Transforming Web Services Choreographies with priorities and time constraints into prioritized-time colored Petri nets

Valentin Valero *, Hermenegilda Macià, Juan José Pardo, María Emilia Cambronero, Gregorio Díaz

Escuela Superior de Ingeniería Informática, Universidad de Castilla-La Mancha, Campus Universitario s/n, 02071 Albacete, Spain

ARTICLE INFO
Article history:
Received 27 September 2007
Received in revised form 22 March 2011
Accepted 30 May 2011
Available online 21 June 2011

Keywords:
Web Services
Web Service composition
Priorities
Choreography
Timed colored Petri nets

ABSTRACT
A Web Service is a self-describing, self-contained modular application that can be published, located, and invoked over a network, such as the Internet. Web Service composition provides a way to obtain value-added services by combining different existing facilities, which are then able to support the integration of commercial applications. WS-CDL (Web Services Choreography Description Language) is a W3C candidate recommendation for the description of peer-to-peer collaborations by participants in a Web Services composition. This paper focuses on several important aspects of WS-CDL, namely, data variables, timed restrictions, as well as the prioritization of collaborations. In WS-CDL there are no priorities, thus, one of our first goals is to provide a WS-CDL definition of prioritized collaborations. We also define a semantics of WS-CDL (with priorities) by means of prioritized-time colored Petri nets.

© 2011 Elsevier B.V. All rights reserved.

1. Introduction

A Web Service can be defined [5] 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, and are becoming increasingly important as a platform for B2B integration. Web Service 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. Web Services can be written in different languages and can be executed on different platforms.

Internet and Web technologies are a new way of doing business cheaply and efficiently, as business firms can provide new and dynamic services in a faster way by the composition of Web Services. However, B2B e-commerce is still emerging, and new software technologies are required to support their development. There is a special need for an effective means to abstract, compose, analyze and evolve Web Services in an appropriate time-frame [19].

Current technology is based on the Web Service architecture stack, proposed by the World Wide Web Consortium, W3C [39], which consists of the following elements:

• SOAP: This defines the basic formatting of a message and the basic delivery options regardless of programming language, operating system, or platform.
- **WSDL**: Describes the static interface of a Web Service. At this point the format of the messages sent and received by Web Services are defined.
- **Registry (UDDI)**: Makes an available Web Service visible and describes its specific capabilities.
- **Security layer**: Ensures that exchanged information is not modified or forged in a verifiable manner and that parties can be authenticated.
- **Reliable Messaging layer**: Provides a reliable layer for the exchange of information between parties.
- **Context, Coordination and Transaction layer**: Defines interoperable mechanisms for propagating the context of long-lived business transactions and enables parties to meet correctness requirements by following a global agreement protocol.
- **Business Process Languages layer (WSBPEL)**: Describes the execution logic of Web Services-based applications by defining their control flows (e.g., conditional, sequential, parallel and exceptional execution) and prescribing the rules for consistently managing their non-observable data.
- **Choreography layer**: Describes collaborations between parties by globally defining their common and complementary observable behavior, where information exchanges occur, and when the jointly agreed ordering rules are satisfied.

It should be noted that Web Service Choreography and Orchestration specifications are aimed at the composition of interoperable collaborations between all types of parties, regardless of the supporting platform or programming model used in the implementation of the hosting environment.

In this paper we focus on the Choreography layer, with special attention to the description of timed and prioritized interactions. WS-CDL [40] is a W3C 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 between the parties involved by means of choreographies and activities. However, the WS-CDL standard does not allow specifying the priorities associated with these activities, so we also propose an extension of the standard in order to obtain this capability. This is done by associating priorities with interactions, taking into account that in a composite Web service certain interactions may need preference over others.

We use a prioritized-timed model of colored Petri nets [22] to capture the main WS-CDL elements, thus providing a formal framework to describe the behavior of the parties involved in a choreography. This specific model of prioritized-timed Petri net is supported by CPN Tools [18], which make it possible to carry out simulations and analyze properties.

The objectives are therefore twofold: 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 service and the interactions among the different participants. On the other hand, Petri nets are also a formal tool, in the sense that they provide not only a static vision of a system, but also its dynamic behavior. We can thus use the Petri net representation to validate and verify the composed Web service.

The paper is structured as follows: Section 2 presents a brief review of a selection of works related to this topic; Section 3 gives a brief description of the main elements of WS-CDL; Section 4 explains how to introduce priorities in WS-CDL. The particular model of prioritized-timed colored Petri net that we use is introduced in Section 5, and the translation is shown in Section 6, together with the appropriate translation support tool (WST). Some properties of the PTPNs obtained are proved in Section 7, and a case study that illustrates the translation is shown in Section 8. Finally, in Section 9 the conclusions are formulated, and some indications are given about future work.

