A Petri net approach for the design and analysis of Web Services Choreographies

Valentín Valero, M. Emilia Cambronero, Gregorio Díaz*, Hermenegilda Macià

Departamento de Sistemas Informáticos, Escuela Politécnica Superior de Albacete, Universidad de Castilla-La Mancha, Campus Universitario s/n, 02071 Albacete, Spain

ARTICLE INFO
Available online 1 November 2008

Keywords:
Web services
Web service composition
Priorities
Choreography
Time Petri nets

ABSTRACT

A Web Service is a self-describing, self-contained modular application that can be published, located, and invoked over a network, e.g. the Internet. Web Services composition provides a way to obtain value-added services by combining several Web Services. The composition of Web Services is, therefore, suitable to support enterprise application integration. WS-CDL (Web Services Choreography Description Language) is a W3C candidate recommendation for the description of peer-to-peer collaborations for the participants in a Web Services composition. In this paper we focus our attention on the development of a methodology for the design and validation of composite Web Services using WS-CDL as the language for describing Web Services interactions and Petri nets as a formalism that allows us to simulate and validate the described systems. We specifically intend, then, to capture timed and prioritized collaborations in composite Web Services, so the model of Petri nets that we use is a prioritized version of Time Petri nets.

© 2008 Elsevier Inc. All rights reserved.

1. Introduction

A Web Service can be defined [4] 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 that can be obtained through the Internet. Web Services are becoming more and more important as a platform for B2B integration and Web Services composition has appeared as a natural and elegant way to provide new value-added services as a combination of several established Web Services. Hence, services provided by different suppliers can act together to provide another service; in fact, they can be written in different languages and executed on different platforms. Web Services composition thus provides a mechanism for distributed software integration, where different enterprise solutions cooperate to achieve a common goal.

Web services current technology is based on the Web Service architecture stack, proposed by the World Wide Web Consortium, W3C [31], which consists of the main following components: SOAP, WSDL, Registry (UDDI), Security layer, Reliable Messaging layer, Context, Coordination and Transaction layer, Business Process Languages layer (WSBPEL) and Choreography layer. The three basic layers are the SOAP, WSDL and UDDI. The SOAP layer describes the message format and delivery options, the WSDL language describes the static interface of a Web Service, whereas the UDDI layer makes a Web Service visible and available. The intermediate layers, security, reliable messaging, context, coordination and transaction layers provide a wide range of quality properties for the communications process. Finally, the highest and more abstract...
layers are the Business Process Languages and the Choreography layers. The Business Process Languages layer describes the execution logic by defining its control flow and prescribing the rules for managing its non-observable data, and is also known as the Orchestration layer. The Choreography layer describes collaborations of parties by defining from a global viewpoint their common and complementary observable behaviour, where information exchanges occur, when the jointly agreed ordering rules are satisfied. One of the most widely spread W3C pre-standardized protocols for this layer is the Choreography Description Language [30], WS-CDL for short.

The composition of Web Services entails the integration of the requirements of each component. These requirements include the format of the messages exchanged among the parties, the channels used in the communications, the type of communications (request and/or response), the control flow, exception handling, but also timed and prioritized aspects can be considered, and all of these requirements can be covered by the choreography layer. Our intention in this paper, then, is the development of a methodology for the design of composite Web Services, considering many of the described requirements, with special attention to timed and prioritized interactions. Our starting point are Web Services descriptions written in WS-CDL, after which we provide an automatic translation to a prioritized-timed model of Petri nets.

Our goal is twofold. On the one hand, we describe a methodology for developing web service compositions by using WS-CDL as a basic description language, whilst on the other we obtain a graphical representation of WS-CDL compositions behaviour in terms of prioritized-timed Petri nets, which can be very helpful for the software designer in order to have a complete view of the Composite Web Service and the interactions that take place among the different participants. But Petri nets are also a formal tool and they allow us to describe not only a static vision of a system, but also its dynamic behaviour. We can then use the Petri net representation to validate and verify the Composed Web Service.

Two examples can help clarify the relevance of this work. The first is an airline ticket reservation system, in which there are two types of client, the travelers, who can make their reservations themselves, and the travel agents, whose requests have a lower priority than those made by the travelers. In this system the reservations that both kinds of client make are only valid for a bounded period of time, so the interactions that confirm the reservations have a time-out associated. This example thus allows us to illustrate how time requirements can be introduced into the interactions among the parties.

Our second example is a supplier service, which attends to client requests according to a scheme of priorities, established depending on the client types. We can think in principle that priorities could be dealt with only by the supplier server, but we consider that it is beneficial for all parties involved to be aware of this information, because, on the basis of this knowledge, the clients may decide to change their contract type in order to get a better service.

We have structured the paper as follows: A discussion of related work is shown in Section 2. The Collaborative Web Service-Petri Nets (CWS-PN) methodology is introduced in Section 3; in Section 4 we present a brief description of the main elements of WS-CDL and we show how to introduce priorities in WS-CDL. The particular model of prioritized-timed Petri net that we use is introduced in Section 5 and the translation is shown in Section 6. The two examples above mentioned are described in Section 7. Finally, in Section 8 the conclusions and some indications about our future work are formulated.

2. Related work

The developers of WS-CDL claim that its design has been based on a formal language, the $\pi$-calculus [22], and that therefore WS-CDL is a particularly well-suited language for describing concurrent processes and dynamic interconnection scenarios. This relationship has been studied in [12], where the authors compare a formalised version of WS-CDL, called global calculus, with the $\pi$-calculus. They discuss how the same business protocols can be described by WS-CDL and $\pi$-calculus equivalently, as two different ways of describing communication-centred software in the form of formal calculi.

In [11] a model based on state machines is used to represent and analyse conversations of Web Services. Yang et al. [33] have 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 has been recently extended [23] by including a projection of the choreography level into the orchestration level, the dominant role concept being introduced which is used in the implementation of any choice or interaction structure of the choreography.

Thomas et al. [27] have defined a timed Petri net representation of Web Services Flows; in this case, only the flow of messages and methods are considered, the starting point being WSDL (Web Service Description Language) [32]. Hamadi and Benatallah [15] have proposed a Petri net-based algebra to model Web Services Control flows; hence, constructions such as sequence, choice, iteration, parallelism, discriminator, selection and refinement are studied in that paper, but they omit consideration of timed or prioritized interactions. There are some works defining translations from BPEL4WS [6] to some specific classes of Petri nets, Martens [19], for instance, defines a translation to a particular class of Petri nets called workflow modules and, based on this formalism, notions like compatibility and usability are defined and studied. Verbeek and van der Aalst [28] have also defined a translation of the main activities of BPEL4WS into a class of Petri nets, named workflow nets (WF-nets). However, none of these works consider time or priorities, and all of them work on BPEL4WS, which supports the modelling and implementation of individual executable processes at the orchestration level. By contrast, WS-CDL is a proposal for specifying the interactions among the participants involved in a business process from a global point of view. Some authors advocate the use of abstract BPEL as a choreography language: in [2] van der Aalst et al. take as starting point a specification written in abstract BPEL, automatically obtaining a Petri net representation for the intended choreography.
Then, the authors address the problem of verifying whether the interaction between the orchestrated individual services conforms the conversation specified in the choreography specification.

