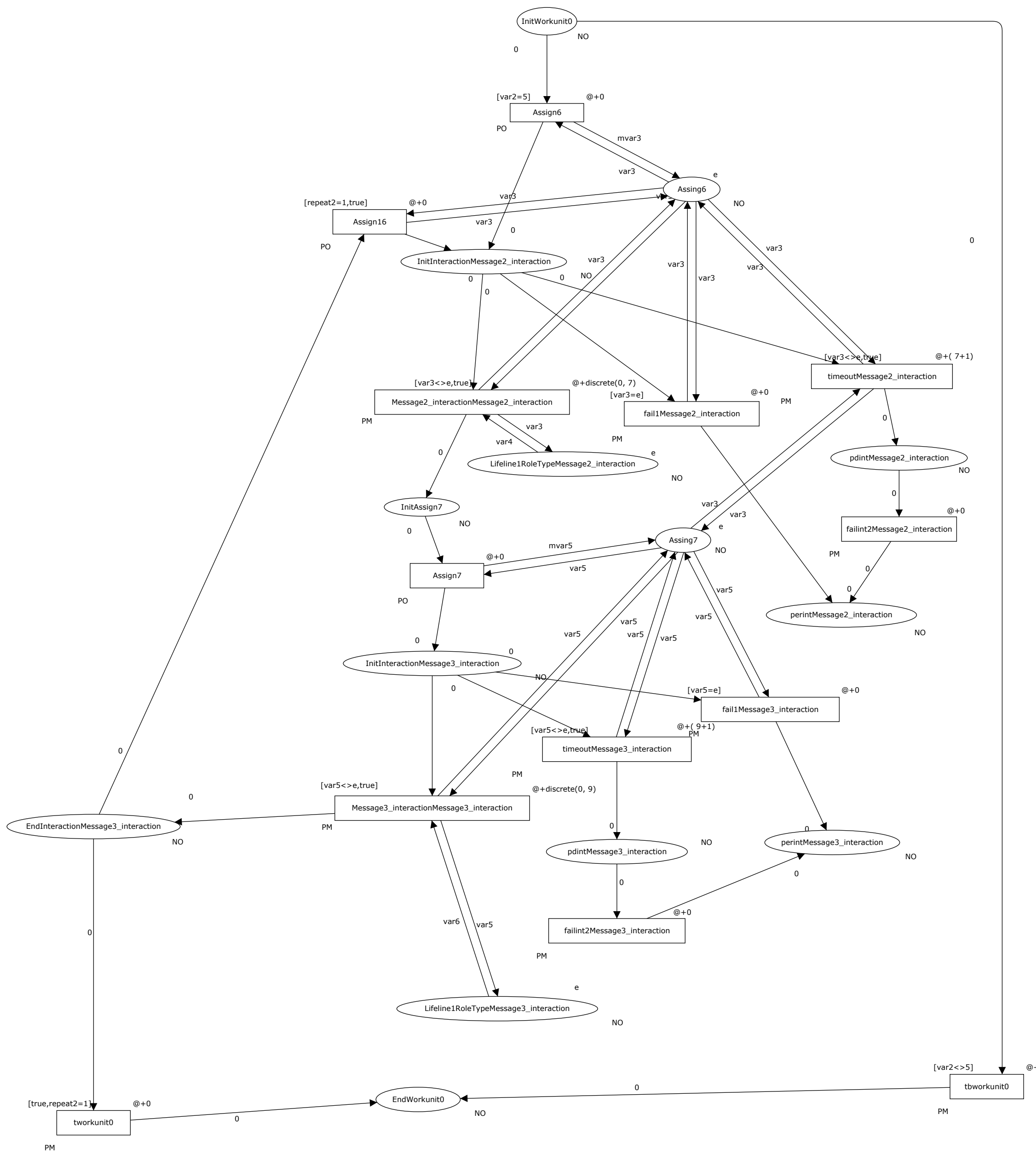
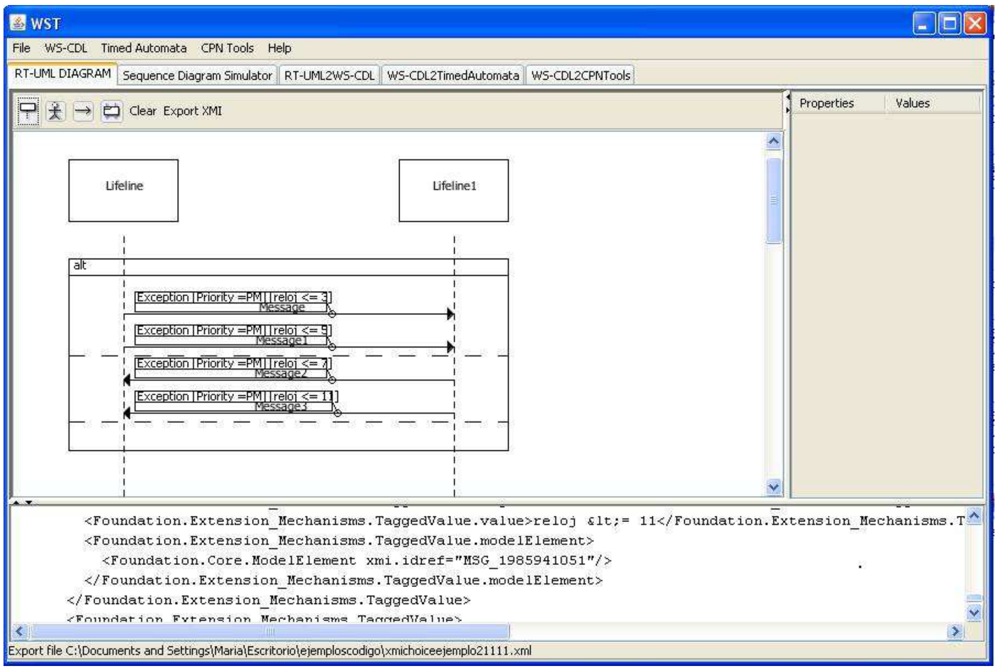

InitWorkunit0
NO
0
[var2=5]
@+0
Assign6
PO
mvar3
var3
e
Assing6
NO
[repeat2=1,true]
@+0
Assign16
PO
var3
InitInteractionMessage2_interaction
NO
0
[var3<>e,true]
@+discrete(0, 7)
Message2_interactionMessage2_interaction
PM
var4
Lifeline1RoleTypeMessage2_interaction
[var3=e]
fail1Message2_interaction
[var3<>e,true]
@+( 7+1)
timeoutMessage2_interaction
pdintMessage2_interaction
failint2Message2_interaction
perintMessage2_interaction
InitAssign7
Assing7
mvar5
var5
Assign7
InitInteractionMessage3_interaction
[var5=e]
fail1Message3_interaction
[var5<>e,true]
@+( 9+1)
timeoutMessage3_interaction
[var5<>e,true]
@+discrete(0, 9)
Message3_interactionMessage3_interaction
EndInteractionMessage3_interaction
pdintMessage3_interaction
perintMessage3_interaction
failint2Message3_interaction
var6
Lifeline1RoleTypeMessage3_interaction
[var2<>5]
tbworkunit0
EndWorkunit0
[true,repeat2=1]
tworkunit0

Figure 25: Workunit PartII

### 5.3.3 workunit

- A “workunit” is essentially a conditional and looping constructor combined in one enclosing. To design a “workunit” activity, we need at least two variables: the first variable, guard, is used to check the access into the “workunit” activity, and the second variable, repeat, is used to check the access of repetitions of the “workunit” activity. To assing a value to each of these two variables we use an “assign” activity and an “interaction” activity.

- Once we know the value of the variables for the workunit’s entry, we are allowed to continue with the internal workunit’s activities. The “workunit” is composed of two interactions whose function is to exchange the value of the variables. As we can see, the value of the varible “repeat” is false, and if the variable “repeat” is eve true the internal activity of the workunit until be repeated (see Fig. 23, 24, 25))


WST
File WS-CDL Timed Automata CPN Tools Help
RT-UML DIAGRAM
Sequence Diagram Simulator
RT-UML2WS-CDL
WS-CDL2TimedAutomata
WS-CDL2CPNTools
Clear Export XMI
Properties
Values
Lifeline
Lifeline1
alt
Exception | Priority =PM | [reloj <= 3]
Message
Exception | Priority =PM | [reloj <= 9]
Message1
Exception | Priority =PM | [reloj <= 7]
Message2
Exception | Priority =PM | [reloj <= 11]
Message3

```
    <Foundation.Extension_Mechanisms.TaggedValue.value>reloj &lt;= 11</Foundation.Extension_Mechanisms.T
    <Foundation.Extension_Mechanisms.TaggedValue.modelElement>
      <Foundation.Core.ModelElement xmi.idref="MSG_1985941051"/>
    </Foundation.Extension_Mechanisms.TaggedValue.modelElement>
  </Foundation.Extension_Mechanisms.TaggedValue>
  <Foundation.Extension_Mechanisms.TaggedValue>
```

Export file C:\Documents and Settings\Maria\Escritorio\ejemploscodigo\xmichoiceejemplo21111.xml