2. Related works

Formal models have been widely used to describe Web service compositions. For instance, in [16] a model based on state machines is used to represent and analyze conversations from Web Services. Yang Hongli et al. [43] also made a translation of WS-CDL into a formal model, in this case a small language (CDL), for which they provide an operational semantics. This work was later extended [29], by including a projection of the choreography level in the orchestration level. The dominant role concept is introduced, which is used in the implementation of any choice or interaction structure of the choreography. More recently, Tasharofi and Sirjani [34] defined a translation from WS-CDL to Reo [7] and Constraint Automata with State Memory (CASM), which they exploit for conformance validation. However, Petri nets are not used in these works, nor are time or priorities considered. In [10] we can find an approach that associates priorities with interactions in distributed systems. In this work the focus is mainly on distributed memoryless controllers, but the technique presented, based on components modeled by LTS (Labeled Transition Systems) can also be applied to service-oriented systems.

Compositional models of Petri nets have been widely used to define the semantics of many concurrent languages. Best et al. defined a compositional semantics of a simple concurrent programming language, B(PN), using low-level Petri nets [11] and high-level Petri nets [12], but time or priorities were not considered. In [12] M-nets are used, which have some similarities with colored Petri nets, since places, arcs and transitions carry some associated information, but the high-level structure of this model is less general than that of colored Petri nets. However, in M-nets places are also labeled with information on their type (internal, entry or exit) for compositional purposes, which is a feature that we also use in this work, but we consider another sort of label, the “error” places, which will be marked in the event of a failure.

Regarding Petri net representations of Web service compositions, we have the work of Thomas et al. [35], who defined a timed Petri net representation of Web Service Flows; in this case, only the flow of messages and methods are considered,
Choreographies, Information Types, Variables and Tokens. Participant Types, Role Types and Relationship Types

on the basis of appropriate infrastructuresupport. The globalspecification is in turn performed by a combination of the resulting common and complementary observable behavior of all the parties involved. Each of which can then use the global definition conditions and constraints under which messages are exchanged. The contract describes, from a global viewpoint, the interactions among the parties involved.

Some works use algebraic models: Salaun et al. [32] defined a process algebra to derive the interactive behavior of a business process starting from a BPEL4WS specification. Brogi et al. [15] defined a translation of WSCI (Web Service Choreography Interface) [42] to CCS [27], and described its advantages. Yeung [44] defined a mapping from WS-CDL and BPEL4WS into CSP [20], providing a formal approach to verifying the behavior of collaborating Web services. Another mapping from WS-CDL into a process algebra (FSP) is defined in [31], where a simplified model of WS-CDL known as Chor [29] is encoded into FSP.

Finally, some works are based on $\pi$-calculus, defining a calculus to formalize the interaction semantics of Web Services. Laneve and Padovani [23], for example, define a Web Service orchestration model, and use an extension of $\pi$-calculus to join patterns. They propose a constraint on the input join pattern where communication channels are co-located. Carbone et al. [17] present two different formalisms to describe communication behaviors. The first focuses on global message flows and the second on end-point behaviors, but both use a parallel extension of session types.

3. WS-CDL

The Web Services Choreography specification offers a precise description of the collaborations between the parties involved in a choreography. WS-CDL specifications are contracts containing “global” definitions of the common ordering conditions and constraints under which messages are exchanged. The contract describes, from a global viewpoint, the common and complementary observable behavior of all the parties involved. Each of which can then use the global definition to build and test solutions that conform to it. The global specification is in turn performed by a combination of the resulting local systems, on the basis of appropriate infrastructure support.

The WS-CDL model consists of the following entities [40]:

- **Participant Types, Role Types and Relationship Types.** A role type enumerates the observable behavior a party exhibits in order to collaborate with other parties, whereas a participant type identifies a set of role types that must be implemented by the same logical entity or organization. Its purpose is to group together those parts of the observable behavior that must be implemented by the same logical entity or organization. We will only use role types, assuming that each participant is associated with a single role type. Relationship types are used to identify the mutual commitments that must be made between two parties for them to collaborate successfully.

- **Information Types, Variables and Tokens.** Information types describe the type of information used in a choreography, for instance, integer values. In WS-CDL the variables contain information about commonly observable objects in a collaboration, such as the information exchanged or the observable information of the roles involved. Tokens are aliases that can be used to refer to parts of a (structured) variable.

- **Choreographies.** As previously stated, these establish the common rules that govern the ordering of exchanged messages and collaborative behavior among the involved participants. A WS-CDL document, in general, consists of a hierarchy of choreographies, which are executed by using the WS-CDL perform activity. For the sake of simplicity, we only consider the case of a WS-CDL document with a single choreography (root choreography), but the extension to a hierarchy of choreographies can be easily made by using a hierarchy of modules in CPN tools.

A choreography in WS-CDL consists of three parts:

- **Choreography Life-line:** This describes the body (main activity) of a collaboration. Initially, the collaboration is established between the parties, after which some work is performed within it, and finally it completes either normally or abnormally.

- **Choreography Exception Block:** This specifies the additional interactions that should occur when a Choreography behaves in an abnormal way.