In contrast to BPEL4WS, with WS-CDL we have a global description of the collaborations among the parties from an independent viewpoint of its participant processes. Additionally, in this work we not only capture the flow of collaborations in terms of activities of WS-CDL, but also enrich the description by adding time restrictions and priorities to the interactions. There are also translations that use algebraic models: Salaun et al. [25] have defined a process algebra to derive the interactive behaviour of a business process starting from a BPEL4WS specification, Brogi et al. [9] have defined a translation of WSCI (Web Service Choreography Interface) to CCS [21], showing the benefits of such translation, and Yeung [34] has defined a mapping from WS-CDL and BPEL4WS into CSP, providing a formal approach to verifying the behaviour of collaborating Web services.

3. CWS-PNs methodology

The proposed methodology, Collaborative Web Service Petri Nets (CWS-PNs for short), consists of three phases (Fig. 1): analysis, design and model validation. The analysis is performed by using the KAOS goal model [18]. This goal model allows analysts and specifiers to gather the requirements of software systems in a hierarchical order, i.e. from general and strategic goals to concrete requirements. Goals are objectives the system under construction must achieve, and with the KAOS technique a structured goal model is constructed as an AND–OR graph.

Thus, goals are organized in AND/OR refinement-abstraction hierarchies, where higher-level goals are in general strategic, coarse-grained and involve multiple roles, whereas lower-level goals are in general technical, and involve fewer roles. In such structures, AND-refinement links relate a goal to a set of subgoals possibly conjoined with domain properties; this means that satisfying all subgoals in the refinement is a sufficient condition in the domain for satisfying the goal. OR-refinement links may relate a goal to a set of alternative refinements and, in this case, it is enough to satisfy just one of the subgoals.

In the analysis phase, for instance, we gather the time requirements, such as deadlines, time-outs and any other constraints where time plays a crucial role. We can think of a travel reservation system, where once you have made a reservation you have a period of time for performing the payment, or of a supplier system, where the products must be dispatched to the clients in a limited period of time. All of these restrictions are considered in the corresponding KAOS goal model of these examples (see Figs. 13 and 16).

After the analysis, the design phase starts. To complete this phase we use the information about the entities that collaborate in the system, the constraints that have been identified and then, a WS-CDL document is produced as a result of the design phase. In the travel reservation system example, we can identify three entities: the traveler, the travel agent and the airline reservation system. Travelers and travel agents make requests for reservations, and the time restrictions referred to their requests are concerned with the maximum time that a reservation is maintained. With this information we can produce a WS-CDL document that captures the relationship between these roletypes.

The third phase of the proposed methodology is devoted to the validation and verification of the system model. Model validation and verification are a key instrument for developing correct software systems; in that sense Hoare [17], Clarke et al. [14], together with a large number of authors have agreed on its importance [16]. In order to accomplish this objective we use a timed-prioritized extension of Petri Nets (PTPNs), providing an automatic translation of WS-CDL to PTPNs for that purpose.

The specific model of timed-prioritized Petri nets that we use is an extension of Merlin’s nets [20]. With this model we are able to capture the prioritized and timed interactions of the different participants of a composite Web Service. Therefore,
by means of the translation that we introduce in this paper, we provide a way to simulate and verify the system behaviour, by using a tool supporting the Petri net model.

The validation and verification of the system model is error-driven, in the sense that once we have detected a failure we go back to the design or the analysis phase in order to fix it, after which, the PTPN must be built again. Thus, we can profit from one of the main benefits of the use of formal techniques, specifically for validation and verification purposes, namely, the early detection of errors, which are identified in the initial phases, and can be corrected before the software comes into use.

4. WS-CDL

The Web Services Choreography specification offers a precise description of 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 behaviour of all the parties involved. Each party can use the global definition to build and test solutions that conform to it. The global specification is, in turn, brought about by a combination of the resulting local systems, on the basis of appropriate infrastructure support.

In real-world scenarios, corporate entities are often unwilling to delegate control of their business processes to their integration partners. Thus, choreography offers a means by which the rules of participation within a collaboration can be clearly defined and agreed to jointly. Each entity may implement its portion of the Choreography as determined by the common or global view. It is the aim of WS-CDL that the conformance of each implementation to the common view expressed in WS-CDL is easy to determine.

The WS-CDL model consists of the following entities [30]:

- **Participant Types, Role Types and Relationship Types.** A Participant Type groups together those parts of the observable behaviour that must be implemented by the same logical entity or organization. A Role Type enumerates the observable behaviour a party exhibits in order to collaborate with other parties. A Relationship Type identifies 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. 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 Variable. Both Variables and Tokens have Types that define the structure of what the Variable contains or the Token references.

- **Choreographies:** As stated earlier, these establish the common rules that govern the ordering of exchanged messages and the collaborative behaviour. They consist of three parts:
  - **Choreography Life-line:** This describes the progression of a collaboration. Initially, the collaboration is established between the parties; then, 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).

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

- **WorkUnits** prescribe the constraints that must be fulfilled for making progress and describe some activities within a Choreography.

- **Activities and Ordering Structures.** Activities describe the work that the Choreography must perform. There are basic activities (which perform the lowest level actions) and ordering structures. Ordering structures combine activities with other Ordering Structures in a nested structure to express the ordering conditions under which information within the Choreography is exchanged. One of the basic activities supported by WS-CDL is interaction activities, which describe the exchange of information between parties, the possible synchronizations of their observable information changes and the actual values of the exchanged information. They can be assigned a time-out, i.e. a time to be completed. Consequently, time information in WS-CDL can appear both in the interactions (time-outs) and also in date/time variables (using XPath).

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

```xml
<timeout time-to-complete="XPath-expression"
    fromRoleTypeRecordRef="list of NCName"?
    toRoleTypeRecordRef="list of NCName"?/>
```

In the **time-to-complete** attribute the timeframe in which an interaction must complete is specified. Hence, when this time expires (after the interaction was initiated), if the interaction has not completed, the time-out occurs and the interaction finishes abnormally, causing an exception block to be executed in the choreography. The optional attributes **fromRoleTypeRecordRef** and **toRoleTypeRecordRef** are XML-Schema lists of references to record elements that will take effect at both roleTypes of the interaction. XPath 2.0 supports date and time variables, so we can also use these variables in WS-CDL. Actually, XPath provides a number of functions to manage these datatype values. These variables can be used in particular to delay the
execution for a certain time, or to establish the instants at which some actions must be executed. For that purpose, we may use the guards of workunits, by including in a guard an expression related with the value of a time variable. In fact, as we intend to capture delays or instants of execution, the specific expressions allowed are those constructed using the operators \( \text{op:time-equal} \), \( \text{op:time-less-than} \) and \( \text{op:time-greater-than} \) of XPath 2.0.