Figure 26: UML Diagram for a Choice activity

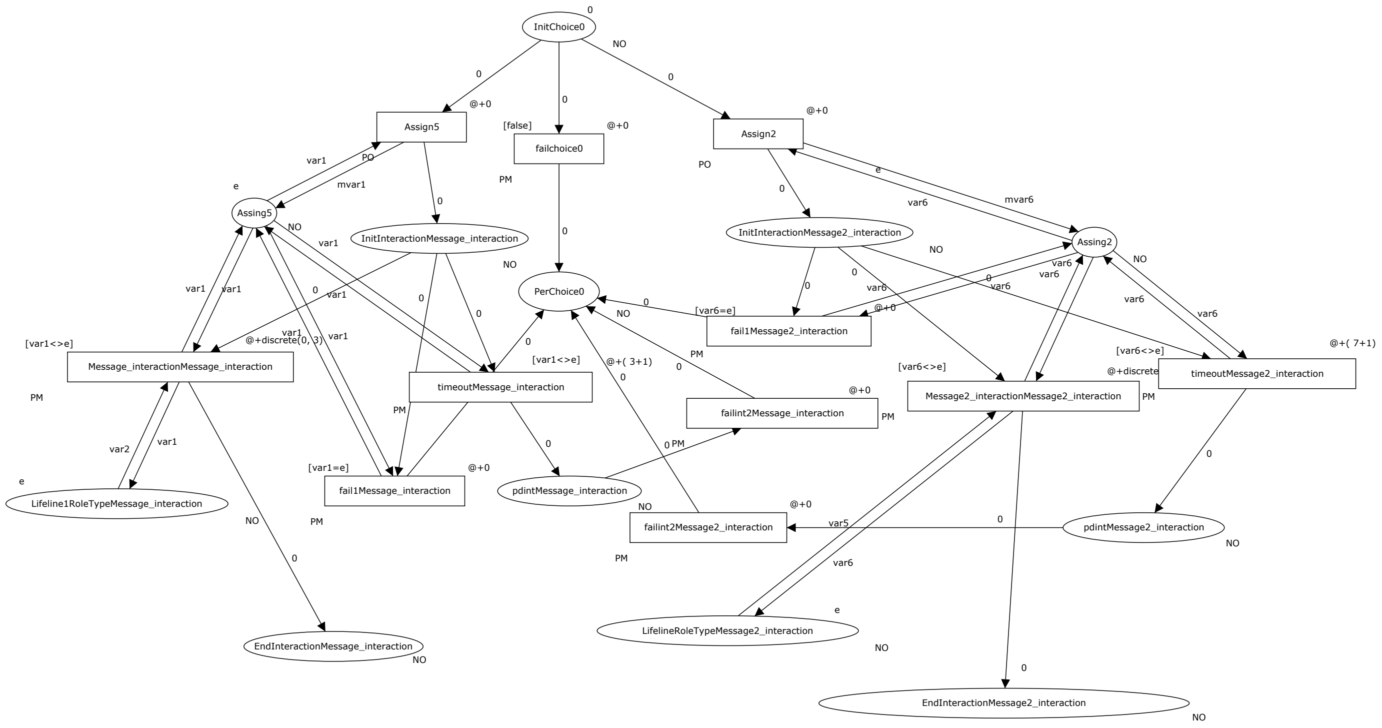

InitChoice0
Assign5
failchoice0
Assign2
Assing5
InitInteractionMessage_interaction
PerChoice0
InitInteractionMessage2_interaction
Assing2
fail1Message2_interaction
Message_interactionMessage_interaction
timeoutMessage_interaction
failint2Message_interaction
Message2_interactionMessage2_interaction
timeoutMessage2_interaction
Lifeline1RoleTypeMessage_interaction
fail1Message_interaction
pdintMessage_interaction
failint2Message2_interaction
pdintMessage2_interaction
LifelineRoleTypeMessage2_interaction
EndInteractionMessage_interaction
EndInteractionMessage2_interaction
[false]
[var1<>e]
[var1=e]
[var6=e]
[var6<>e]
@+0
@+discrete(0, 3)
@+( 3+1)
@+( 7+1)
@+discrete
PO
PM
NO
var1
mvar1
var2
var5
var6
mvar6
e
0

Figure 27: Choice Part I

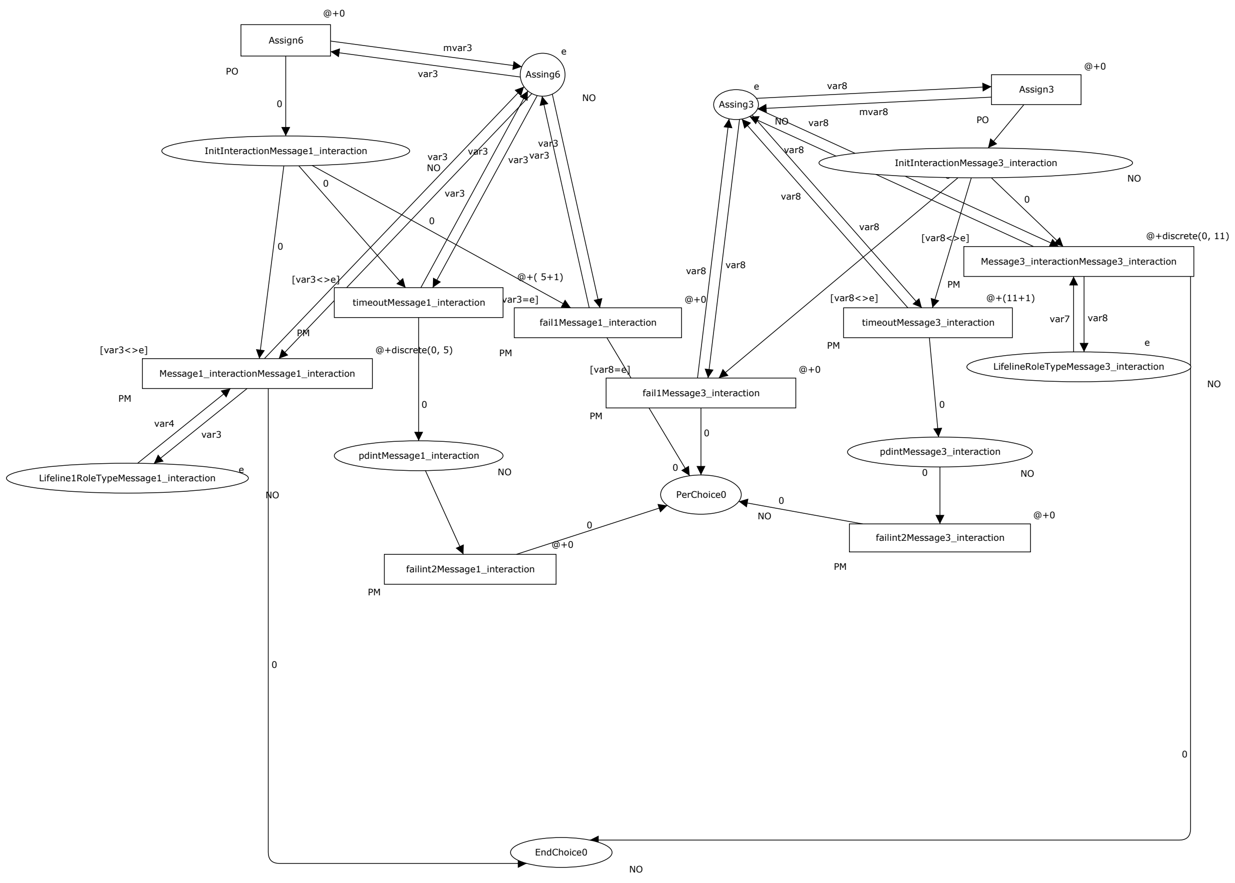


Figure 28: Choice Part II

### 5.3.4 choice

- The “choice” activity allows us to execute one of its descendants independently from any situation.
  In this example, we can see two options, both of them are composed of two “Interaction” activities.
- A situation we must deal with is that of places that are merged or joined in the translation process, according to the semantics described in Section 4.3. For instance, in the choice operator, we have that both final places of the internal activities are collapsed in a single one, so we just need one place for the final place of the choice. This aspect must be considered in the XML output document, and therefore in the corresponding XSLT rule.
- The error places of the different activities in a choice are also collapsed in a single error place. This aspect is also considered in the XSLT rule (see Fig. **??**(see Fig. 26, 27, 28))

### 5.3.5 parallel

- We must link the initial transition of a parallel activity with all the initial places of the activities that are to be run in parallel. Likewise, the final places of these activities are also connected with the final place of the parallel activity by means of a transition representing the end of all of these activities.