- **Choreography Finalizer Block:** This describes how to specify additional interactions that should occur to modify the effect of an earlier successfully completed choreography (for example to confirm or undo the effect).
Choreographies do not have finalizer blocks, since no other choreography can have been performed previously. Thus, since we only consider a single (root) choreography, we do not need to consider finalizer blocks.

- **Channels** establish a point of collaboration between parties by specifying where and how information is exchanged.

- **Activities and Ordering Structures.** The collaborative behavior of the participants in a choreography is described by means of activities. These are the actions performed within a choreography, and are divided into three groups: basic activities, ordering structures and workunits. Basic activities are used to establish the variable values (assign), to indicate an inner action of a specific participant (silent_action), or that a participant does not perform any action (noaction), and also to establish message exchanging between two participants (interaction).

  In general, WS-CDL interactions may contain several message exchanges (in both directions). However, for the sake of simplicity, we will only consider a simple sender-receiver message exchange in each interaction. Thus, within an interaction, the value of a source variable (from the sender) is transferred to a target variable (the receiver), but when the source variable has not been initialized an error occurs and the choreography exception block is executed.

  Interactions can also be assigned a time-out, i.e. a completion time. When the time-out expires, if the interaction has not been completed, the interaction finishes abnormally, and the choreography exception block is performed.

The ordering structures are used to combine activities within a nested structure to express the ordering conditions under which information is exchanged. The ordering structures are the sequence, choice and parallel, with the usual interpretations. Finally, workunits allow the execution of some activities when a certain condition holds, so that a workunit encapsulates an activity, which can only be executed if the corresponding guard is evaluated to be true. The workunits also contain a further guard to allow the iteration of the enclosed activity.

According to the previous description, time information in WS-CDL can appear in both interactions (time-outs) and date/time variables (using XPath).

Time-outs in interactions are specified with the following syntax:

```xml
<timeout time-to-complete="XPath-expression"/>
```

The time-to-complete attribute specifies the time frame in which an interaction must be completed. When this time expires (after initiation), if the interaction has not been completed, the timeout occurs and the interaction finishes abnormally, causing an exception block to be executed in the choreography.

XPath 2.0 supports date and time variables, so they can be used in WS-CDL, by a number of different functions. These variables can be used in particular to delay the execution for a certain time, or to establish the times at which some actions must be executed. Workunit guards can be used for this purpose, by including an expression related to the value of a date/time variable. In fact, as we intend to capture delays or execution times, the specific expressions allowed are those constructed using the XPath 2.0 functions `getCurrentDateTime` (returns the current date and time at a given role), `hasDurationPassed` (returns true when the specified time has elapsed at a given role), and the XPath 2.0 comparison operators, which can be used with date/time variables: `eq` (equal to), `gt` (greater than), `ge` (greater than or equal), `lt` (less than) and `le` (less than or equal). WS-CDL specification actually suggests the use of these operators for timing purposes.

### 4. WS-CDL with priorities

In many cases certain interactions need to be given preference over others, i.e. in the composition of Web Services some parties can express their interest in the prioritization of certain interactions, for example, for selling or reserving items of some different kinds. Clients interact with the Web server to buy or reserve items by interactions that may have different levels of priority, depending on the item or even on the client involved.

WS-CDL has a choice construct, which allows us to choose among some different activities. However, the textual description in [40] is a little vague, as it states “when two or more activities are specified in a choice element, only one activity is selected and the other activities are disabled.” But “if the choice has workunits with guard conditions, the first workunit that matches the guard condition is selected and the other workunits are disabled, and when there is more than one match, lexical ordering is used to select a match.” It also says “if the choice has other activities, it is assumed that the selection criteria for those activities are non-observable.” It is not clear from this description, then, what should be done when both guarded activities and non-guarded activities appear as alternatives in a choice.

As a matter of fact, this textual description introduces a kind of prioritization by means of lexical ordering in the case of guarded workunits. However, we consider that lexical ordering is not the best way to prioritize interactions, as it is not a flexible technique (a complete piece of code must be moved in the event of priority change), and it does not allow different interactions with the same priority to be considered. In [43] the authors have solved the problem by distinguishing two types of choice, non-deterministic and general choice (guarded workunits). We have decided to equip interactions with priorities, and then, one of the interactions with the maximum priority (lowest numeric value) is selected for execution. When there are several interactions with the same (lowest) priority value, the selection criteria are non-observable.

Accordingly, in this section we propose an extension of WS-CDL with priorities. Priorities are established as natural numbers, with the same interpretation as in CPN Tools, the greater the number, the lower the priority for the corresponding

activity in the system. They are associated with interactions, so we extend the syntax of the WS-CDL interaction activities with an attribute \textit{priority} (see Fig. 1 for our proposed syntax).

The interpretation of this attribute is the following: in the event of conflict only those interactions with the maximum priority (lowest value) are allowed.