4.1. WS-CDL with priorities

In many cases, it may be desirable to favour some interactions over others, i.e. in the composition of Web Services, some parties can express their interest in the prioritization of certain interactions. We can think of a Web Service for selling or reserving items of different sorts. Therefore, clients interact with the Web Server to buy or reserve items, but these interactions may have associated different levels of priority, depending on the kind of item or even on the client that makes the request. WS-CDL has a choice construct, which allows us to choose among some different activities. However, the textual description in [30] is somewhat too vague on this. It states that “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”. But it also states that “if the choice has other activities, it is assumed that the selection criteria for those activities are non-observable”. From this description it is not clear what should be done when both guarded activities and non-guarded activities appear as alternative 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 case of change of priorities), and it does not allow us to consider several interactions with the same priority. In [33] the authors have solved the problem by distinguishing two types of choice, non-deterministic and general choice (guarded workunits). In our case we have decided to equip interactions with priorities, and then, the highest priority is selected for execution. When we have several interactions with the same priority (the maximum), the choice is non-deterministically resolved.

Accordingly, we propose an extension of WS-CDL with priorities. Priorities are established as natural numbers, with the usual interpretation, the greater the number, the greater 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 priority, in which we indicate the priority level of the corresponding interaction (see Fig. 2 for the specific syntax that we propose).

The interpretation of this attribute is the natural one, in case of conflict only the highest priority interactions are allowed.

5. Prioritized-time Petri nets

In this section, we introduce the specific model of timed-prioritized Petri net considered for the translation. In the literature about timed extensions of Petri nets we can identify a first group of models, which assign time delays to transitions, either using a fixed and deterministic value [24,26] or choosing it from a probability distribution [3]. Other models use time intervals to establish the enabling times of transitions [20]. Finally, we also have some models that introduce time on tokens [1]. In [29] the interested reader may find a description of the different approaches used to introduce time in Petri nets.

Priorities were also introduced in Petri nets to extend the power of description of the model [7,8], usually by associating priority levels to transitions and modifying the firing rule to prevent the firing of a transition when another one having
a greater priority is enabled. In [10] a model can be found that extends Merlin’s nets by including dynamic priorities and resources.

The particular model that we use is also an extension of Merlin’s nets, including priorities and two transition types (black and white). All transitions are assigned both a time interval and a static priority. The time interval restricts the instants at which a transition is allowed to be fired. White transitions are not forced to fire when their clock reaches its latest firing time, whereas the black ones must fire once their clock reaches that value (unless they are involved in a conflict). The priority is used to resolve conflicts. Hence, priorities are only used in case of conflict, when at a given instant two or more enabled transitions compete for firing, only a highest priority transition is allowed to be fired at that moment.

**Definition 1 (Prioritized-Time Petri Nets).** We define a prioritized-time Petri net (PTPN) as a tuple $N = (P, T, F, \alpha, \beta, \pi)$, where $P$ is a finite set of places, $T$ is a finite set of transitions ($P \cap T = \emptyset$), such that $T = T_1 \cup T_2$, with $T_1 \cap T_2 = \emptyset$. Transitions in $T_1$ are called white, whereas transitions in $T_2$ are called black. $F$ is the flow relation ($F \subseteq (P \times T) \cup (T \times P)$), $\alpha$ and $\beta$ define the time intervals that restrict the firing of transitions, $\alpha : T \to \mathbb{N}$, $\beta : T \to \mathbb{N} \cup \{\infty\}$, where $\mathbb{N} = \{0, 1, 2, \ldots\}$, and fulfilling $\alpha(t) \leq \beta(t), \forall t \in T$. Finally, $\pi$ is the priority function, $\pi : T \to \mathbb{N}$, which assigns a priority level to each transition.

We use the classical notation on Petri nets to denote the precondition and postcondition of both places and transitions:

$$\forall x \in P \cup T : \ast x = \{y | (y, x) \in F\} \ast x = \{y | (x, y) \in F\}$$

Markings are defined in the usual way, as an annotation of tokens over places, hence a marking $M$ is formally defined as a function $M : P \to \mathbb{N}$, which indicates the number of tokens on each place.

The semantics of PTPNs is captured by the following definitions, which extend the firing rule of Merlin’s nets by considering the priority information and the two transition types that we have introduced.

**Definition 2 (Enabling transitions).** Given a PTPN $N = (P, T, F, \alpha, \beta, \pi)$, a marking $M$ of it and a transition $t \in T$, we say that $t$ is enabled at $M$ if each of its input places contains at least one token, i.e. $\forall p \in \ast t, M(p) > 0$, i.e. it is enabled in the underlying Petri net $(P, T, F)$. As usual, we denote this by $M(t)$ and the set of transitions enabled at $M$ by $E(N, M)$.

We restrict our attention to a particular class of PTPNs, for which no transition will be enabled more than once at a time, i.e. it will never be the case that two or more instances of the same transition are enabled at a certain instant. With this restriction we avoid the semantic problems that appear in Merlin’s nets when multiple enabling of transitions are allowed (see [29]).

**Definition 3 (States in PTPNs).** Given a PTPN $N = (P, T, F, \alpha, \beta, \pi)$, we define a state of it as a pair $(M, I)$, where $M$ is a marking and $I$ is a function $I : E(N, M) \to \mathbb{N} \times (\mathbb{Z} \cup \{\infty\})$, which is defined for enabled transitions, and indicates the lower and upper time bounds that have to be fired with respect to the current instant. Negative values can appear in the upper bounds for white transitions that were not fired once their maximum firing time was reached. As long as these transitions remain enabled, their upper time bound decreases as time elapses.

For $I(t) = (x_1, x_2)$, we will denote $x_i$ by $\Pi_i(I(t))$ for $i = 1, 2$. The initial state of a PTPN is defined by considering an initial marking $M_0$ and the function $I_0$ defined as follows: $I_0(t) = (\alpha(t), \beta(t)), \forall t \in E(N, M_0)$.

Now the firing rule can be precisely defined, but we first need a function capturing time elapsing.

**Definition 4 (Time elapsing).** Given a PTPN $N = (P, T, F, \alpha, \beta, \pi)$ and a state of it $(M, I)$, we say that $x$ units of time can elapse if either $E(N, M) \cap T_2 = \emptyset$ or for every $t \in E(N, M) \cap T_2$ we have $\Pi_2(I(t)) \geq x$. In that case, the new state reached after that time will be $(M', I')$, where $\forall t \in E(N, M), I'(t) = (x_1 - x, x_2 - x)$, taking $I(t) = (x_1, x_2)$ and $x = \text{Max}(0, x - y)$.

From this definition we can see that white transitions may lose their opportunity to fire, if they are not fired when their clock has reached the latest firing time. In that case, their upper time bound will become negative, which prevents their execution. Nevertheless, this does not mean that they are definitely dead, because the tokens on their preconditions can be used to fire other transitions, and they can become enabled again later.

We now establish the condition that the enabled transitions must fulfill in order to be fireable.

**Definition 5 (Potentially Fireable Transitions).** Given a PTPN $N = (P, T, F, \alpha, \beta, \pi)$, a state of it $(M, I)$ and an enabled transition $t \in E(N, M)$, we say that $t$ is potentially fireable at that state if and only if its earliest firing time is 0 and its latest firing time is greater than or equal to 0: $\Pi_1(I(t)) = 0 \land \Pi_2(I(t)) \geq 0$.