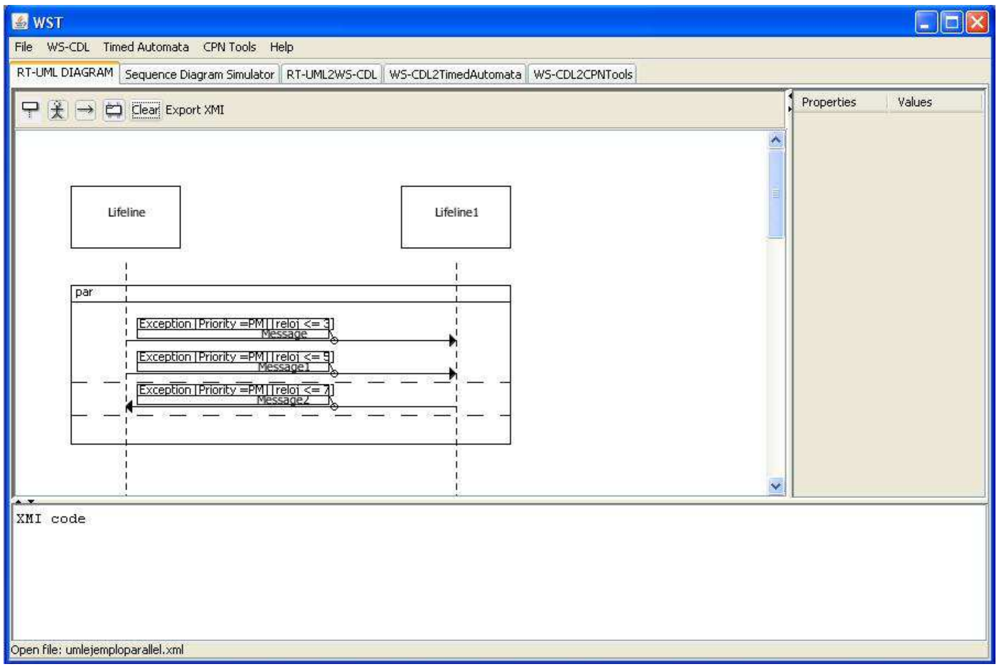


Figure 29: UML Diagram for a Parallel activity

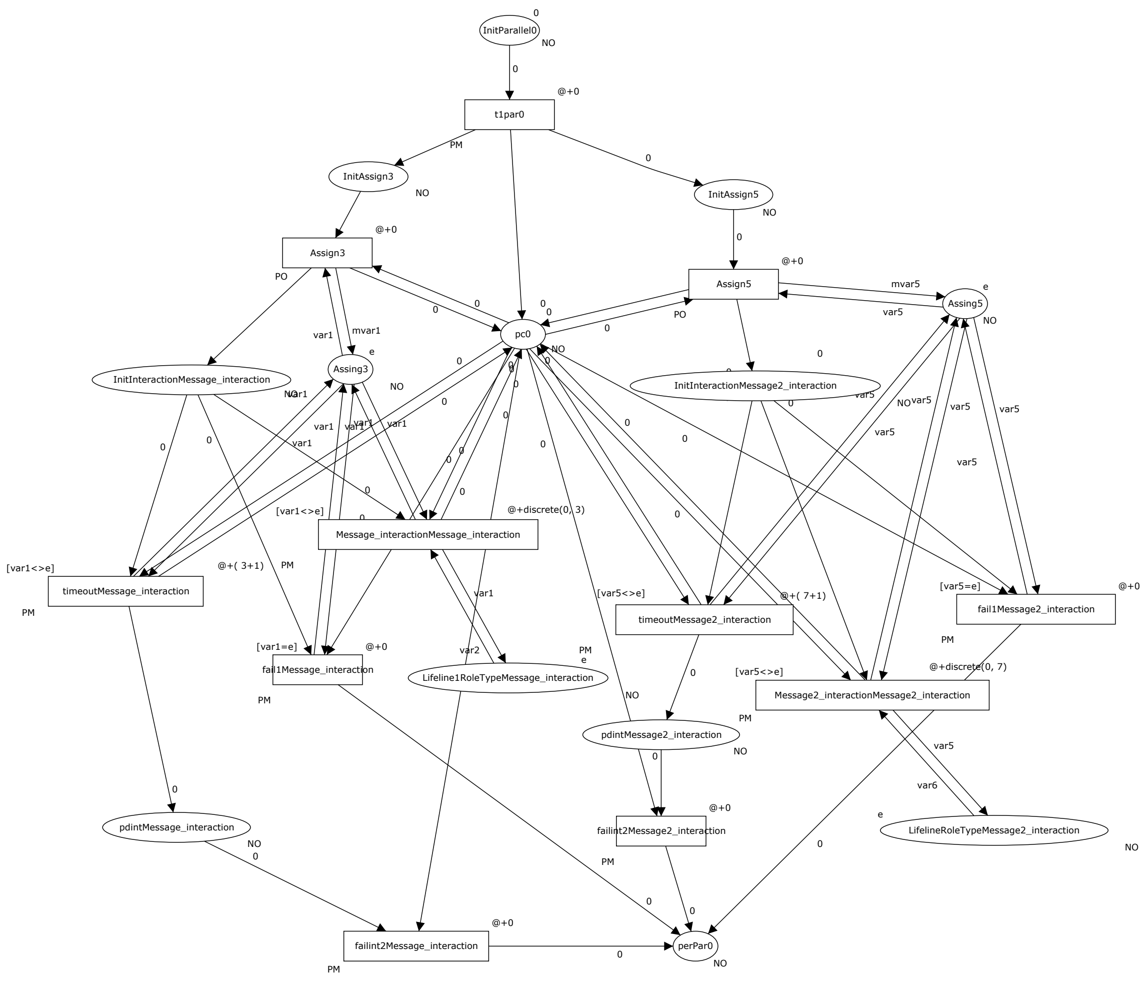

InitParallel0
NO
t1par0
@+0
PM
InitAssign3
InitAssign5
Assign3
Assign5
PO
pc0
Assing3
Assing5
mvar1
var1
mvar5
var5
e
InitInteractionMessage_interaction
InitInteractionMessage2_interaction
Message_interactionMessage_interaction
@+discrete(0, 3)
[var1<>e]
[var1<>e]
@+( 3+1)
timeoutMessage_interaction
[var1=e]
fail1Message_interaction
Lifeline1RoleTypeMessage_interaction
var2
[var5<>e]
timeoutMessage2_interaction
@+( 7+1)
[var5=e]
fail1Message2_interaction
@+discrete(0, 7)
[var5<>e]
Message2_interactionMessage2_interaction
var6
pdintMessage2_interaction
failint2Message2_interaction
LifelineRoleTypeMessage2_interaction
pdintMessage_interaction
failint2Message_interaction
perPar0
0

Figure 30: Parallel Part I

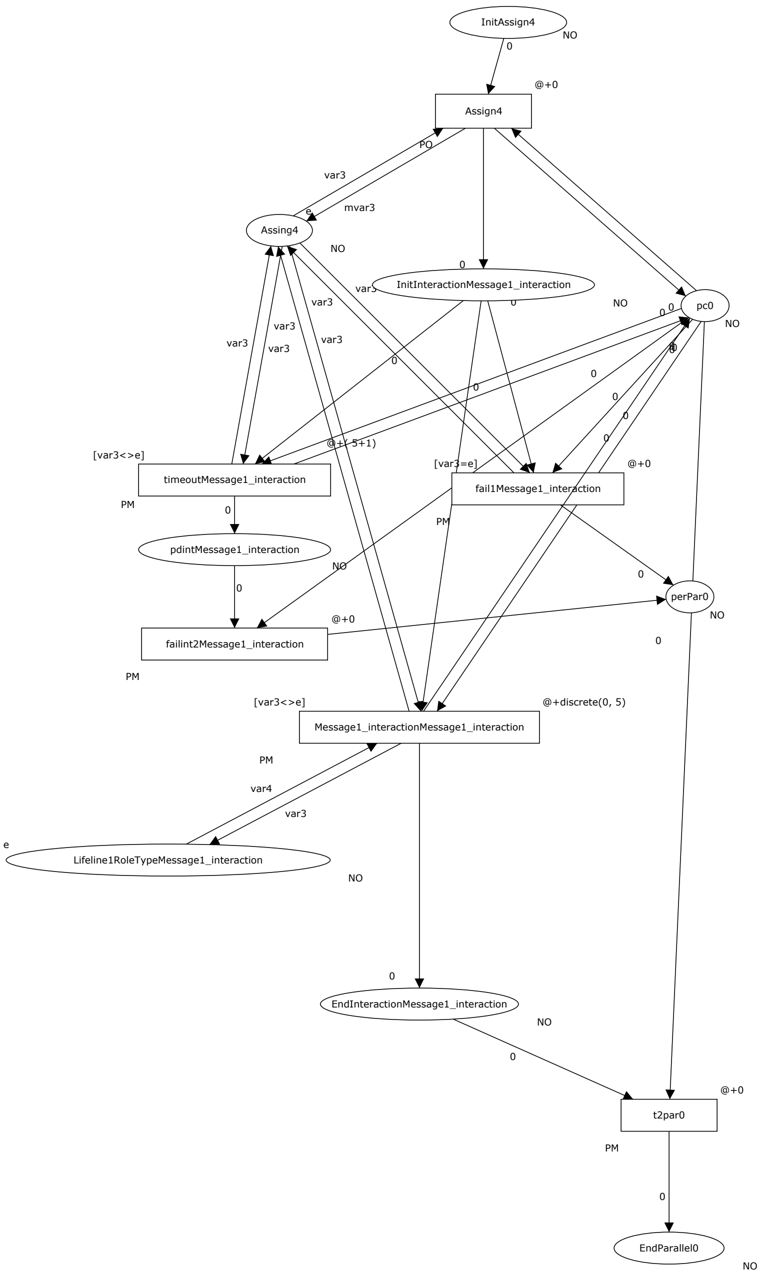

InitAssign4
NO
0
@+0
Assign4
PO
var3
mvar3
e
Assing4
NO
0
InitInteractionMessage1_interaction
var3
var3
var3
var3
var3
var3
0
0
NO
pc0
NO
0
0
0
0
0
0
0
[var3<>e]
@+( 5+1)
timeoutMessage1_interaction
PM
[var3=e]
@+0
fail1Message1_interaction
PM
0
pdintMessage1_interaction
NO
0
0
perPar0
NO
@+0
failint2Message1_interaction
PM
0
[var3<>e]
@+discrete(0, 5)
Message1_interactionMessage1_interaction
PM
var4
var3
e
Lifeline1RoleTypeMessage1_interaction
NO
0
EndInteractionMessage1_interaction
NO
0
@+0
t2par0
PM
0
EndParallel0
NO

Figure 31: Parallel Part II

– For the error places of the activities inside a parallel, we collapse these error places into a simple one, and we also include a new place, named Pc, which controls the parallel execution, to arrest it in the case of a failure (see Fig. 29, 30, 31))

### 5.4 Working environment

WST tool has been developed on a Windows PC machine, but it also works in Unix/Linux environments. We have checked it in some Ubuntu distributions.

## 6 Case Studies

In this section we consider two case studies that illustrate the benefits of our framework.