5. Prioritized-timed colored Petri nets

In this section we introduce the specific model of prioritized-timed colored Petri net that we consider for the translation. In the literature on timed extensions of Petri nets we can identify a first group of models, which assign time delays to transitions, by using either a fixed deterministic value \cite{30,33,37} or choosing it from a probability distribution \cite{4}. Other models use time intervals to establish the enabling times of transitions \cite{26}. There are also models that introduce time on tokens \cite{1,2,13}. In \cite{14,38} a description is given of the different approaches to introduce time in Petri nets. Priorities were also introduced in Petri nets to extend the descriptive power of the model \cite{8,9,28}, usually by associating priority levels with transitions and modifying the firing rule to prevent the firing of a transition when another one with higher priority is enabled.

We use prioritized-timed colored Petri nets, which are a prioritized-timed extension of colored Petri nets \cite{21}, the well-known model supported by CPN Tools \cite{18}, developed by the CPN group at the University of Aarhus. In this model, places have an associated color set (data types). Each token then has an attached data value (\textit{token color}), which belongs to the color to which the token is associated.

We will use timed colors, for which the first component will be a non-negative integer value, representing the data value, and the second component will be the token timestamp, a natural number representing the time at which the token will be available.

There is also a discrete global clock that represents the total time elapsed in the system model. Arcs also have an associated inscription (\textit{arc expressions}), constructed using variables, constants, operators and functions. To evaluate an arc expression we need to bind the variables, which consists of 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 associated time information both for place-transition and transition-place arcs. However, only time inscriptions are needed 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 with the transition instead of the output arcs. This is the specific model that we use in our WS-CDL semantics, i.e. we will only consider these time inscriptions in the transitions. We will therefore not use any time inscription in the arcs.

The time inscription associated with a transition 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. Transitions can also have associated guards, which are Boolean expressions that can 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.

\textbf{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 $C : X \rightarrow \mathbb{N}$, 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 $x \neq x_i$, $i = 1, \ldots, n$. 

![Fig. 1. WS-CDL interactions extended with priorities.](image-url)
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:
  \[ C_1 + C_2 \in \mathcal{B}(X), \text{ where } \forall x \in X : (C_1 + C_2)(x) = C_1(x) + C_2(x). \]
  \[ C_1 \subseteq C_2 \text{ if and only if } \forall x \in X : C_1(x) \leq C_2(x). \]
  \[ \text{If } C_2 \subseteq C_1 \text{ we can define the subtraction } C_1 - C_2 \in \mathcal{B}(X), \text{ where } \forall x \in X : (C_1 - C_2)(x) = C_1(x) - C_2(x). \]

- For any \( C \in \mathcal{B}(\Sigma) \), we define the first projection \( \Pi_1(C) \in \mathcal{B}(N) \), as follows: \( \forall n \in N, \; \Pi_1(C)(n) = \sum_{m \in N} C(n, m) \).

- For any \( C \in \mathcal{B}(\Sigma) \) and \( n \in N \) 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 \( (n, m) \in C, \forall i = 1, \ldots, \Pi_1(C)(n) \) and \( m_i \leq m_{i+1} \).

- For any \( C_1, C_2 \in \mathcal{B}(\Sigma) \), we say that \( C_1 \leq C_2 \) if and only if the following conditions hold:
  \[ \forall i \in 1, \ldots, \Pi_1(C_1)(n), \text{ taking } \Pi_2(C_1, n) = (m_1^1, \ldots, m_{\Pi_1(C_1)(n)}^1) \text{ and } \Pi_2(C_2, n) = (m_1^2, \ldots, m_{\Pi_1(C_2)(n)}^2), \text{ we must have } m_i^1 \geq m_i^2, \]
  \[ \forall i = 1, \ldots, \Pi_1(C_1)(n). \]

These conditions state that for every \( n \) the total number of elements \( (n, m) \) (moving \( m \)) must be less 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 \( C_1 \leq C_2 \), we define \( C_2 \ominus C_1 \) in the following (recursive) way:
  \[ C_2 \ominus C_1 = \emptyset \text{ if we take } C_2 \ominus C_1 = C_2, \]
  \[ \text{For } C_1 \neq \emptyset, \text{ let us consider that } \]
  \[ C_2 = \{ r_1^1.(n_1, m_1^1), \ldots, r_i^1.(n_i, m_i^1), \ldots, r_k^1.(n_k, m_k^1) \}, \text{ where } n_i \neq n_j, \forall i \neq j, \text{ and } m_j < m_{j+1}. \]
  \[ \forall i = 1, \ldots, k \text{ and } \forall j \in 1, \ldots, i. \]
  Since \( C_1 \leq C_2 \), we can take one element \( (n_i, m) \) from \( C_1 \), for some \( i \in \{ 1, \ldots, k \} \), as well as the largest index \( j \) for which \( m_j \leq m \). We then define recursively:
  \[ C_2 \ominus C_1 = \{ (r_1^1.(n_1, m_1^1), \ldots, r_i^1.(n_i, m_i^1), \ldots, r_{i-1}^1.(n_{i-1}, m_{i-1}^1), \ldots, r_{i-1}^1.(n_1, m_1^1)), \ldots, r_k^1.(n_k, m_k^1) \} \ominus \{ (1 \cdot (n_i, m)) \}. \]
  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 \( C_1 \leq C_2 \). 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 \( (P, T, A, V, C, G, E, \lambda, D, \pi) \), 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. \( \text{Type}(v) \in \Sigma \), for all \( v \in V \).
- \( C: T \rightarrow \text{EXPRv} \) is the guard function, which assigns a Boolean expression to each transition, i.e. \( \text{Type}(G(t)) = \text{Bool} \).
- \( E: A \rightarrow \text{EXPRv} \) is the arc expression function, which assigns an expression to each arc, such that \( \text{Type}(E(a)) = \mathcal{B}(\mathbb{N} \times \{ 0 \}) \), which corresponds to untimed arcs, since, as mentioned above, we only attach time delays to transitions.
- \( \lambda \) is the labeling function, defined both on places and transitions.