---

1 $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$ is the set of integer numbers.
Given a marking at a certain instant, we may have several potentially fireable transitions that want to be fired at that instant; one of these transitions can then be fired at that instant if there is no other potentially fireable transition with a greater priority that wants to be fired at the same instant. Thus, the evolution of a PTPN is defined in two phases, in the first one we select the set of potentially fireable transitions that want to be fired at the current instant, and in the second one we fire one of the highest priority transitions. Notice that if a black transition is potentially fireable at the current instant, and its upper time bound is 0, no time can elapse until this transition is selected for firing, unless it is in conflict with another one that is fired at that same instant.

**Definition 6 (Firing rule).** Given a PTPN \( N = (P, T, F, \alpha, \beta, \pi) \), a state of it \( (M, I) \) and a set of potentially fireable transitions \( B \), a transition \( t \in B \) can be fired at that state if and only if \( \exists \ t' \in B, \pi(t') > \pi(t) \).

The firing of \( t \) leads us to a new state, \( (M', I') \), which is defined as follows:

1. The marking \( M' \) is obtained by applying the classical firing rule on Petri nets, i.e. \( M'(p) = M(p) - W_F(p, t) + W_F(t, p) \), where \( W_F \) is defined as follows: \( W_F : (P \times T) \cup (T \times P) \to (0, 1) \), \( W_F(a) = 1 \) for \( a \in F \), and \( W_F(a) = 0 \) for \( a \notin F \).
2. For every transition \( t' \in E(N, M) \cap E(N, M') \), \( t' \neq t \), we take \( I'(t') = I(t') \).
3. For every transition \( t' \in E(N, M') \setminus E(N, M) \) we take \( I'(t') = (\alpha(t'), \beta(t')) \).
4. In the case that \( t \in E(N, M') \) we take \( I'(t) = (\alpha(t), \beta(t)) \).

Notice that firing a transition takes no time to complete, so we keep in the new state the time restrictions of the transitions that were enabled before the firing and remain enabled after it. It can also be that the fired transition becomes enabled again at the new marking, in which case it should be noted that its local clock is reset.

**Example 1.** In Fig. 3 we can see the graphical representation of PTPNs. Transitions are painted in black and white according to their type. They are annotated both with the priority level and the static time interval associated. In the adjacent table we can see the time bounds that the enabled transitions have at the current state of the system. Observe that \( t_3 \) is potentially fireable at that state, in fact it must be fired, and no time can elapse because it is black and its latest firing time is 0.

### 6. PTPN semantics for WS-CDL with priorities

In this section, we provide a PTPN semantics for a subset of WS-CDL with priorities. Of course, the syntax of WS-CDL is too vast for that purpose, so we need to restrict it to a subset of WS-CDL, only considering the elements of WS-CDL related with the flow of collaborations in terms of activities, and further taking into account the time restrictions and priorities in the interactions.

Thus, our goal is to obtain a PTPN representation that captures the main aspects of the Web Services composition, and specially those related with time and priorities. This representation will capture the visible behaviour of the participants of a Web Service composition and their interactions. The information about the actions made by the parties is therefore very important in this context, so in the PTPN representation we will label each transition with the roletypes that are involved in its execution. Note that it can also happen that no specific RoleType is involved in the execution of a transition, in which case we will omit this information in the graphical representation of the PTPN.

The obtained PTPNs 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 PTPNs will have one initial place,\(^2\) which activates the PTPN 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 PTPN.

\(^2\) This does not mean that this is the only initially marked place.
The starting point is a WS-CDL document with the syntax of interactions extended considering priorities (Fig. 2). The different elements of the document are translated as follows:

- **RoleTypes**: These are used to enumerate the observable behaviour of each party. As stated earlier, in the PTPN representation transitions are labelled with the roletypes involved in their execution.
- **RelationShipTypes**: These are used in interactions, so they are (implicitly) considered in the translation provided for interaction activities.
- **ParticipantTypes**: We do not need to translate these elements, because they are only used to group together some previously declared roletypes.
- **ChannelTypes**: These are used in interactions, so they are implicitly considered in the interaction activity translation.
- **Information types and Variables**: These are used in interactions, so they are (implicitly) considered in the translation provided for interaction activities.
- **Choreographies**: This is, of course, the main element of the WS-CDL document. A Choreography describes the activities to be made for the different participants, and it can contain an exception block and a finalizer block. Therefore, translating compositionally each one of these elements we have:

\[
N_a = (p_a, T_a, F_a, a_0, p_0, \pi_0) \quad \text{(PTPN for the activities)} \\
N_e = (p_e, T_e, F_e, a_0, p_0, \pi_e) \quad \text{(PTPN for the exception block)} \\
N_f = (p_f, T_f, F_f, a_0, p_0, \pi_f) \quad \text{(PTPN for the finalizer block)}
\]

Let \(p_{ain}, p_{ein}\) and \(p_{fin}\) be the initial places of \(N_a, N_e\) and \(N_f\) respectively; \(p_{ain}, p_{ein}, p_{fin}\) their correct exit places, and \(p_{aer}, p_{eer}, p_{fer}\) their erroneous exit places. Thus, the PTPN for the choreography is that shown in Fig. 4. Let us note that the initial place of the choreography \((p_{ain})\) is exactly \(p_{ain}\), its erroneous exit place is \(p_{eer}\); and this place coincides with the places \(p_{ein}\) and \(p_{ein}\) when the choreography has an exception block, because, when a choreography terminates by executing an exception block, it is considered to be terminating abnormally. Otherwise, if the choreography does not have an exception block, we take \(p_{aer} = p_{eer}\). Notice that according to the WS-CDL terminology the choreography enters into a closed state when it terminates, either normally or abnormally, i.e. there is no distinction about how it has been closed. However, in our model we capture this information, as we are distinguishing the different ways we have to close the choreographies.

The description of the translation of the finalizer block and the corresponding finalize activities is delayed to Section 6.6.

- **Activities**: We may have basic activities, workunits, ordering structures activities and choreographies composition activities. Furthermore, choreographies may have exception and finalizer blocks. The translation for each one is shown in the following subsections.

### 6.1. Basic activities

The basic activities of WS-CDL that we consider are interaction activities, assign, silent and noaction activities.

- Interaction activities: An interaction activity involves two roletypes, and an exchange of information between them. Actually, in WS-CDL several exchanges of information are allowed in a single interaction, and they can be either request or respond type, and these actions can be synchronous or asynchronous, depending on the align attribute. For the sake of the specific syntax of each element, which can be found in the WS-CDL description document [30], and we also omit the formal definitions of the PTPNs obtained for each case, which can be easily deduced from the figures.
of simplicity, we will only consider an interaction as a simple synchronous exchange of information, but we abstract
from the information itself. The interaction is thus translated as a transition that corresponds to the rendez-vous of both
role-types at this point of the choreography. A similar vision for the interactions of choreographies is considered in [9],
where Brogi et al. consider an algebraic formalization for WSCI (Web Services Choreography Interface), and interactions
are just seen as communications of CCS.
In WS-CDL interactions may have a time-out associated, in which case, when the time-out expires we consider that they
finish abnormally. Furthermore, a priority attribute may have been indicated, this value being used as the priority for the
corresponding transition in the PTPN representation, otherwise it has priority 0. Fig. 5 illustrates this translation: part (a)
shows the translation when a time-out has been indicated, \( x \) is the maximum time we have for this interaction to complete
(time-out), \( l \) is the priority level (0 when it is not indicated), and \( r_1, r_2 \) are the role types involved in the interaction. In part
(b) of this figure we see that if a time-out is not indicated, we do not consider an abnormal termination (\( p_{er} \) is isolated).