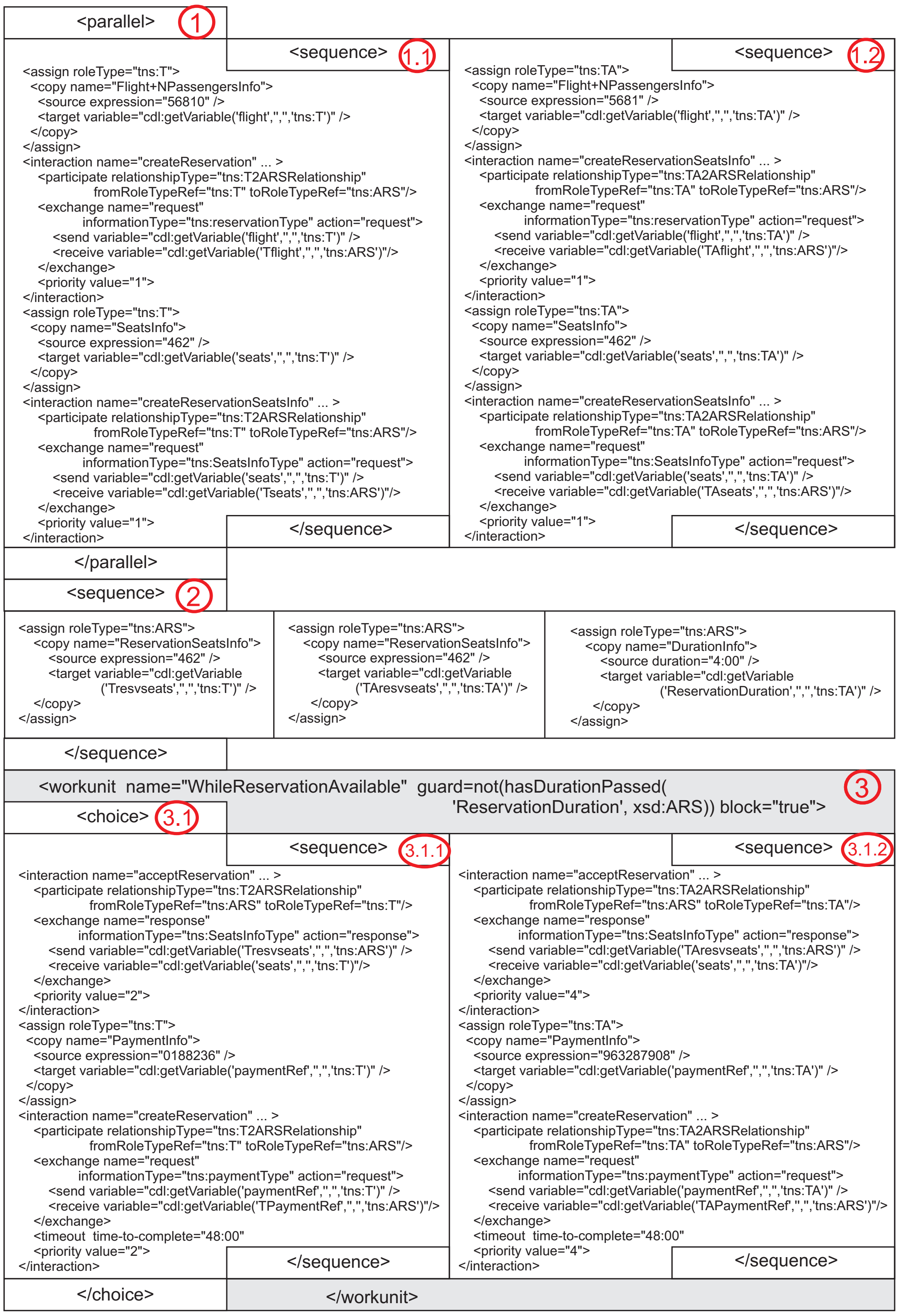
### 6.1 An Airline Ticket Reservation System

We consider an airline ticket reservation system (ATRS), which consists of three role types: Traveler (T), Travel Agent (A) and Airline Reservation System (R). The system works as follows: the Airline Reservation System receives requests from travelers and travel agents to reserve seats. Travelers have higher priority, i.e. travelers' requests are served first in the event of a conflict. Thus, $R$ receives a trip request for a specific date and flight, to which it must respond with seat bookings (to simplify we assume there are free seats). We have the following timed restrictions: 4 hours must elapse after getting the information on available seats to make a reservation. Reservations are only valid for a period of 48 hours, which means that if they have not been confirmed and paid for in two days they are canceled, and the seats are released.

Figure 32 contains the relevant parts of a WS-CDL document describing this system, in which there are three numbered sections, which correspond to a parallel structure (T and A request the seat information in parallel), a sequence structure (the seat booking information is set), and a delayed workunit structure to delay the execution for 4 hours. This workunit (number 3 in the Figure) consists of a choice, whose first activities are the reservation interactions from T and A, which have different priorities (travelers have higher priority). The final part of both branches corresponds to the payment, for which an interaction activity has been introduced with an associated time-out.

Using the WST tool we have obtained the corresponding PTCPN from this WS-CDL document, depicted in Figures 33 (top) and 34 (bottom), which are connected by the common places $P6$ and $Perror$.

Looking at Fig. 33 we can see a transition $T1$ that forks both initial parallel activities, which correspond to two assign activities and two interactions activities. Both parallel activities join by means of transition $T2$. The firing of $T2$ activates the execution of the three sequential assign activities, which can be seen as three assign transitions in a row in Figure 34. After, at least their execution the delayed workunit starts, transition $tt$ captures the 4 hours delay, after which a token is generated on $P10$. The choice inside the workunit now appears. Notice that the initial transitions of both branches have been replicated, although in this case the replicas can never be executed, because the workunit is not repetitive. Finally, both parts terminate with the payment interactions, which have



```
<parallel>
<sequence>
<assign roleType="tns:T">
  <copy name="Flight+NPassengersInfo">
    <source expression="56810" />
    <target variable="cdl:getVariable('flight','','','tns:T')" />
  </copy>
</assign>
<interaction name="createReservation" ... >
    <participate relationshipType="tns:T2ARSRelationship"
               fromRoleTypeRef="tns:T" toRoleTypeRef="tns:ARS"/>
    <exchange name="request"
             informationType="tns:reservationType" action="request">
        <send variable="cdl:getVariable('flight','','','tns:T')" />
        <receive variable="cdl:getVariable('Tflight','','','tns:ARS')"/>
    </exchange>
    <priority value="1">
</interaction>
<assign roleType="tns:T">
  <copy name="SeatsInfo">
    <source expression="462" />
    <target variable="cdl:getVariable('seats','','','tns:T')" />
  </copy>
</assign>
<interaction name="createReservationSeatsInfo" ... >
    <participate relationshipType="tns:T2ARSRelationship"
               fromRoleTypeRef="tns:T" toRoleTypeRef="tns:ARS"/>
    <exchange name="request"
             informationType="tns:SeatsInfoType" action="request">
        <send variable="cdl:getVariable('seats','','','tns:T')" />
        <receive variable="cdl:getVariable('Tseats','','','tns:ARS')"/>
    </exchange>
    <priority value="1">
</interaction>
</sequence>

<sequence>
<assign roleType="tns:TA">
  <copy name="Flight+NPassengersInfo">
    <source expression="5681" />
    <target variable="cdl:getVariable('flight','','','tns:TA')" />
  </copy>
</assign>
<interaction name="createReservationSeatsInfo" ... >
    <participate relationshipType="tns:TA2ARSRelationship"
               fromRoleTypeRef="tns:TA" toRoleTypeRef="tns:ARS"/>
    <exchange name="request"
             informationType="tns:reservationType" action="request">
        <send variable="cdl:getVariable('flight','','','tns:TA')" />
        <receive variable="cdl:getVariable('TAflight','','','tns:ARS')"/>
    </exchange>
    <priority value="1">
</interaction>
<assign roleType="tns:TA">
  <copy name="SeatsInfo">
    <source expression="462" />
    <target variable="cdl:getVariable('seats','','','tns:TA')" />
  </copy>
</assign>
<interaction name="createReservationSeatsInfo" ... >
    <participate relationshipType="tns:TA2ARSRelationship"
               fromRoleTypeRef="tns:TA" toRoleTypeRef="tns:ARS"/>
    <exchange name="request"
             informationType="tns:SeatsInfoType" action="request">
        <send variable="cdl:getVariable('seats','','','tns:TA')" />
        <receive variable="cdl:getVariable('TAseats','','','tns:ARS')"/>
    </exchange>
    <priority value="1">
</interaction>
</sequence>
</parallel>

<sequence>
<assign roleType="tns:ARS">
    <copy name="ReservationSeatsInfo">
        <source expression="462" />
        <target variable="cdl:getVariable
                  ('Tresvseats','','','tns:T')" />
    </copy>
</assign>

<assign roleType="tns:ARS">
    <copy name="ReservationSeatsInfo">
        <source expression="462" />
        <target variable="cdl:getVariable
                  ('TAresvseats','','','tns:TA')" />
    </copy>
</assign>

<assign roleType="tns:ARS">
    <copy name="DurationInfo">
        <source duration="4:00" />
        <target variable="cdl:getVariable
                  ('ReservationDuration','','','tns:TA')" />
    </copy>
</assign>
</sequence>

<workunit  name="WhileReservationAvailable"  guard=not(hasDurationPassed(
                              'ReservationDuration', xsd:ARS)) block="true">
<choice>
<sequence>
<interaction name="acceptReservation" ... >
    <participate relationshipType="tns:T2ARSRelationship"
               fromRoleTypeRef="tns:ARS" toRoleTypeRef="tns:T"/>
    <exchange name="response"
             informationType="tns:SeatsInfoType" action="response">
        <send variable="cdl:getVariable('Tresvseats','','','tns:ARS')" />
        <receive variable="cdl:getVariable('seats','','','tns:T')"/>
    </exchange>
    <priority value="2">
</interaction>
<assign roleType="tns:T">
  <copy name="PaymentInfo">
    <source expression="0188236" />
    <target variable="cdl:getVariable('paymentRef','','','tns:T')" />
  </copy>
</assign>
<interaction name="createReservation" ... >
    <participate relationshipType="tns:T2ARSRelationship"
               fromRoleTypeRef="tns:T" toRoleTypeRef="tns:ARS"/>
    <exchange name="request"
             informationType="tns:paymentType" action="request">
        <send variable="cdl:getVariable('paymentRef','','','tns:T')" />
        <receive variable="cdl:getVariable('TPaymentRef','','','tns:ARS')"/>
    </exchange>
    <timeout  time-to-complete="48:00"
    <priority value="2">
</interaction>
</sequence>

<sequence>
<interaction name="acceptReservation" ... >
    <participate relationshipType="tns:TA2ARSRelationship"
               fromRoleTypeRef="tns:ARS" toRoleTypeRef="tns:TA"/>
    <exchange name="response"
             informationType="tns:SeatsInfoType" action="response">
        <send variable="cdl:getVariable('TAresvseats','','','tns:ARS')" />
        <receive variable="cdl:getVariable('seats','','','tns:TA')"/>
    </exchange>
    <priority value="4">
</interaction>
<assign roleType="tns:TA">
  <copy name="PaymentInfo">
    <source expression="963287908" />
    <target variable="cdl:getVariable('paymentRef','','','tns:TA')" />
  </copy>
</assign>
<interaction name="createReservation" ... >
    <participate relationshipType="tns:TA2ARSRelationship"
               fromRoleTypeRef="tns:TA" toRoleTypeRef="tns:ARS"/>
    <exchange name="request"
             informationType="tns:paymentType" action="request">
        <send variable="cdl:getVariable('paymentRef','','','tns:TA')" />
        <receive variable="cdl:getVariable('TAPaymentRef','','','tns:ARS')"/>
    </exchange>
    <timeout  time-to-complete="48:00"
    <priority value="4">
</interaction>
</sequence>
</choice>
</workunit>
```

Figure 32: WS-CDL description of the ATRS.

