Specification and Verification of Views over Composite Web Services Using High Level Petri-Nets

Khouloud Boukadi[1], Chirine Ghedira[1], Zakaria Maamar[2], and Hanifa Boucheneb[3]

[1]LIRIS Laboratory, Claude Bernard Lyon 1 University, Lyon, France
{khouloud.boukadi, chirine.ghedira}@univ-lyon1.fr
[2]College of Information Systems, Zayed University, Dubai, U.A.E
zakaria.maamar@zu.ac.ae
[3]Ecole Polytechnique de Montreal, Montreal, Canada
hanifa.boucheneb@polymtl.ca

Abstract

This paper presents a high level Petri-Net approach for specifying and verifying views over composite Web service. High level Petri-Nets have the capacity of formally modeling and verifying complex systems. A view is mainly used for tracking purposes as it permits representing a contextual snapshot of a composite Web service specification. The use of the proposed high level Petri-Net approach is illustrated with a running example that shows how Web services composition satisfies users' needs. A proof-of-concept of this approach is also presented in the paper.

Keywords. Context, Petri-Net, View, Web service.

1 Introduction

For the World Wide Web Consortium (W3C), a Web service *is a software application identified by a URI, whose interfaces and binding are capable of being defined, described, and discovered by XML artifacts and supports direct interactions with other software applications using XML-based messages via Internet-based applications*. Web services have given Web applications a new shape, from content display to service supplier. The capacity of defining composite Web services is an advantage that currently backs the widespread use of Web services. Businesses and academia have shown a significant interest in Web services composition [1]. Several composition standards exist such as WSFL [2], Xlang [3], BPEL [4], BPML [5] , etc.

Despite the large body of research on Web services, much work still needs to be done to tie informal methods, e.g., Petri-Nets, with specification languages for composite Web services. Few initiatives have looked into the use of Petri-Nets in Web services including [6] and [7]. Indeed, there is no guarantee that the specification of a composite Web service is free of errors. Conflicting actions like concurrent accept or reject and deadlocks may occur during the specification execution. Fixing errors at run-time is time-consuming and requires another round of low-level programming, which could be expensive, as well as error-prone. An attractive solution would consist of allowing developers to detect and fix issues prior to any Web services deployment and to formally verify the business processes underlying composite Web services against some desired properties [6].

Composition does not only make Web services bind to each other, but emphasizes the cornerstone of

handling users' preferences and constraints as part of the process of meeting personalization requirements. Personalization is tightly related to the features of the environment in which Web services will operate after triggering. These features can be related to users (e.g., state, location), computing resources (e.g., fixed device, mobile device), time periods (e.g., in the afternoon, in the morning), physical places (e.g., mall, cafeteria), etc. Sensing, gathering, and refining the features and changes in an environment contribute towards the definition of what is known as context. Context is *the information that characterizes the interactions between humans, applications, and the surrounding environment* [8]. Embedding Web services with context-awareness mechanisms has several advantages as reported in [9]. To be aware of which part of the specification of the composite Web service has to be adjusted because of changes in the user environment, an assessment of what-was-previously-expected and what-is-effectively-happening is deemed appropriate. This specific part of the composite Web service specification is referred to as view [10]. A view is *a dynamic snapshot over the specification of a composite Web service according to a certain context* [10] .

In this paper, we aim at discussing the value-added of Petri-Nets to the specification of firstly, the composite Web services and secondly, the views that run over those ones. We emphasize the use of high level Petri-Nets, particularly Colored-Petri-Nets (CPN) and Hierarchical-Colored-Petri-Nets (HCPN) [11]. CPN and HCPN have the capacity to specify and analyze concurrent systems [12]. Our contributions in this paper are as follows: a definition of a composite Web service using high level Petri-Nets, an approach for checking the correctness of a composite Web service, a specification of a view based on high level Petri-Nets, and finally, automatic mechanisms for extracting and showing up views over composite Web services.

The rest of this paper is organized as follows. Section 2 presents a motivating scenario, reviews some advantages of Petri-Nets, and suggests a list of related projects. Section 3 discusses the use of Petri-Nets in modeling Web services and composite Web services, and talks about modeling component and composite Web services using high level Petri-Nets. Section 4 describes the concept of view as a means for tracking the execution of a composite Web service specification. Section 5 presents the prototype that was developed as a proof-of-concept of our use of Petri-Nets in Web services. Concluding remarks are drawn in Section 6.