- Assign, Silent and Noaction activities: These are translated in the same way (Fig. 6), by means of a single transition with
priority 0, labelled with the role type that executes this basic activity. As the time required to execute this kind of actions
is negligible we consider them as immediate. We also consider that these basic activities cannot finish abnormally.

6.2. Workunits

As we have said before, we allow the use of time variables in WS-CDL, which can be used to delay the execution of a
workunit, by using the guard facility. Furthermore, a workunit may have more general guards, which can be considered in the
PTPN model. Hence, we consider a separate translation for both kinds of guards, on the one hand, we provide the translation
for the general case (Fig. 7a), in which a guard is evaluated as true or false; when it is true the activities inside the workunit
are executed, whereas when it is false the behaviour depends on the block attribute of the workunit. If it is true the workunit
remains blocked until the guard condition changes, otherwise it is skipped.
We also consider the translation for the case in which a time variable is used in a guard (Fig. 7b), but in this case the block
value must be true to enforce the delay, and no repetition condition can have been indicated.
In both figures \( N_a \) is the PTPN obtained compositionally for the activities inside the workunit.
For the general case (Fig. 7a), \( t_1, t_2 \) are used to capture the guard condition evaluation. When the guard condition is true
\( p_{oin} \) becomes marked and the activity inside the workunit starts. Otherwise, \( \neg g \) is marked, but note that it is even possible
to move the token from \( \neg g \) to \( p_{oin} \), because the variables involved may change their value while the system is blocked.
Place \( \neg block \) will only be (initially) marked with one token if the block attribute in the workunit has been put to false.
Therefore, from the WS-CDL document we know whether it is initially marked or not, but note that if it is marked, it will
remain marked for ever, because \( t_4 \) immediately puts one token on it when it is fired. Therefore, by means of this place,
when the guard condition is false, we immediately finish the execution of the workunit, marking \( p_{wok} \), which is its correct
exit place. Otherwise, when the guard condition is false and \( \neg block \) is not marked, no transition can be fired but \( t_3 \), i.e. the
PTPN is blocked until the guard condition changes. Transitions \( t_1 \) to \( t_6 \) are assigned a priority level \( M \), where \( M \) is a value
greater than any other priority level in the WS-CDL document, the reason being that guard and repetition conditions must be evaluated before any other action, to prevent a priority interaction inside the workunit from being executed after another with lower priority.

It can also happen that the workunit does not have a guard condition, in which case, the translation would be easier, all the upper part of Fig. 7a (over Na) would be removed and \( p_{\text{win}} = p_{\text{ain}} \).

Let us note that workunits can have a repetition condition, which has been considered by transitions \( t_5 \) and \( t_6 \). The firing of \( t_5 \) corresponds to the evaluation of this condition to true, and therefore the activity inside the workunit is restarted. On the other hand, when \( t_6 \) is fired, it is assumed that the repetition condition is false, and therefore the workunit finishes, by marking \( p_{\text{work}} \). The erroneous exit place of the workunit is that of Na, i.e. \( p_{\text{wer}} = p_{\text{aer}} \). Finally, if the repetition condition is omitted we would take \( p_{\text{work}} = p_{\text{aok}} \) and \( t_5, t_6 \) would be removed.

In Fig. 7b we can see the translation for the workunits that are used to delay the execution; the values \( x, y \) in the time interval of transition \( t_1 \) are obtained from the XPath expression in the guard condition of the workunit. Remember that only the time variable can be used in these guards, and only for comparing its value with both a lower and an upper bound. Actually, as it is used to delay the execution, it must have been initialized (to zero) immediately before the workunit where it is used.

### 6.3. Ordering structures

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

- **Sequence:** A sequence of two activities (with PTPNs \( N_a \) and \( N_b \), respectively) is translated in an easy way (Fig. 8a), by just collapsing in a single place the correct exit place of \( N_a \) with the initial place of \( N_b \), and also collapsing the erroneous exit place of \( N_a \) and \( N_b \), which is \( p_{\text{ser}} \) (the erroneous exit place of the sequence), i.e. in case of failure in \( N_a \) or \( N_b \) the complete sequence terminates abnormally by marking \( p_{\text{ser}} \).

- **Parallel:** we now consider two parallel activities, with PTPNs \( N_a, N_b \). The translation for the parallel ordering structure is that shown in Fig. 8b. Transition \( t_1 \) forks both parallel activities, while \( t_2 \) joins both once they have (correctly) finished. If one of them (or both) finishes abnormally, \( p_{\text{per}} \) will become marked. Observe that transitions \( t_1 \) to \( t_5 \) are assigned priority \( M \) (a value greater that any priority level in the WS-CDL document), in order to allow any initial transition of \( N_a \) or \( N_b \) to be executed before some other external one with lower priority that could be enabled at the same time.

- **Choice:** we now have two activities, and only one of them can be finally executed. However, in the case of (general) guarded workunits as an alternative, we must discard them when their guards are evaluated to false. Moreover, workunits can be
repetitive (see Fig. 7a), so the translation must consider this aspect, namely, once a workunit has been selected, it is the only one that can be executed, i.e., the choice has been resolved and only the selected alternative can proceed. Thus, we provide a translation for the case in which no general guarded workunit appears as an alternative in a choice\(^4\) (Fig. 8c), and a translation for this specific case (Fig. 9). The figure here corresponds to the case in which only one of the alternatives is a general repetitive guarded workunit. From this case it is straightforward to obtain the translation for the other possible cases, as well as to extend this translation to a general choice, with \(n\) alternatives.

For the first case (Fig. 8c), notice that \(t_1\) marks both initial places of \(N_a\) and \(N_b\), as well as place \(q\), which is used to avoid the execution of any transition in a net when the other one has started. For that purpose, we connect this place with all the postcondition transitions of the initial places of both PTPNs. When the selected PTPN finishes, either its correct or its erroneous exit place is marked; transitions \(t_2\) to \(t_5\) are then used to mark the correct or the erroneous exit place of the choice activity, but notice that they remove the token that was still in the initial place of the other PTPN.