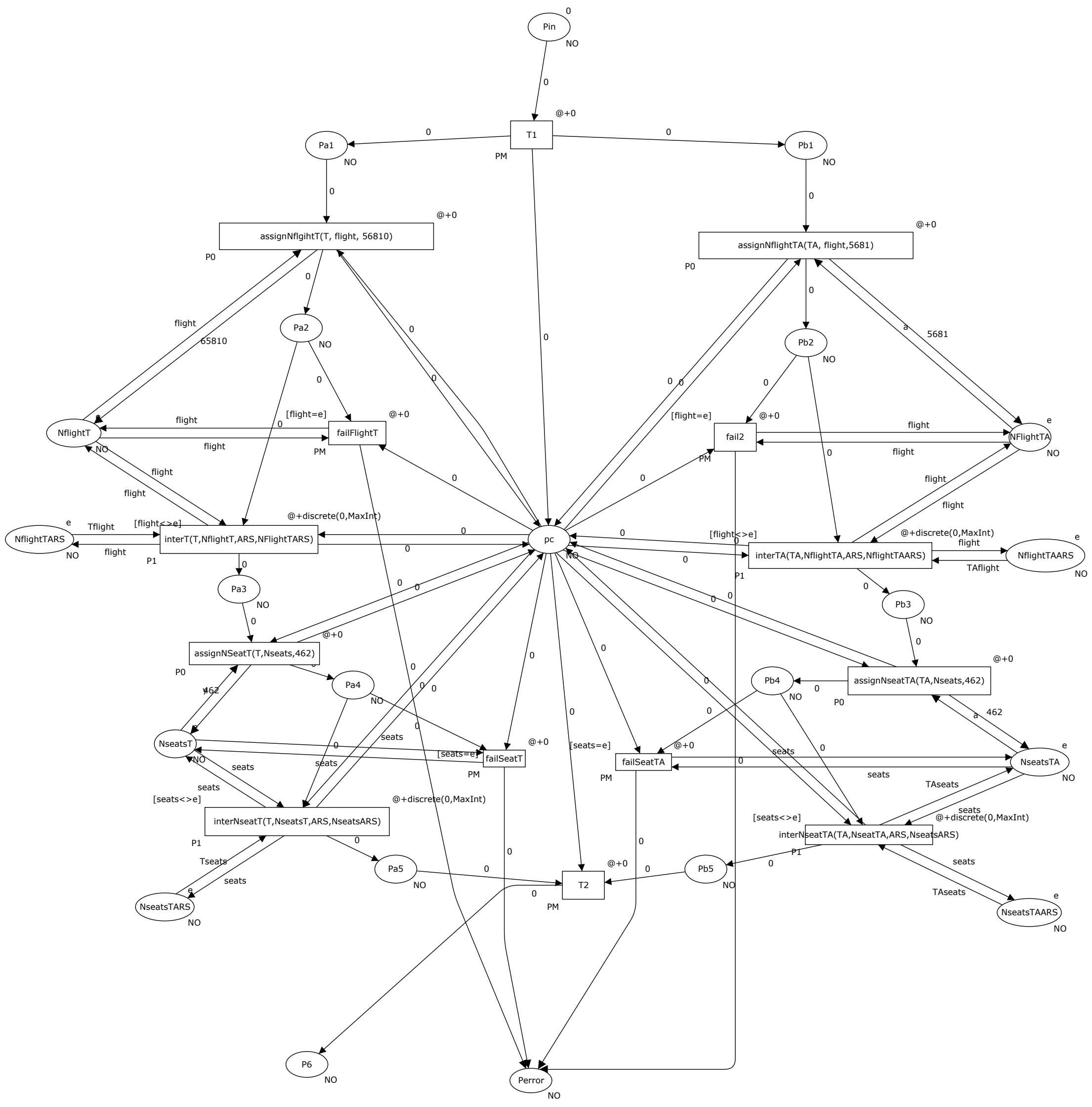

Pin
T1
Pa1
Pb1
assignNflgihtT(T, flight, 56810)
assignNflightTA(TA, flight,5681)
Pa2
Pb2
NflightT
failFlightT
fail2
NFlightTA
NflightTARS
interT(T,NflightT,ARS,NFlightTARS)
pc
interTA(TA,NflightTA,ARS,NflightTAARS)
NflightTAARS
Pa3
Pb3
assignNSeatT(T,Nseats,462)
assignNseatTA(TA,Nseats,462)
Pa4
Pb4
NseatsT
failSeatT
failSeatTA
NseatsTA
interNseatT(T,NseatsT,ARS,NseatsARS)
interNseatTA(TA,NseatTA,ARS,NseatsARS)
Pa5
T2
Pb5
NseatsTARS
NseatsTAARS
P6
Perror
@+discrete(0,MaxInt)
[flight=e]
[flight<>e]
[seats=e]
[seats<>e]

Figure 33: PTCPN corresponding to the airline reservation system (Part I)

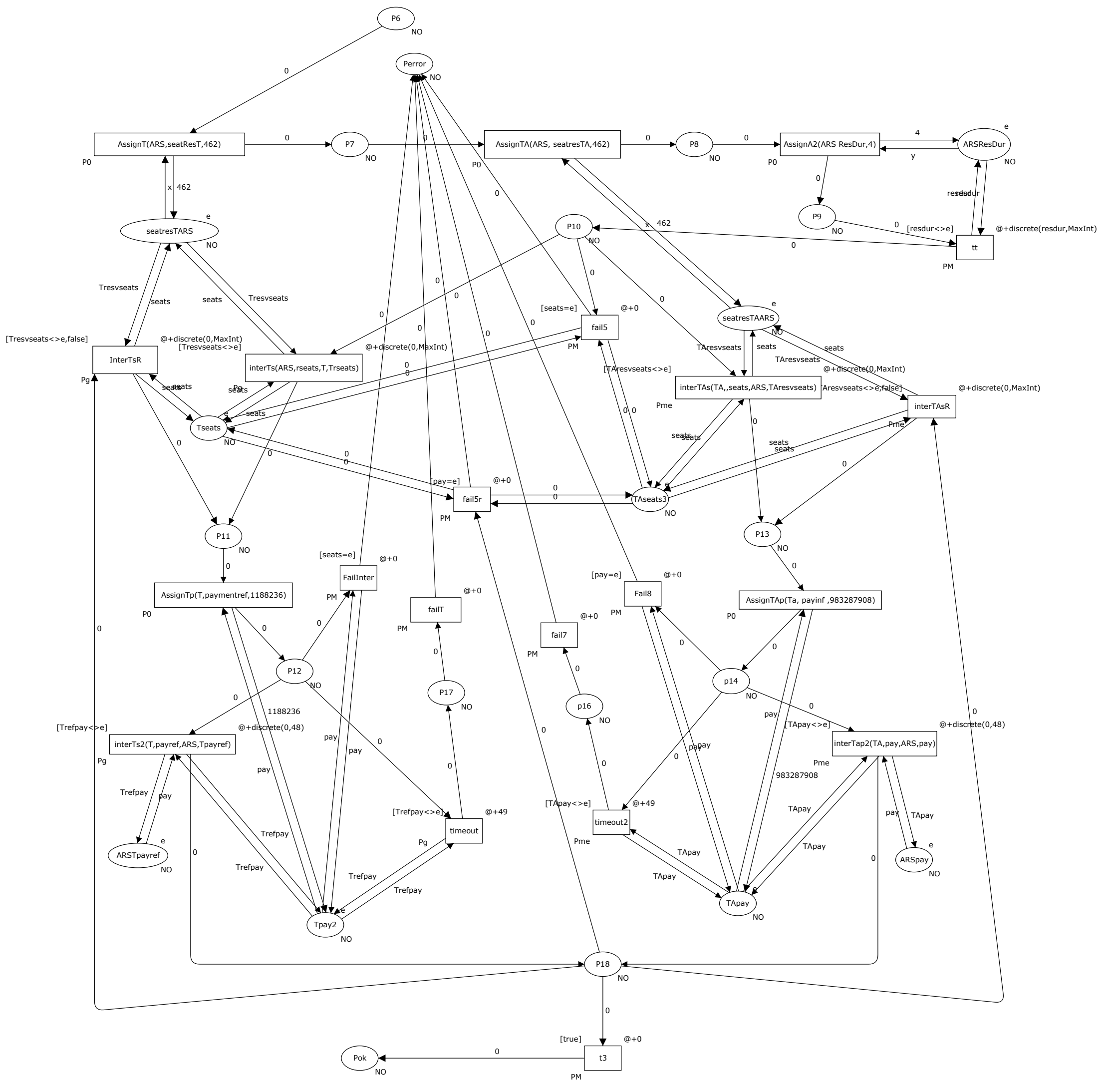


Figure 34: PTCPN corresponding to the airline reservation system (Part II)

an associated time-out, and transition $T3$ is the final transition of the enclosing delayed workunit.