Places are labeled as \( \text{entry places, exit places, error places, internal places and variable places} \), which, respectively, correspond to the following labels: \( \{ \text{in, ok, er, i, rv} \} \). In our specific model, a PTCPN will have an only \( \text{entry place} \) \( p_{\text{in}} \), such that \( p^{\text{in}}_{\text{in}} = \emptyset \), which will be initially marked with a single token of color \( (0, 0) \). There is also an only \( \text{exit place} \) \( p_{\text{out}} \), such that \( p^{\text{out}}_{\text{out}} = \emptyset \), which will be marked with one token when the system finishes correctly. Each PTCPN has also a single \( \text{error place} \) \( p_{\text{error}} \), such that \( p^{\text{error}}_{\text{error}} = \emptyset \), which will become marked with one token in the event of a failure. Variable places are denoted by \( p_v \), 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: \( \lambda(t) \in L \cup \{ \text{fail} \} \cup \{ \text{time\_out, silent, noaction(r), assign(r, v, n), inter(r_1, r_2, v_1, v_2) \} \} \).

---

\(^1\) We use the classical notation on Petri nets to denote the precondition and postcondition of both places and transitions:

\[ \forall x \in P \cup T : *x = \{ y \mid (y, x) \in A \} \quad x^* = \{ y \mid (x, y) \in A \}. \]
Let us consider the marked PTCPN depicted in $\pi_D$ on $(4)$ of color value must be 1, and the timestamp depends again on the chosen delay in the time interval.

- $\pi : T \rightarrow \mathbb{N}$ 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 $M : P \rightarrow B(\Sigma)$, 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 [22], 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 $b$ 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 $e(b)$ the evaluation of $e$ for the binding $b$.

A binding element $e$ 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:

1. $x' \geq x$.
2. $G(t)(b) = true$.
3. For all $p \in \mathbf{t}^*$, $E(p, t)(b)_{x'} \leq M(p)$, where $E(p, t)(b)_{x'}$ consists of the same colors as $E(p, t)(b)$, 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:

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

**Example 1.** Let us consider the marked PTCPN depicted in Fig. 2, obtained from CPN Tools. Observe that timed color tokens in CPN Tools are drawn using the notation $n'@v$, 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 used there to represent the union of timed multisets.

Thus, $p_m$ 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 $p_m$ 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 $p_m$, 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 $p_m$ 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 $t_2$, 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 $t_2$ 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 $(1, 4)$.

Two tokens now remain in $p_i$, with colors $(4, 3)$ and $(6, 3)$, and $t_2$ becomes the only transition enabled (due to condition (4) of Definition 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 $p_{ok}$, 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.
6. 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, which means that for every reachable marking we will have at most one token on every place. 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 (Fig. 1). 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.

As mentioned above 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:

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

\(^2\) This does not mean that this is the only initially marked place.

\(^3\) We omit the specific syntax of each element, which can be found in the WS-CDL description document [40], 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 Fig. 3, where we are joining the following places:

$$
\begin{align*}
  p_{c_{in}} &= p_{a_{in}} \\
  p_{c_{er}} &= p_{a_{ok}} = p_{e_{er}} \\
  p_{c_{ok}} &= p_{a_{ok}} = p_{e_{ok}} \\
  p_{a_{er}} &= p_{e_{in}}
\end{align*}
$$

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, $p_{c_{ok}}, p_{a_{er}}, p_{e_{er}}$ 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 $1$.

$$(e, 0), e \in \mathbb{N}$$

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.

### 6.1. Basic activities

As we are considering only a choreography (root), we do not need to consider either the basic activities `perform` or `finalize` (see [40] 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 denominated it $P_0$, 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**.

  As mentioned in Section 3, 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 ($P_0$).
**Fig. 4.** Basic activities translation.

**Fig. 5.** Translation of an interaction without 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 can be fired at any moment, so its associated time interval is `[0, MaxInt]`, where `MaxInt` represents the integer maximum 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 have been 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`).