When one of the alternatives of the choice is a general repetitive guarded workunit (Fig. 9) the translation requires more effort, because we need to replicate the initial places and transitions of \(N_a\) in order to repeat the activity when a repetition condition has been indicated. Therefore, \(N_{a}^{'} = (P_{a}^{'}, T_{a}^{'}, F_{a}^{'}, \alpha_{a}^{'}, \beta_{a}^{'}, \pi_{a}^{'})\) is obtained as follows:

\(^4\) Directly, or as first activity in a sequence.
be selected for execution. Once the activity finishes, we may have either activity completes normally, transitions actually, taking into account that this construct will be very unusual, we could have even banned it in the structure and for the choice alternatives. Hence, we have decided to maintain the PTPN semantics as indicated in this case would be quite complex, requiring a distinction of cases both for the activities inside the parallel ordering to execute the parallel activities when one of their activities is selected for execution. However, the translation in immediately, because of the immediate transition that is used to fork the parallel activities. Another criterion could be construct as an alternative in a choice. According to Fig. 8b and c our PTPN semantics would resolve the choice immediately, because of the immediate transition that is used to fork the parallel activities. Another criterion could be to execute the parallel activities when one of their activities is selected for execution. However, the translation in this case would be quite complex, requiring a distinction of cases both for the activities inside the parallel ordering structure and for the choice alternatives. Hence, we have decided to maintain the PTPN semantics as indicated in Fig. 8c; actually, taking into account that this construct will be very unusual, we could have even banned it in the translation.

According to Fig. 9 we can see that \( t_0 \) marks both initial places \( p_{w_{ab}} \) and \( p_{b_{ab}} \), as well as \( q \), as in Fig. 8c. However, in this case \( q \) is connected with the initial transitions of \( N_b \), and thus the activity inside the workunit can only start if the guard has been evaluated to true. In this case no block place is needed, because when the guard is false this alternative cannot be selected for execution. Once the activity finishes, we may have either \( p_{d_{ab}} \) or \( p_{a_{er}} \) marked. In the first case, when that activity completes normally, transitions \( t_4 \) and \( t_5 \) are used to check the repetition condition. Thus, if the workunit must be restarted, \( p_{d_{ab}} \) is marked, whereby the activity inside the workunit is restarted.

Transitions \( r_1, r_2 \) and \( r_3 \) capture the correct termination of \( N_b \), by marking \( p_{c_{ab}} \). Notice that when \( p_{d_{ab}} \) is marked, one of the three places \( p_{w_{ab}}, p_{a_{in}} \), or \( \neg g \) must be marked, and thus, these transitions are included to remove that token. For the same reason we have included transitions \( r_1', r_2' \) and \( r_3' \), in this case \( N_b \) has finished abnormally, so \( p_{c_{er}} \) becomes marked with their firing. Finally, transition \( t_6 \) captures the abnormal termination of the workunit, so it marks \( p_{c_{er}} \) too.

It is important to observe that the translation requires that no alternative of a choice can be a choice too. However, these inner choices are not semantically different from the outer ones, i.e. we can put all of these inner alternatives on the outer level, thus obtaining a single choice with no inner choice as an alternative. This can be done by means of a pre-processing of the WS-CDL document, in order to flatten the specification so that there is no choice structure inside a choice (see Fig. 10).

Another case that requires some explanation is that of a parallel activity as alternative. The WS-CDL document description in [30] has nothing to say about this specific case, probably because it would be rare to find a parallel construct as an alternative in a choice. According to Fig. 8b and c our PTPN semantics would resolve the choice immediately, because of the immediate transition that is used to fork the parallel activities. Another criterion could be to execute the parallel activities when one of their activities is selected for execution. However, the translation in this case would be quite complex, requiring a distinction of cases both for the activities inside the parallel ordering structure and for the choice alternatives. Hence, we have decided to maintain the PTPN semantics as indicated in Fig. 8c; actually, taking into account that this construct will be very unusual, we could have even banned it in the translation.

\[ P_a = P_a \cup \{ p_{a_{in}} \} \], where \( p_{a_{in}} \) is the initial place of \( N_a \)
\[ T_a = T_a \cup \{ t_a \mid t_a \in p_{a_{in}}^* \} \]
\[ F_a = F_a \cup \{ (p_{a_{in}}, t_a') \mid t_a' \in T_a \setminus T_a \} \cup \{ (t_a', p) \mid t_a' \in T_a \setminus T_a, (t_a, p) \in F_a \} \]
\[ \alpha_a(t_a') = \alpha_a(t_a) \text{ for } t_a' \in T_a \setminus T_a, \text{ and } \alpha_a(t_a) = \alpha_a \]
\[ \beta_a(t_a') = \beta_a(t_a) \text{ for } t_a' \in T_a \setminus T_a, \text{ and } \beta_a(t_a) = \beta_a \]
\[ \pi_a(t_a') = \pi_a(t_a) \text{ for } t_a' \in T_a \setminus T_a, \text{ and } \pi_a(t_a) = \pi_a \]
6.4. Exception blocks

Choreographies may have one exception block, which consists of some (possibly guarded) workunits, but only one of them can be finally executed (the first one whose guard evaluates to true). For simplicity we assume that at most only one non-guarded workunit can be defined in the exception block (the so called default exception workunit). Furthermore, exception workunits cannot be repetitive and their block attribute must be false. The translation of the default exception workunit is therefore that of the activity inside it. When the exception block is executed, the choreography terminates abnormally, even if the default exception workunit has terminated its own execution correctly (see Fig. 4).

6.5. Choreographies composition

We may have a hierarchy of choreographies, one of them being the root choreography. The perform activity enables a choreography to specify that another choreography is performed as an enclosed choreography at the point where the perform activity appears.

We allow at most one instance of a choreography to be performed in the WS-CDL document, and we assume that the performing choreography waits for the performed choreography to complete before the perform activity does so, which is the default semantics of the perform activity in WS-CDL. The translation of the perform activity is that shown in Fig. 11a and b, where the PTPN enclosed in a dashed-rectangle is the PTPN corresponding to the performed choreography, which has a new place, called $p_{cok}'$, which will become marked once the choreography has been performed successfully, and that will be used in the finalize activity to enable the (potential) execution of its finalizer block. The initial place of the perform activity ($P_{in}$) is connected by means of transition $t_1$ with the initial place of the performed choreography, in order to activate it. Furthermore, when the performed choreography has an exception block (Fig. 11a), once this exception block has been executed ($p_{cer}$ is marked), the performing choreography continues its execution normally, for this reason we connect the (unique) final place of the exception block ($p_{cer}$) with the correct exit place of the perform activity.

On the other hand, when the performed choreography does not have an exception block (Fig. 11b), in the event of failure on it, the exception is propagated to the performing choreography; thus, we connect the erroneous exit place of the performed choreography to the erroneous exit place of the perform activity.
6.6. Finalizer blocks

Choreographies may have one or more finalizer blocks,5 but only one of them can be finally executed, so we will assume that there is only one finalizer block. A finalizer block consists of one activity, so the translation of this block is that of the enclosed activity.

A finalize activity can be used to execute the finalizer block of a successfully completed immediately enclosed choreography. The actions introduced in the finalizer blocks are used to confirm, cancel or modify the effects of the performed choreography.

The translation of the finalize activity is depicted in Fig. 12. In this figure, \( N_c \) is the PTPN corresponding to the enclosed choreography whose finalizer block is executed, and \( N_{fz} \) is the PTPN corresponding to its finalizer block. If the enclosed choreography has been performed successfully, \( p'_{cok} \) will be marked, and \( t_1, t_2 \) will be enabled, but since both transitions are black and immediate,6 and \( t_2 \) has a greater priority, this transition will be immediately fired in order to start the choreography finalizer block.