### 6.1.1 Verification and Validation

The obtained PTCPN can be verified and validated using CPN Tools. Validation was performed by means of the CPN simulator engine. We concluded that the system always terminates correctly ($Pok$ marked) or incorrectly when the payment information has not been received in time ($Perror$ marked). We also concluded from simulations that the travel agent's requests could not be served, due to their lower priority, since both requests were made in parallel in this specific choreography.

CPN Tools can also be used to verify the system, by constructing the state space graph (see Figure: 35), and obtaining the corresponding state space report. From this report we can deduce the following properties:

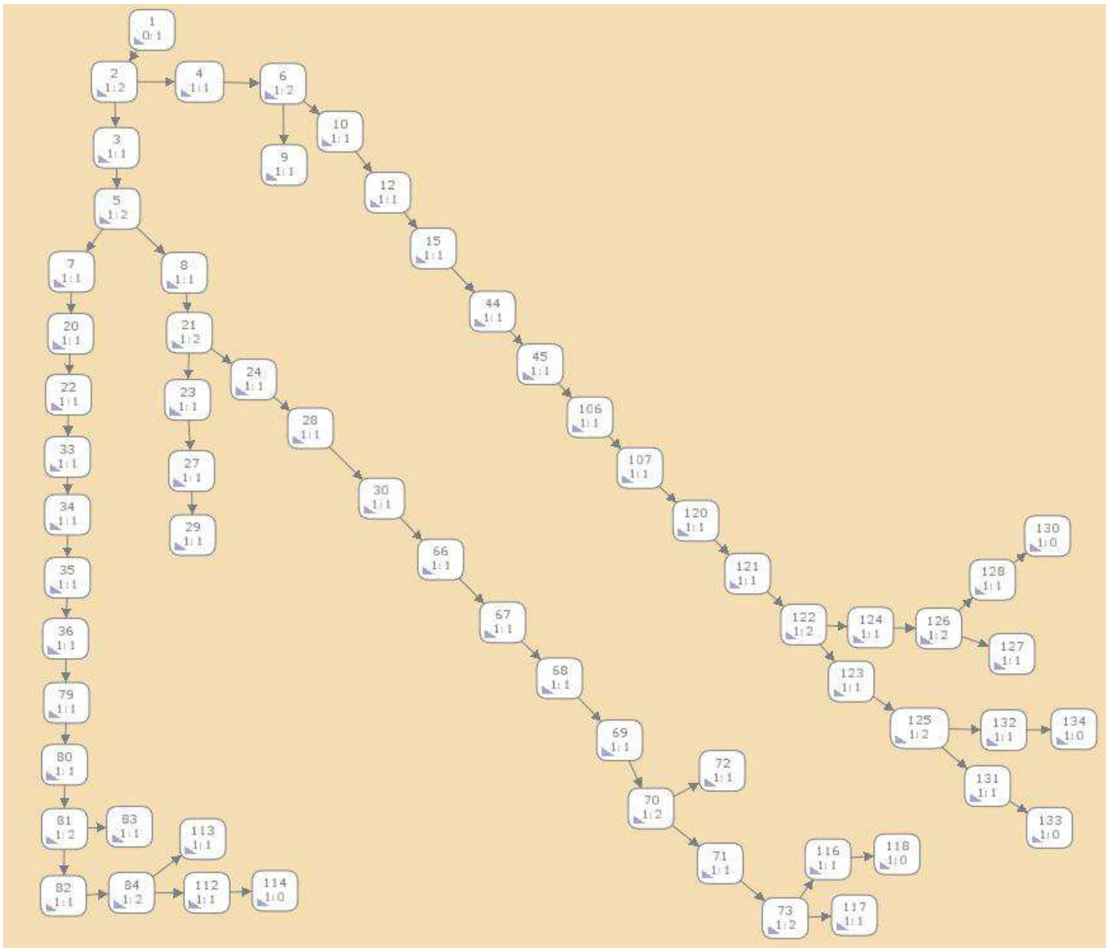


Figure 35: Parcial Diagram

- The PTCPN is 1-safe, i.e. no place can have more than one token at any reachable marking. There are also some places that are never marked, namely, $P13, P14$ and $P16$, which correspond to the part of the travel agent's request confirmation and payment, which is never executed, due to its lower priority.
- As expected, the initial marking is not a *home state*, because we have no way to return to it.

– From the *dead markings* that we obtain we conclude that the system execution always terminates in a final marking in which either *Pok* or *Perror* is marked. This can be interpreted in the sense that the reservation process either terminates correctly or the reservation is canceled in the event of a failure.

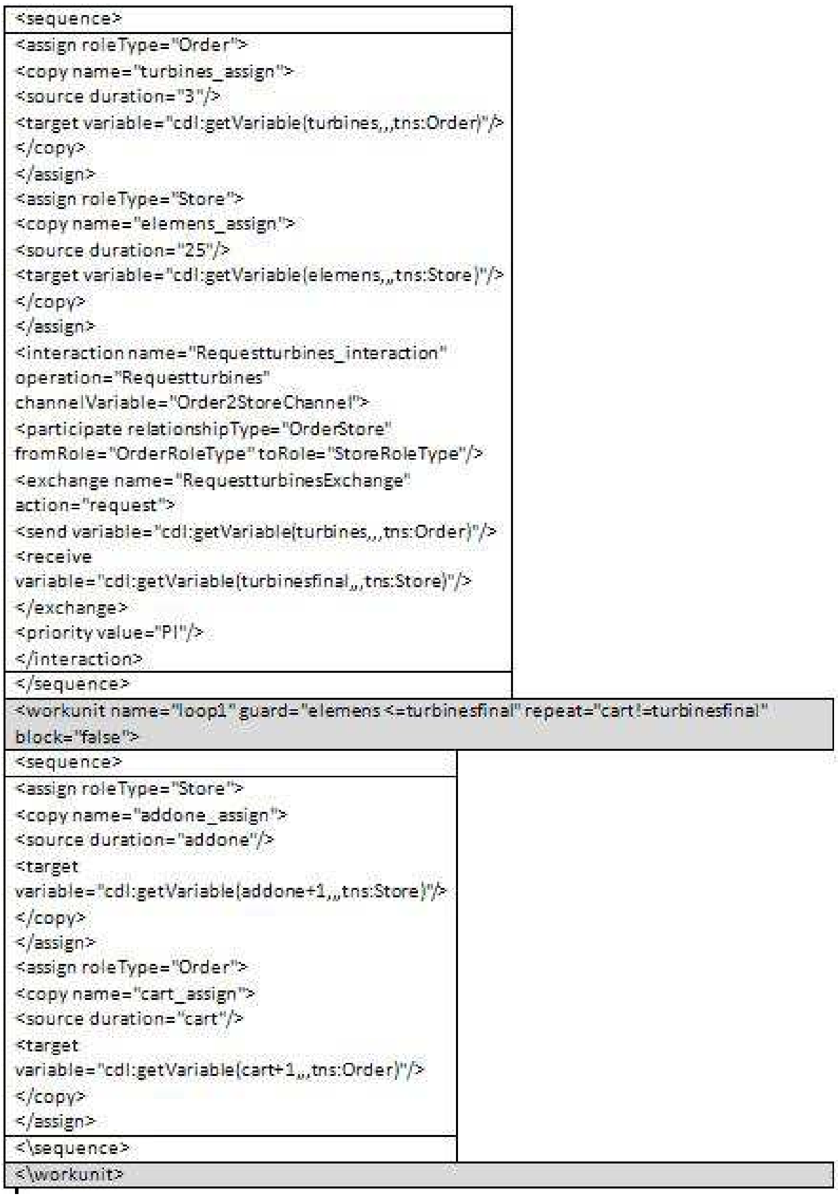
– There are no infinite occurrence sequences, which is a consequence of this system not having any iterative behavior.

– There are some *dead transitions*, some of which are *fail* transitions that cannot be executed because they correspond to failures that cannot occur. There are also some other *dead transitions*, corresponding to the travel agent's request confirmation and payment, which cannot be executed due to their lower priority.

– There is a non-dead *fail* transition, *failT*, which corresponds to the *time-out* of the traveler's request confirmation. This transition can then be fired when this time-out has elapsed.

### 6.1.2 An Aero-electric Management System

The system consists of three parts: wind turbine management system, productivity management system, and demand management system. The productivity management system receives reports from the turbine management system and the demand management system. It then analyzes these reports and decides how many turbines should be working. The demand management system controls the power requirements for the area drawing up a report, which it sends to the productivity management system. To illustrate the translation we focus our attention on one event in the system, namely, sharp increase in power demand. The demand management system periodically checks the need to increase the production of electric power. In all, the increase of demand must be attended to within 3 minutes. The demand management system calculates how much power is necessary, and sends a message to the productivity management system to increase the production. Then the productivity system analyzes the need, sending a message to the turbine system to request how many turbines are available to turn on. The turbine system get the number of idle turbines. Then it sends a reply to the productivity system with the number of available turbines and when the answer is received the system decides if it is possible to satisfy the demand. If there are enough turbines it sends a message to the turbine system for them to be turned on or else it sends a message to the demand system to indicate that it is not possible to satisfy the new demand.



```
<sequence>
<assign roleType="Order">
<copy name="turbines_assign">
<source duration="3"/>
<target variable="cdl:getVariable(turbines,,,tns:Order)"/>
</copy>
</assign>
<assign roleType="Store">
<copy name="elemens_assign">
<source duration="25"/>
<target variable="cdl:getVariable(elemens,,,tns:Store)"/>
</copy>
</assign>
<interaction name="Requestturbines_interaction"
operation="Requestturbines"
channelVariable="Order2StoreChannel">
<participate relationshipType="OrderStore"
fromRole="OrderRoleType" toRole="StoreRoleType"/>
<exchange name="RequestturbinesExchange"
action="request">
<send variable="cdl:getVariable(turbines,,,tns:Order)"/>
<receive
variable="cdl:getVariable(turbinesfinal,,,tns:Store)"/>
</exchange>
<priority value="PI"/>
</interaction>
</sequence>
<workunit name="loop1" guard="elemens <=turbinesfinal" repeat="cart!=turbinesfinal"
block="false">
<sequence>
<assign roleType="Store">
<copy name="addone_assign">
<source duration="addone"/>
<target
variable="cdl:getVariable(addone+1,,,tns:Store)"/>
</copy>
</assign>
<assign roleType="Order">
<copy name="cart_assign">
<source duration="cart"/>
<target
variable="cdl:getVariable(cart+1,,,tns:Order)"/>
</copy>
</assign>
<\sequence>
<\workunit>
```

Figure 36: WS-CDL description of the AMS.