### 6.2. Workunits

**Fig. 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`,\(^4\) The value `e` means that the variable remains unassigned.
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 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 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. Fig. 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 Fig. 9, in which \( A_1 \) is the activity inside the workunit and \( N_{A_1} \) its corresponding PTPCN. There is a new transition \( tt \) connecting \( p_{in} \) with \( p_{inA_1} \), 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 \( tI \in p_{inA_1}^* \) we consider a new transition, \( tIr \), with the same interval, label and priority as \( tI \), and its guard is obtained as a conjunction of the guard of \( tI \) (\( g(tI) \) in the Figure) and the repetition condition of the workunit (\( g' \)). Then, \( tIr \) is connected as follows: \( tIr = \{p_{okA_1} \cup (\{tI \setminus \{p_{inA_1}\}\} \cup \{p_{out} | v_i \text{ appears in } g'\}) \} \cup \{p_{out} | v_i \text{ appears in } g'\} \). For any variable \( v_i \) appearing in \( g' \), if there is already a self-loop arc connecting \( tIr \) with \( p_{in} \), we keep the existing label in both arcs. Otherwise, both arc expressions are \( v_i \). The purpose of these transitions \( tIr \) is therefore to perform \( A_1 \) again when it has been correctly terminated and \( g' \) is evaluated to be 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:
\[
\begin{align*}
&\text{\textbf{Block = true} (Fig. 10).} \\
&\text{In this case every initial transition } t_1 \in p_{N_{A_1}}^* \text{ of } N_{A_1} \text{ (the PTCN corresponding to the activity inside the workunit) is replaced by two new transitions, } t_1' \text{ and } t_1'', \text{ connected as follows:} \\
&\quad t_1' = t_1 \cup \{p_{\eta t_1^i} | v_i \text{ appears in } g\} \\
&\quad t_1'' = \{p_{\eta t_1^i} \cup (t_1 \setminus \{p_{\eta t_1^i}\}) \cup \{p_{\eta t_1^i} | v_i \text{ appears in } g'\} \\
&\quad t_1'' = t_1'' \cup \{p_{\eta t_1^i} | v_i \text{ appears in } g'\}.
\end{align*}
\]

For any variable \( v_i \) appearing in \( g \) (resp. \( g' \)), if there was already a self-loop arc connecting \( t l' \) (resp. \( t_i'' \)) with \( p_{\eta t_1^i} \), 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 \( p_{ok} \).
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.

6.3. 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 (Fig. 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.
• Parallel: The translation for a parallel activity is depicted in Fig. 13, which includes two new transitions $t_1$ and $t_2$. 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 $t_1$, another transition of another parallel activity (with lower priority) could be executed first if $t_1$ is not executed immediately (as an action with the maximum priority in the model).

Transition $t_1$ 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, $p_e$, 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 $p_e$, 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 ($p_e$ 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 Fig. 14 this transition would be fired immediately, due to its maximum priority, i.e. we would not actually have a choice.

We can see in Fig. 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 ($t \in p_{\text{init}}^{\text{fail}}$, $\lambda(t) = \text{fail}$), 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.

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.** Fig. 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 Fig. 16. □

6.4. 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.

6.5. Implementation

This PTCPN semantics of the considered subset of WS-CDL has been implemented as an additional feature of a tool we are developing for the analysis and design of composite Web services. It is called the WST (Web Services Translation Tool), and can be obtained from [http://www.dsi.uclm.es/retics/wst](http://www.dsi.uclm.es/retics/wst).
The WST is an integrated environment that supports the specification of composite Web services by different formalisms. We can, for instance, describe a composite Web service by using UML 2.0 sequence diagrams extended with frames, and then automatically obtain a corresponding WS-CDL document, which, in turn, can be translated into a timed automata or PTCPN representation. The PTCPN representation can be directly used by CPN Tools, so that we immediately have the ability to simulate and analyze the system under study.

7. Properties

In this section we prove some properties related to the PTCPNs that can be obtained by the translation. We first show that the initial place $p_{in}$ can only be marked initially, i.e. excepting the initial marking there is no reachable timed marking at which $p_{in}$ is marked again.

We will also prove that the obtained PTCPNs are 1-safe, and that only one of the two exit places can finally be marked.

We will then show that these PTCPNs are clean, which means that for every reachable timed marking when $p_{in}$ or $p_{ok}$ are marked with one token, no other place can be marked, excepting those associated with variables ($p_{ri}$, $p_{rv}$), which are always marked with one token, whereas if $p_{er}$ becomes marked with one token, we may have some dead tokens remaining on some places.

**Proposition 1.** Let $N = (P, T, A, V, G, E, \lambda, D, \pi)$ be a PTCPN obtained from a WS-CDL document by applying the translation. As initial marking $M_0$ of $N$ we take one token at the initial place $p_{in}$, and one token with value $e$ on each $p_{ri\nu}$ place. Then, for any reachable marking $M \neq M_0$ we have $|M(p_{in})| = 0$.