A finalizer block invoked for a choreography that has not been performed successfully has no effect. This is captured by means of \( t_1 \), which can be fired when \( p'_{cok} \) is unmarked, and immediately produces one token over \( p_{fok} \). When the finalizer block activity fails, the finalize activity terminates abnormally, \( p_{ferr} \) becoming marked.

6.7. Safeness

In this section we prove that the PTPNs that we obtain by applying the defined translation are 1-safe. We also prove that only one of the two exit places of the obtained PTPN will finally be marked.

**Proposition 1.** Let \( N = (P, T, F, \alpha, \beta, \pi) \) be a PTPN obtained from a WS-CDL document by applying the translation. As initial marking of \( N \) we consider one token at the initial place, and one token on each \( \neg \)-block place for which the corresponding block

---

5 Except for the root choreography, which cannot have any finalizer blocks.
6 Their time interval is \([0, 0]\).
Fig. 14. WS-CDL description of the airline reservation system.
attribute on the WS-CDL document was false (see Fig. 7a). Then, for any reachable state 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, as well as the block places, which keep their initial marking, and the \(p_{cok}'\) places of the performed choreographies for which no finalize activity has been executed, which would also have one token.

**Proof:** By structural induction, the base cases are those corresponding to the basic activities (interactions, assign, silent and noaction), which are all immediate. For the general case we must consider the different structured activities of WS-CDL and the composition of choreographies, and for each one of them, we must prove that the construction generates a 1-safe PTPN, assuming as an induction hypothesis that all the argument activities of the considered structured activity are 1-safe, taking as the initial marking for them the one obtained by marking their initial place and their block places whose corresponding block attribute in the WS-CDL was false.

We distinguish the following cases:

- **Choreographies:** These are translated as indicated in Fig. 4, but the hierarchical relationship between the choreographies is established according to the perform activities (Fig. 11) and the finalize activities (Fig. 12).

As we are assuming that all the PTPNs of the argument activities are 1-safe, it follows from these pictures that the resulting PTPN will be 1-safe too, but notice that when a finalizer block is not executed for a performed choreography, its place \(p_{cok}'\) will keep one token forever. Nevertheless, this does not cause any problems as we are assuming that choreographies are executed at most once.
• Workunits: We now have some cases, Fig. 7a being the most general one, corresponding to a guarded and repetitive workunit, whilst Fig. 7b corresponds to a delayed workunit. Other cases\(^7\) are obtained as modifications of Fig. 7a. For all of these cases the proof is again a simple application of the induction hypothesis for the activities inside the workunit.

• Ordering structures: The sequence and parallel constructs are immediate, taking into account the PTPNs of Fig. 8a and b. For the choice construct a distinction of cases must be made, taking into account the constructs that appear at the first level as alternatives of the choice. All of these cases are very similar; they use a place \(q\) to inhibit the firing of the initial transitions in the alternatives when one alternative has started its activities, and when the executed alternative ends the tokens that were still at some places of the other alternatives are removed. Thus, using the induction hypothesis and the reachability graphs of the PTPNs corresponding to each case it can be easily checked that the obtained PTPNs are 1-safe and that only one of the two exit places will finally be marked. □

7. Case studies

In this section, we present two examples that allow us to illustrate our methodology. The first one is an airline ticket reservation system, which illustrates how time requirements and priorities can be introduced in the interactions among the parties in a choreography.

Our second example is a supplier system, in which we consider a hierarchy of choreographies, and a finalizer block in one of them. In this case, we have a supplier that attends to client requests according to a scheme of priorities, established depending on the client types.

7.1. An airline ticket reservation system

We consider an airline ticket reservation system, which consists of three participants: Traveler, Travel Agent and Airline Reservation System (ARS).

The ARS receives requests from travelers and travel agents to reserve seats. However, in case of conflict the traveler’s requests are served first. The reservation system works as follows: a trip request for a specific date is received by the ARS, which replies with a list of available flights on that date. The user will then choose the best option and will order a reservation on a specific flight. Reservations are only valid for a period of two days, which means that if a final confirmation has not been received within that period, the reservation is cancelled, and the seats are released.

7.1.1. Analysis phase

The goal model diagram of this example is shown in Fig. 13. The main goal is Correct System, which consists of two subgoals linked by an AND-node: Customer Service and Proper Ending. The first goal is also refined into three subgoals linked by AND-nodes: Information request (travelers and travel agents can make their trip requests), Reservation Priority (in case of conflict travelers are priority) and Booking and Payment (after the reservation the clients must confirm and pay the ticket). The second goal is refined into two subgoals: Correct Sequence (request of information, reservation and confirmation must be performed in that order) and Reservation Time Restriction (reservations prescribe in two days).

\(7\) Non-guarded or non-repetitive workunit.
Fig. 14 contains the relevant parts of the WS-CDL document describing this choreography. From this document we obtain the PTPN depicted in Fig. 15. In this PTPN we have removed some places and transitions that can never be reached, e.g., the isolated places obtained for untimed interactions. In this figure, transition $t_2$ represents the request for information by a traveler, whereas $t_3$ is the request for information by a travel agent. Additionally, $t_7$ represents a traveler’s reservation, and $t_8$ a travel agent’s reservation. Finally, $t_{14}$ and $t_{15}$ are the bookings made by the traveler and the travel agent respectively.

7.1.3. Validation and verification phase

In order to analyse this system we have generated the timed reachability graph of the PTPN of Fig. 15. By means of this graph we have checked the goals gathered in the KAOS goal model. The first goal, InformationRequest, is simple to prove, both the traveler and the travel agent can initially put in a request for flight information, which corresponds to the possible firing of either $t_2$ or $t_3$ starting from the initial marking at any later instant. The second goal, Reservation Priority, is captured by the reachable marking at which both $t_7$ and $t_8$ are potentially fireable, and both have been requested for firing. In that case, $t_7$ (traveler's reservation) will be fired. The third and fifth goals, Booking and Payment and Reservation Time Restriction are captured by the sequential firing of transitions $t_{10}, t_{14}$ (travelers) and $t_{11}, t_{15}$ (travel agents), and the time intervals associated with $t_{10}$ and $t_{11}$. Actually, when a confirmation has not been received in time, a time-out raises ($t_9$ or $t_{12}$ is fired), and the choreography terminates abnormally (the place er becomes marked).

Concerning the fourth goal, Correct Sequence, from the timed reachability graph we can see that a traveler can make a request for information, and afterwards a travel agent that has not made a request for information can make a reservation, so this can be considered as a design mistake, which can be fixed by changing the WS-DCL document in the appropriate way. In particular, the initial choice can be changed by a parallel activity, which would allow both the travelers and the travel agents to make their complete sequence of operation. Then, once the WS-CDL document is rewritten, the new PTPN is obtained and the identified KAOS goals are checked again.
7.2. A supplier system

We have clients that buy perishable products by using the Internet. They connect to a supplier server that offers different products, and then, once they have selected the product they want to buy, they send to the supplier server a purchase order that includes the type of product sought after and after that, the client and the supplier interact to agree on the payment method.