Figure 36 contains the relevant parts of a WS-CDL document describing the system, the first part corresponds to a sequence structure and the second one is a workunit structure containing a sequence structure as well.
Using the WST Tool, we have obtained the corresponding PTCPN from this WS-CDL document (see Figures 37, 38, 39)

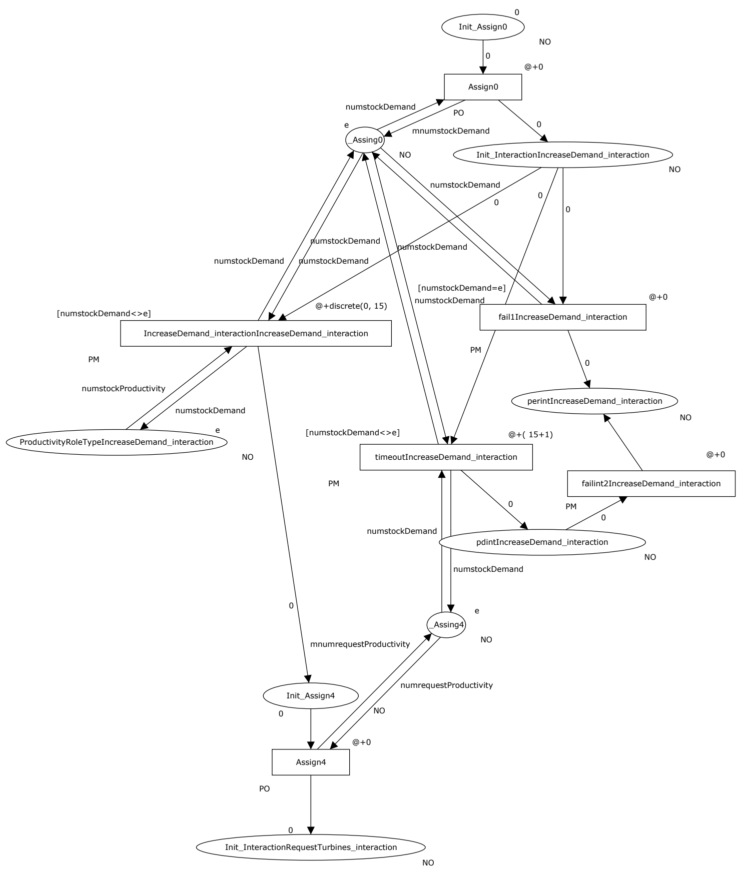


Figure 37: PTCPN corresponding to the aero-electric management system (Part I)

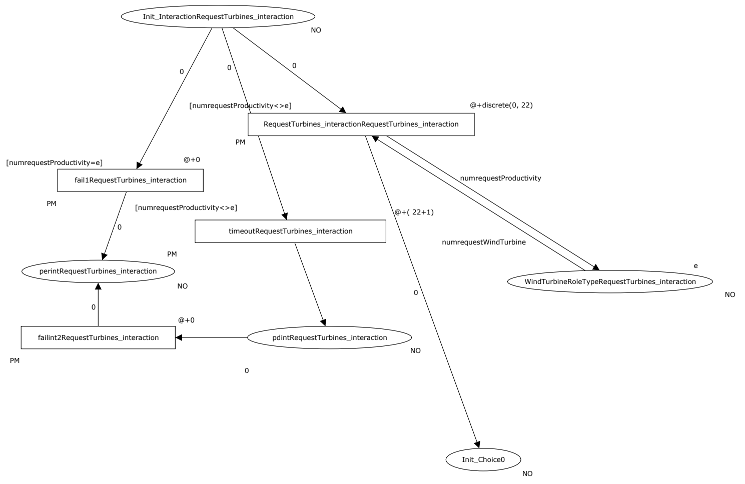

Init_InteractionRequestTurbines_interaction
NO
0
0
0
[numrequestProductivity<>e]
@+discrete(0, 22)
RequestTurbines_interactionRequestTurbines_interaction
PM
[numrequestProductivity=e]
@+0
fail1RequestTurbines_interaction
numrequestProductivity
PM
[numrequestProductivity<>e]
@+( 22+1)
0
timeoutRequestTurbines_interaction
numrequestWindTurbine
PM
perintRequestTurbines_interaction
e
NO
WindTurbineRoleTypeRequestTurbines_interaction
0
0
NO
@+0
failint2RequestTurbines_interaction
pdintRequestTurbines_interaction
PM
NO
0
Init_Choice0
NO

Figure 38: PTCPN corresponding to the aero-electric management system (Part II)

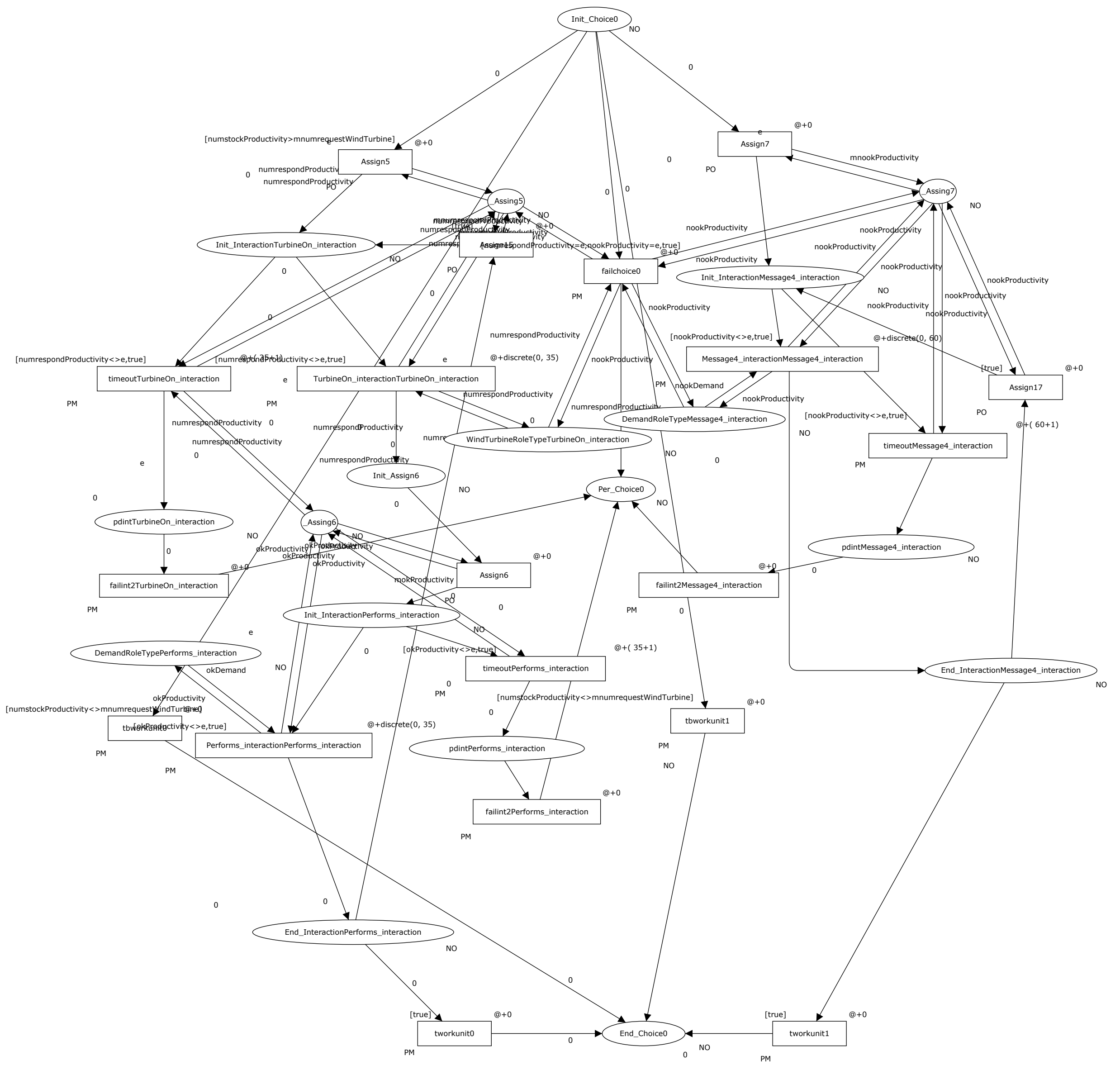


Figure 39: PTCPN corresponding to the aero-electric management system (Part III)

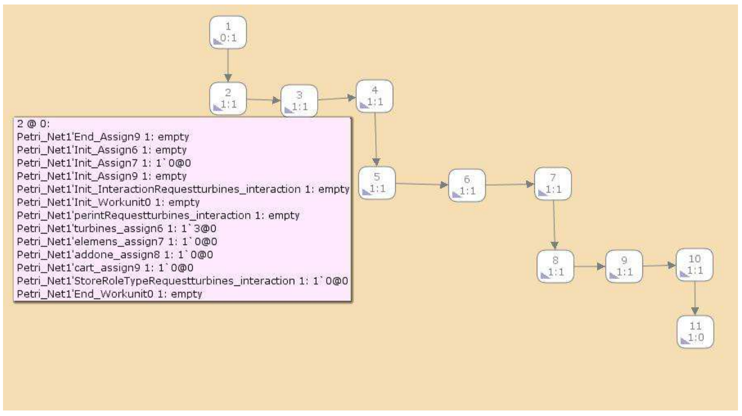
## 6.2 Verification and Validation

The obtained PTCPN can be verified and validated using CPN Tools.