**Proof.** We proceed by structural induction. The base cases are those of the assign, silent, noaction and interaction activities, which are all immediate. For the general case, we have that for all the constructions, namely, choreography (Fig. 3), data workunits (Figs. 10 and 11), sequence (Fig. 12), parallel (Fig. 13) and choice (Fig. 14), as there is no arc reaching $p_{in}$, it cannot be marked again. □
Proposition 2. Let \( N = (P, T, A, V, G, E, \lambda, D, \pi) \) be a PTPN obtained from a WS-CDL document by applying the translation. As initial marking \( M_0 \) of \( N \) we take one token at the initial place, and one token with value \( e \) on each \( \mathit{p_{in}} \) place. Then, for any reachable timed marking of \((N, M_0)\) we will have at most one token on every place. Furthermore, in the final marking of the PTPN only one of the two exit places can be marked.

Proof. By structural induction, the base cases are those corresponding to the basic activities (interactions, assign, silent and no action), which are all immediate. For the general case we must distinguish the following cases:

- **Choreography:** We only have the root choreography, which is therefore the outermost element in the syntax of the WS-CDL document. It is translated as indicated in Fig. 3, which just joins some places of the PTPNs corresponding to the body and the exception block of the choreography. A simple application of the induction hypothesis thus allows us to conclude the property.

- **Workunits:** For the delayed workunit, using the structural induction hypothesis it follows that the PTPN for the activity inside the workunit is 1-safe. The modified PTPN (Fig. 9) for this activity now only replicates its initial transitions, with the purpose of iterating its behavior when the final guard is true. Using the induction hypothesis we can then conclude from this construction that the PTPN obtained for the delayed workunit is 1-safe.

  For the data workunits we distinguish two cases (Figs. 10 and 11). In both cases we use the same modification as in delayed workunits for the repetition (by replication of the initial transitions). Thus, again using the induction hypothesis and these constructions we conclude that the obtained PTPNs are 1-safe.
• **Ordering structures:** Figs. 12–14 illustrate the translation for the ordering structures (sequence, parallel and choice). In all of them we simply apply the induction hypothesis and inspect the possible behaviors of these PTCPNs to conclude the 1-safeness. □

**Definition 7.** Let $N = (P, T, A, V, G, E, \lambda, D, \pi)$ be a PTCPN, with an initial marking $M_0$. We say that $(N, M_0)$ is clean if and only if the following conditions hold for any reachable timed marking $(M, x)$:

1. $|M(p_m)| = 1 \Rightarrow |M(p)| = 0$, $\forall p \neq p_m$, i.e. if the initial place $p_m$ is marked with one token, no other place can be marked, excepting those associated with variables.
2. $|M(p_a)| = 1 \Rightarrow |M(p)| = 0$, $\forall p \neq p_a$, i.e. if the correct exit place $p_a$ is marked with one token, no other place can be marked, excepting those associated with variables.
3. $|M(p_r)| = 1 \Rightarrow (\forall t \in T \exists p \in \cdot t$ such that $|M(p)| = 0)$, i.e. all the transitions are dead. □

**Proposition 3.** All the PTCPNs obtained from a WS-CDL document by applying the translation are clean.

**Proof.** Again by structural induction. The base cases are all immediate (interactions, assign, silent and noaction).

For the general case we distinguish the following cases:

• **Choreography:** We have the construction indicated in Fig. 3. Then, if $p_{in}$ or $p_{ok}$ are marked with one token, we only need to apply the induction hypothesis on $N_a$, and no place can be marked in $N_a$, excepting those corresponding to variables. If $p_{er}$ becomes marked with one token, using the induction hypothesis on $N_1$, it follows that all transitions in $N_a$ are dead. Furthermore, the activation of $N_1$ could only be made by putting one token on $p_{ok}$, which again implies by the induction hypothesis on $N_1$ that all the transitions in $N_a$ were dead at that time, and obviously remain dead when $p_{er}$ becomes marked.

• **Workunits:** The reasoning is very similar for the delayed workunits and the two versions of the data workunits, so we only give here the proof for the delayed workunits.

In this case (Fig. 9), if $p_{in}$ is marked, no other place can be marked (except those corresponding to variables), since we are still at the initial marking. If $p_{ok}$ is marked, we must have fired transition $t$, which, in turn, has required one token on $p_{ok}$ and $g'$ evaluated to be false. Thus, using the induction hypothesis on $N_1$, we conclude that no other place is marked at that time on $N_a$, which also occurs on its modified version. As a consequence of the construction, $p_a$ cannot be marked either. In the event of $p_{er}$ becoming marked with one token, since $p_{er} = p_{er1}$, using the induction hypothesis we will have that all the transitions in $N_{A2}$, are dead, which is also the case in the modified version.

• **Sequence:** When $p_{in}$ is marked, by the induction hypothesis no other place is marked in $N_{A1}$, except those associated to variables. As $N_{A1}$ has not yet been activated, no tokens can appear on their places (except those of variables).