The clients are divided into two types, normal and vip, according to the contract type they have established with the supplier. Vip clients receive preferential treatment, i.e. in case of conflict the supplier server first attends the requests from vip clients.

When the supplier server receives the client purchase order, and once both have agreed on the payment method, it contacts the shipper in order to arrange the delivery of the product, which must be received by the client within 36 h (otherwise the product rots and the purchase is cancelled). Finally, once the product has been delivered, the shipper informs the supplier.

7.2.1. Analysis phase

In Fig. 16 the goal model diagram of this system is depicted. The root goal Correct System is decomposed into two subgoals by an And-refinement, which means that both goals, Supplier Service and Dispatching Service, must be fulfilled.

The first goal, Supplier Service is in turn refined into two subgoals: Client Priority (in case of conflict vip clients are served first) and Purchase Order and Payment (once the client has sent the purchase order both the client and the supplier negotiate the payment information), and the second goal, Dispatching service, is refined into three subgoals: Correct Sequence (the supplier sends to the shipper the information about the product to be delivered after the negotiation of the payment method), Delivery On Time (the shipper must deliver the product in 36 h) and Confirmation (the shipper informs the supplier that the product has been delivered).
7.2.2. Design phase

This system has been designed by considering a root choreography (Fig. 17), with two nested choreographies, one for the purchase order (Fig. 18), and another for the shipper (Fig. 19). The root choreography sequentially invokes these choreographies, and it has a finalize activity for the shipper choreography, which is used to inform the supplier that the product has been delivered to the client.

From this WS-CDL document we can obtain the corresponding PTPN, by applying the translation defined in Section 6. This PTPN is depicted in Fig. 20, where we have separated in dashed-rectangles the PTPNs of the two nested choreographies. The initial place of the root choreography is root_in, whereas root_ok and root_er are its correct and erroneous exit places, respectively. The finalizer block of the Dispatcher choreography has also been graphically separated by means of a dashed line. We can see, for instance, that transitions t_{21} and t_{22} correspond to the finalize activity of the root choreography, and t_1, t_9 are associated with both perform activities, for the execution of the two nested choreographies. Another point of interest is the translation of the choices that appear in both nested choreographies, specifically that of the shipper choreography, where we find interactions with time-outs as alternatives, and we therefore need to consider the corresponding erroneous exit place for this choreography (dp_{er}).

7.2.3. Validation and verification phase

We have generated the timed reachability graph of the PTPN of Fig. 20. By means of this graph we have checked the goals gathered in the KAOS model. The first subgoal, Client Priority, is captured by the conflict between the transitions t_3 and t_4, which have priorities 1 and 3, respectively. Thus, when both are simultaneously fireable, and both have been requested for firing, t_4 will finally be fired, which corresponds to the purchase order of a vip client. The second subgoal, Purchase Order and Payment, is captured by the sequential firing of t_5, t_6 (normal clients), or t_4, t_6 (vip clients). The next subgoal Correct Sequence corresponds to the firing of t_{10} after either t_5 or t_6, which is immediate. The subgoal Delivery on Time is captured by transitions t_{13} (normal clients) and t_{14} (vip clients), and their time-out transitions, t_{12}, t_{15}, respectively, which would fire in the event of the shipper not having delivered the product on time. Finally, the Confirmation subgoal corresponds to the sequence of transitions t_{20}, t_{22} and t_{25}, which are executable once dp_{ok} becomes marked. Notice that t_{21} can never be executed in this case, because the finalize activity is executed once the shipper choreography has been successfully performed.
8. Conclusions

In this paper we have presented a methodology, CWS-PNs, for the analysis, design and model validation of Web Services systems. For the analysis phase we use the KAOS methodology, which allows us to obtain a goal diagram that is later used for validation/verification purposes. The design phase is accomplished by using WS-CDL, a W3C proposal for describing Web Services choreographies. We have also defined a PTPN semantics for a relevant subset of WS-CDL, in which the composition of choreographies, and the main activities of them have been considered (basic and structured activities). Thus, one of the main contributions of this paper is to show the use of formal methods within the classical perspective of software development. Our methodology starts by establishing the KAOS goal model, and then, a WS-CDL document is produced as first product of the design phase. In the classical software development cycle, the implementation phase comes after the design phase. However, we have arranged for another stage to come between them, namely, the model validation and verification phase, where the elaborated designs are checked with the properties gathered in the analysis, by using techniques based on formal models. Thus, if the design does not satisfy some property, we can modify it until all the properties are satisfied. Notice that, we have not introduced any comments about the implementation, but web services are supported by a widely range of software languages, such as JAVA, C++, PHP, C#, etcetera, and the WS-CDL can be deployed to them by using tools like pi4soa (http://pi4soa.sourceforge.net/)

Another contribution of the paper is the introduction of priorities in WS-CDL. The introduction of priorities allows the parties of a Web Composition to favour some interactions, which can be useful in many situations, for instance, to distinguish clients or items, as we have seen in the case studies. Time restrictions have also been considered in this paper, both in interactions (time-outs) and in workunits, to delay the execution. We have defined a translation of WS-CDL into a timed-prioritized model of Petri Nets (PTPNs), which allows us to simulate and verify the system. The obtained PTPNs are 1-safe, which means that only one token can be at any place in any reachable marking. Therefore, they can be easily simulated and the verification of properties is also possible by using some tool supporting the PTPN model.

The official semantics of WS-CDL [30] is defined in a textual manner. Thus, another important advantage of the PTPN semantics is that it can be used as an alternative to the textual document 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 are not completely described, and a formalization also serves to detect these deficiencies.

Another important consequence of the translation is that it can be used to obtain the particular behaviour of each party. Notice that transitions of the obtained PTPN are labelled with the RoleTypes involved in their execution. We can thus extract the PTPNs of each RoleType, and these PTPNs can then be used as a high-level design for them. We therefore obtain as a subproduct a first design for the different RoleTypes, which can be progressively detailed by refinements.

As future work, we plan to extend the translation supporting a richer subset of WS-CDL. For instance, the inclusion of variables and the extension of some WS-CDL constructs. Variables are used in WS-CDL in many ways, for instance, as workunit guards. The translation presented in this paper for guarded workunits only considers a choice between two transitions, which represent the guard evaluation, and the same occurs for repetition conditions, but we have abstracted from the variables that are used for that. Therefore, using a Petri net model that supports variables (High-Level Petri nets) we will be able to define a translation capturing the WS-CDL semantics more appropriately. Another aspect that can be improved is that of abnormal terminations, in the presented translation we have only considered the case of a time-out that has expired, but there are other situations (mainly related with variables) that cause abnormal termination that could be considered in an extended version of this work, but they require once again a High-Level Petri net model.

The work can also be extended by considering a broader spectrum of WS-CDL constructs. For instance, we have restricted the perform activity to its blocking form, but in WS-CDL it can also be used in a non-blocking form, so that we plan to extend our work to take in this possibility.

References