Observing the state graph obtained from the tool, we can deduce the following properties:

- As expected, the initial marking is not a *home state*, because we have no way to return to it.

- From the *dead markings* that we have obtained, we conclude that the system execution always terminates concurretly, marking Pok. in a final marking *Pok*. This can be interpreted in the following way, when an order is delivered, then the system is always completes successfully.



```
2 @ 0:
Petri_Net1'End_Assign9 1: empty
Petri_Net1'Init_Assign6 1: empty
Petri_Net1'Init_Assign7 1: 1`0@0
Petri_Net1'Init_Assign9 1: empty
Petri_Net1'Init_InteractionRequestturbines_interaction 1: empty
Petri_Net1'Init_Workunit0 1: empty
Petri_Net1'perintRequestturbines_interaction 1: empty
Petri_Net1'turbines_assign6 1: 1`3@0
Petri_Net1'elemens_assign7 1: 1`0@0
Petri_Net1'addone_assign8 1: 1`0@0
Petri_Net1'cart_assign9 1: 1`0@0
Petri_Net1'StoreRoleTypeRequestturbines_interaction 1: 1`0@0
Petri_Net1'End_Workunit0 1: empty
```

Figure 40: Diagram

– There are no infinite occurrence sequences, which is a consequence of this system not having any iterative behavior.

– There are two *dead transition*, which come respectively, from the variable *munstock-Demand*, and the other one, from the time out corresponding to the interaction for demanding the number of turbines available. Actually, these transition cannot be fired because these condition will never occur (see Figure 40)

## 7 Conclusions and future work

We have extended the WST tool to support the translation from WS-CDL into PTCPNs, so that we can use CPN Tools in order to have an immediate execution verification and validation of the modelled system.

We have them presented a PTCPN semantics for a relevant subset of WS-CDL, in which integer data variables, timed aspects and priorities have been considered. The introduction of priorities allows the parties of a Web Composition to favor some interactions, which can be useful in many situations, for instance, to distinguish clients or items, as we have seen in the first case study. Timed restrictions have also been considered in the translation, both in interactions(time-outs) and in workunits, to delay the execution. The obtained PTCPNs are 1-safe and clean, which means that only one token can be at any place in any reachable marking, and when one of the initial or final places is marked, no other place can also be marked, excepting the places associated to variables or the dead tokens that could remain on some places when the error place has become marked.

The main advantage of this translation is that the PTCPNs obtained are currently supported by CPN Tools, a well known tool that is widely used in the Petri Nets community that allows us to simulate, analyze and verify the described system.

As future work, we plan to extend the translation to support a wider WS-CDL subset, e.g. the inclusion of a hierarchy of choreographies and finalize blocks. Another aspect that can be improved is that of abnormal terminations. In the translation presented here we have only considered the interaction cases that use unassigned source variables and interactions with a time-out. However, there are other abnormal termination situations (mainly related to variables) that we can be consider to extend the present version of the translation.

## References

[1] Samuele Carpineti and Cosimo Laneve. *A Basic Contract Language for Web Services* ESOP, pp 197-213. 2006.

[2] Marco Carbone and Kohei Honda and Nobuko Yoshida. *Structured Communication-Centred Programming for Web Services* ESOP, pp 2-17. 2007.

[3] Cosimo Laneve and Luca Padovani. *Smooth Orchestrators.* FoSSaCS, pp 32-46. 2006.

[4] Giuseppe Castagna and Nils Gesbert and Luca Padovani. *A theory of contracts for web services.* POPL, pp 261-272. 2008.

[5] W.M.P. van der Aalst. *Interval Timed Coloured Petri Nets and their Analysis.* Lecture Notes in Computer Science, vol. 691, pp. 451-472. 1993.

[6] W.M.P. van der Aalst and M.A. Odijk. *Analysis of Railway Stations by Means of Interval Timed Coloured Petri Nets.* Real-Time Systems, vol. 9, pp. 241-263. 1995.

[7] M. Ajmone Marsan, G. Balbo, A. Bobbio, G. Chiola, G. Conte and A. Cumani. *On Petri Nets with Stochastic Timing.* Proc. of the International Workshop on Timed Petri Nets, IEEE Computer Society Press, pp. 80-87. 1985.

[8] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services.* Springer-Verlag, 2002.

[9] T. Andrews et. al. BPEL4WS – Business Process Execution Language for Web Services. Version 1.1. May 2003., *http://www.ibm.com/developerworks/library/specification/ws-bpel/.*

[10] F. Bause. *On the Analysis of Petri Nets with Static Priorities.* Acta Informatica, vol. 33, no.7, pp. 669-686, 1986.

[11] E. Best and M. Koutny. *Petri Net Semantics of Priority Systems.* Theoretical Computer Science, vol.96, pp. 175-215. 1992.

[12] T. Bolognesi, F. Lucidi and S. Trigila. *From Timed Petri Nets to Timed LOTOS.* Proceedings of the Tenth International IFIP WG6.1 Symposium on Protocol Specification, Testing and Verification. North-Holland, 1990.

[13] Fred D.J. Bowden. *Modelling time in Petri nets.* Proc. Second Australia-Japan Workshop on Stochastic Models. 1996.

[14] A. Brogi, C. Canal, E. Pimentel, and A. Vallecillo. Formalizing Web Service Choreography. *In WS-FM'04. Electronic Notes in Theoretical Computer Science, 2004.*

[15] G. Bucci, A. Fedeli, L. Sassoli and E. Vicario. *Modeling Flexible Real Time Systems with Preemptive Time Petri Nets.* Proc. 15th Euromicro Conference on Real-Time Systems (ECRTS'03), pp. 279-286, IEEE Computer Society Press, 2003.

[16] T. Bultan, X. Fu and J. Su. *Analyzing Conversations of Web Services.* In IEEE Internet Computing, vol.10, no.1, pp. 18–25. 2006.

[17] CPN Tools homepage. *www.cs.au.dk/CPNTools.*

[18] Rachid Hamadi and Boualem Benatallah. A Petri Net-based Model for Web Service Composition. In *ADC '03: Proceedings of the 14th Australasian database conference*, 2003.

[19] Hongli Yang, Xiangpeng Zhao, Zongyan Qiu, Geguang Pu, and Shuling Wang. A Formal Model for Web Service Choreography Description Language (WS-CDL). *International Conference on Web Services (ICWS'06), pp. 893-894, IEEE Computer Society Press. 2006.*

[20] Kurt Jensen. Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Monographs in Theoretical Computer Science, Springer-Verlag. 1997.

[21] K. Jensen and L. M. Kristensen. *Coloured Petri Nets. Modelling and Validation of Concurrent Systems.* Springer, 2009.

[22] A. Martens. Analyzing web service based business processes. In *Proc. of International Conference on Fundamental Approaches to Software Engineering (FASE'05)*, Lecture Notes in Computer Science, vol. 3442, pp. 19–33. 2005.

[23] P. Merlin. *A Study of the Recoverability of Communication Protocols.* PhD. Thesis, Univ. of California. 1974.

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

[25] Srini Narayanan and Sheila A. Mcllraith. *Simulation, Verification and Automated Composition of Web Services.* Proc. 11th International Conference on World Wide Web (WWW'02), pp. 77–88. 2002.

[26] J.L. Peterson. *Petri Net Theory and the Modeling of Systems.* Prentice-Hall, 1981.

[27] C. Ramchandani. *Performance Evaluation of Asynchronous Concurrent Systems by Timed Petri Nets.* PhD. Thesis, Massachusetts Institute of Technology, Cambridge. 1973.

[28] W. Reisig. *Petri Nets: An Introduction.* EATCS Monographs on Theoretical Computer Science, Springer-Verlag, vol. 4, 1985.

[29] G. Salaun, L. Bordeaux, and M. Schaerf. Describing and Reasoning on Web Services Using Process Algebra. *In Second International Conference on Web Services, IEEE Computer Society Press. 2004.*

[30] J. Sifakis. *Use of Petri Nets for Performance Evaluation.* Proc. of the Third International Symposium IFIP W.G.7.3., Measuring, Modelling and Evaluating Computer Systems. Elsevier Science Publishers, pp. 75-93. 1977.

[31] Zhangxi Tan and Chuang Lin and Hao Yin and Ye Hong and Guangxi Zhu. Approximate Performance Analysis of Web Services Flow Using Stochastic Petri Net. *Springer-Verlag, Lecture Notes in Computer Science,* vol. 3251, pp. 193-200, 2004.

[32] Johnson P. Thomas, Mathews Thomas, and George Ghinea. Modeling of Web Services Flow. In *IEEE International Conference on E-Commerce, Newport Beach, California, USA*, pages 391–398, June 2003.

[33] H.M.W. Verbeek and W.M.P. van der Aalst. Analyzing BPEL Processes using Petri Nets. In *Proceedings of the Second International Workshop on Applications of Petri Nets to Coordination, Workflow and Business Process Management*, pp. 59–78, 2005.

[34] V. Valero, H. Macià, J.J. Pardo, M.E. Cambronero and G. Díaz. *Transforming Web Services Choreographies with priorities and time constraints into prioritized-time colored Petri nets.* Science of Computer Programming, Elsevier. In press, doi: 10.1016/j.scico.2011.05.002, June 2011.

[35] V. Valero, D. de Frutos, and F. Cuartero. *Decidability of the Strict Reachability Problem for TPN's with Rational and Real Durations.* Proc. 5th. International Workshop on Petri Nets and Performance Models, pp. 56-65. 1993.

[36] J. Wang. *Timed Petri Nets, Theory and Application.* Kluwer. 1998.

[37] World Wide Web Consortium (W3C). *http://www.w3.org/.*

[38] Web Services Choreography Description Language Version 1.0 (WS-CDL). *http://www.w3.org/TR/ws-cdl-10/.*

[39] W3C. Web Services Description Language (WSDL). Version 1.1. *http://www.w3.org/TR/wsdl/*.

[40] Web Service Choreography Interface (WSCI) 1.0 *http://www.w3.org/TR/wsci/*.

[41] WST Tool *http://www.dsi.uclm.es/retics/WST/*.