If $p_{ok}$ becomes marked, since $p_{ok} = p_{ok1}$, we can apply the induction hypothesis to conclude that no other place is marked in $N_{A2}$, except those associated to variables. The activation of $N_{A2}$ required one token on $p_{ok}$, i.e., on $p_{ok1}$, so we can apply the induction hypothesis on this marking to conclude that no other place could be marked at that time on $N_{A2}$, except those of variables. As a consequence, since $N_{A2}$ cannot have evolved from that time, we conclude the property.

Finally, in the event of $p_{er}$ having one token, since $p_{er} = p_{er1} = p_{er2}$, it follows that either $N_{A1}$ or $N_{A2}$ has failed, and all its transitions are dead. If it is $N_{A1}$, it follows that $N_{A2}$ has not yet been activated, so none of their transitions can ever be fired. If $N_{A2}$ has failed, it follows that $p_{ok}$ was to be marked with one token to activate $N_{A2}$, i.e., using the induction hypothesis no place remains marked in $N_{A2}$, except those of variables, so all its transitions are also dead.

• **Parallel:** If $p_{in}$ is marked with one token, from Fig. 13 it is immediate that no other place can be marked, except those of variables, since there are no arcs returning to $p_{in}$, so it can only be the initial marking.

When $p_{ok}$ becomes marked with one token, we must have fired transition $t_2$, which has required one token on $p_t$, $p_{Atok}$ and $p_{At2}$. We can then apply the induction hypothesis on $N_{A1}$ and $N_{A2}$, in order to conclude that no other place is marked on these nets, except those associated with variables ($p_t$ does not affect the firing of transitions, as it is marked).

In the event of $p_{er}$ becoming marked with one token, we must have fired a fail transition in one of the PTCPNs. This firing also removes the token on $p_t$, so no other transition can be fired from that moment on, i.e. all the transitions are dead.

• **Choice:** A simple application of the induction hypothesis allows us to conclude the property, as we may evolve either on $N_{A1}$ or $N_{A2}$, and when both can fail initially we have collapsed this failure in a single transition which removes the token on $p_{in}$ and produces a new token on $p_{er}$. □

8. Case study: 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 h must elapse after getting the information on available seats to make a reservation. Reservations are only valid for a period of 48 h, which means that if they have not been confirmed and paid for in two days they are canceled, and the seats are released.
Fig. 17 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 h. 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 Fig. 18 (top) and 19 (bottom), which are connected by the common places $P_6$ and $P_{error}$.

Looking at Fig. 18 we can see a transition $t_1$ that forks both initial parallel activities, which correspond to two assign activities and two interactions activities. Both parallel activities join by means of transition $t_2$. The firing of $t_2$ activates the execution of the three sequential assign activities, which can be seen as three assign transitions in a row in Fig. 19. After their execution the delayed workunit starts, transition $tt$ captures the 4 h delay, after which a token is generated on $P_{10}$. 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 an associated time-out, and transition $t_3$ is the final transition of the enclosing delayed workunit.
8.1. Case study 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 Fig. 20), and obtaining the corresponding state space report. From this report we can deduce the following properties:

- 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, $P_{13}$, $P_{14}$ and $P_{16}$, 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.

9. Conclusions

In this paper we have presented a PTCPN semantics for a relevant subset of WS-CDL, in which integer data variables, time restrictions and priorities are considered. The introduction of priorities allows the parties of a Web Composition to give priority to certain interactions, which can be useful in many situations, for instance, to distinguish clients or items, as has been shown in the case study. Time restrictions have also been considered in the translation, both in interactions (time-outs) and in workunits, to delay the execution. The PTCPNs obtained are 1-safe and clean, which means that only one token can occupy a place in any reachable marking. 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 may remain in some places when the error place has been marked.
The main advantage of this translation is that the PTCPNs obtained are currently supported by CPN Tools [18], a widely used and recognized Petri nets tool that allows us to simulate and even verify some properties of the described system. We have then developed a tool (WST) to support the translation to a file format that can be immediately used by CPN Tools. Thus, from a WS-CDL XML document, we can obtain the corresponding PTCPN model and use CPN Tools to simulate and verify some properties of the given choreography.

The official semantics of WS-CDL [40] is defined in a textual manner, so that another important advantage of the PTCPN semantics is that it can be used as an alternative to the textual document in order to obtain the WS-CDL semantics, in a more rigorous way. In fact, as we have seen in this paper, some points of the WS-CDL semantics have not been completely described, and a formalization also serves to detect these deficiencies.

Another important result of the translation is that it could be used to obtain the individual behavior of each of the parties. Transitions of the obtained PTCPN could easily be labeled with the RoleTypes involved in their execution. Using this information we could extract the PTCPNs skeletons of each RoleType, and these PTCPNs could be used as a high-level design for them. We thus obtain as subproduct a first design for the different RoleTypes, which can be progressively detailed by refinements.

In future work we plan to extend the translation supporting a richer WS-CDL subset, e.g. the inclusion of a hierarchy of choreographies and finalizer 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 an expired time-out. However, there are other situations (mainly related to variables) that cause abnormal termination that could be considered in an extended version of this work.

References