2 Background

2.1 Motivating scenario

Our motivation scenario concerns Anatole, a 60-years old patient who is treated in the Cardio-Thoracic and Vascular department of "Edouard Herriot" Hospital in Lyon. Anatole has a Portable ECG Monitor (PEM), which is used to detect and manage any cardiac event. An electrocardiogram (ECG) is a test that records the heart's electrical activity. When Anatole feels a chest pain, he turns on the PEM so his ECG is recorded. The PEM starts with a serial analysis of this record and compares it with the referenced ECG. The PEM can suspect any cardiac problems and send an alert to a call center, if needed. The alert triggers a Web service

whose role is to find a first-aid medical-center close from Anatole's current location. Processing both the recorded and referenced ECG, the selected medical center identifies two types of alarm: severe or minor.

In case of a minor alarm, **LookforDoctor** and **TreatmentTransmission** Web services are triggered. When a doctor is assigned to Anatole, he gets access to his medical records and checks the referenced and recorded ECG. Afterwards, he diagnoses the case and prescribes an adequate treatment for Anatole. **TreatmentTransmission** Web service takes care of notifying the treatment, as an SMS message, to Anatole's mobile phone. The language of the message is set according to Anatole's preferred-language like French, English, etc. Before closing Anatole's case, additional operations update his records. In case of a severe alarm, **LookforEmergency** Web service is concurrently triggered with two other separate Web services that upload Anatole's medical records and identify Anatole's location, respectively. Finally, **ContactMobileCare** Web service is activated in case an ambulance needs to be dispatched to Anatole.

2.2 Rationale of colored Petri-Nets

Jensen formulates CPN as a formally founded graphically-oriented modeling language [11]. CPN have got their name because they use different colors to be associated with tokens, which carry data values. This is in contrast to low level Petri-Nets' tokens, which by default are black. On the one hand, Petri-Nets provide the necessary mechanisms for specifying synchronization of concurrent processes. On the other hand, any programming language provides the primitives that are needed for defining and manipulating data types. Compared to CPN, HCPN includes additional features such as substitution transitions and fusion places. The underlying idea in HCPN is to use a number of small CPN in order to develop a complete HCPN. These CPN are called pages (or sub-pages) and are related to each other in a well defined way.

Mapping CPN and HCPN concepts into Web services and composite Web services is to a certain extent straightforward. First, a Web service behavior is basically a partially ordered set of operations. Therefore, it is possible to represent it with a Petri-Net. Web service's operations are modeled by transitions and the states of the Web service are modeled by places [12]. Moreover, the use of colored tokens permits modeling contexts of Web services and users by specifying places used to model these contexts. Moreover, the hierarchy concept of the HCPN shows the components of a composite Web service at a higher level with no mention to their internal details. This is really useful for running views over composite Web services. Indeed, adjusting a composite Web service with the changes in user and Web service's contexts does not require to highlight the detailed descriptions of all the Web services that participate in this composition. The use of HCPN for modeling Web services composition is detailed in the next section.

2.3 Related work

Benatallah et al. propose a Petri-Net-based algebra as a means for guaranteeing the reliability of the business

processes that underpin a composite Web service composition [13]. This algebra is enough expressive to capture the semantics of complex Web service combinations. In this study, context is just ignored, which does not permit capturing the changes in a Web services composition process. Yang et al. introduce an approach whose objective is to verify and analyze composition specification of Web Services once these specifications get translated into a Colored Petri-Net [6],[7]. The obtained colored Petri-Net can be simulated using specific colored Petri-Net tools. In Yang et al.'s works, they suggest to model processes in BPEL as colored Petri nets. However, the transformation process is ambiguous and no formal definition of how to translate a BPEL specification into a colored Petri-Net is given. Xiaochuan et al. propose a model of a simplified Travel Reservation system based on Web services [14]. This system captures both conversation protocols and composition specification. The model uses a colored Petri-Net's properties such as dead marking, boundness, and reachability, during verification. In [15], Bing addresses the shortfalls of Xiaochuan et al.'s work like incomplete conversation between Web services, removal of some major interactions within Web services, and modeling of unnecessary components that make the graphical representation complex.

All these proposals mainly focus on Web services composition modeling with no-emphasis on contexts of Web services and users. It should also be noted that colored Petri-Nets are suitable for validating Web services composition. The transformation process for example from a BPEL specification towards a colored Petri-Net is still ambiguous and a formal definition would be highly appreciated. In order to react in a proper way to the detected changes in user and Web service environment, context needs to be handled during the development of specifications of Web services.

3 Modeling Web services composition using high level Petri-Nets

In this section, we define a Web service and a composite Web service using CPN and HCPN, respectively. To this purpose, we comply with Jensen's work [11]. A composite Web service is defined by an HCPN to be called as Composition Net (CN). Moreover, each component Web service in the CN has a page modeled by a CPN to be called as Service Colored Net (SCN).

3.1 Definition of the service colored net

A Web service is a SCN that is defined as follows: **SCN=<Σ,P,T,L,A,N,C,E,G>**.

- **Σ** is a finite set of types also called color sets.
- **P** is a finite set of places that model the state of a system. A Web service's states as proposed in [13] include: not-instantiated, ready, running, suspended, and completed. In this paper, a Web service's state consists of distributing a data value, i.e., token, on the SCN's places. Two types of places exist:
    - **Message Places** (**MP)** contain messages exchanged between component Web services.
    - **Context Places (CP)** containing the execution context of Web services.

Formally, this is represented as follows: **P: (MP∪CP) and (MP∩CP)= ∅**

- **T** is a finite set of transitions. Each operation in a Web service is captured by a CPN transition. We can distinguish two types of transitions:
    - $\mathbf{T_{gu}}$ is a finite set of transitions with guard condition.
    - $\mathbf{T}_{\overline{gu}}$ is a finite set of transitions without guard condition.

  Formally, this is represented as follows: $\mathbf{T: (T_{gu} \cup T_{\overline{gu}}) \text{ and } (T_{gu} \cap T_{\overline{gu}}) = \emptyset.}$

- **A** is a set of directed arcs. An arc connects a place to a transition and *vice-versa*. In fact, an arc represents a causality relation between places and transitions. Formally, this is represented as follows: $\mathbf{A \subseteq (P \times T) \cup (T \times P)}$
- **L** is a labeling function. We suppose that each operation in a Web service has a label. Formally, this is represented as follows: $\mathbf{\forall\, t \in T\ L: t \rightarrow L(t)}$
- **N** is function that links each arc going from a place to a transition and *vice-versa*.
- **C** is a color function that assigns a unique color to each place p. The color of a place is denoted as C(p). Therefore, each token in a place p must have a color, i.e., data value, from C (p). Formally, this is represented as follows: $\mathbf{C: P \rightarrow \sum \Leftrightarrow \forall\, p \in P, \exists\ C\ (p)\ C\ (p) \subseteq \Sigma}$
- E is a function that describes arcs using a set of variables. These variables determine the token's variables (i.e., a token has a set of variables) that are either consumed or produced during operation. Formally, this is represented as follows: $\mathbf{\forall a \in A, Type(var(E(a)) = C(p(a)) \text{ and } Type(var(E(a))) \subseteq \Sigma}$ where: **Type(var(E(a))** is a function that determines the types of the variables in an arc. In this formula, the first part indicates that the types of the arc's variables must be compatible with the colors set of the starting place (arc connecting place-transition) or the colors set of the arrival place (arc connecting transition-place). The second indicates that the token's types must belong to the colors set of the CPN.
- **G** is a guard function that tests the logical conditions in a transition.

The navigation inside a Service Colored Net of a composite Web service is based on two types of rules. It will be shown later that these rules contribute towards the formal specification of a view.

1. **Firing rule for a transition with guard condition**: In order to get over a transition with guard condition, we must consider the types of places, whether message or context, and thus the conditions of these places. In case of a message place, four conditions should be verified before a transition can be passed. The first condition deals with the color of the place, which must be expected by the color sets necessary for passing over the transition. We assume that each transition has a color set, which is the sum of all the color sets of the different connecting places. The second condition verifies that the

variable set of the arc has the same type as the place connected with this arc. In addition, the values of the variables on an arc must match the expected data types such as integer. The last condition checks if the guard condition returns true assuming that every type of variables of a guard belong to the color sets of the service colored net. We use Is-enabled (T_{gu}) as the function that checks the four conditions for each message place that is connected to transition T_{gu}. In case of a context place, only the three first conditions must be verified. The following algorithm summarizes the different conditions already explained.

```
For each t_gu ∈ T_gu
   For each p ∈ P / ∃ a ∈ A, N (a) ∈ p ∪t_gu
      If p ∈ MP
        Is-enabled (t_gu) iff
            C(p) ⊆ C(t_gu)                                   //1
            && Type (Var(E(a))) = C(p(a))                    //2
            && ∀ v ∈ Var(E(a)): Binding (v) ∈ Type(v)        //3
            && Type (G (t_gu)) =True ∧Type (Var (G (t_gu)) ⊆ Σ   //4
      Else if p∈ CP
        Is-enabled (t_gu) iff
            C(p) ⊆ C(t_gu)                                   //1
            && Type (Var(E(a))) = C(p(a))                    //2
            && ∀ v ∈ Var(E(a)): Binding (v) ∈ Type(v)        //3
      End if;
   End for ;
End for ;
```

2. **Firing rule for a transition without guard condition:** this transition is independent of the type of its connected places. Is-enabled ($\mathbf{T}_{\overline{gu}}$) function verifies only the first three conditions.

3.2 Definition of the composition net

A composite Web service is a HCPN that is defined as follows: **CN**=<**S, ST, SA, PP, PT, PA, FP**> where:

- **S** is a set of pages that represent the atomic Web services. Each page s∈S is a SCN=<∑s,Ps,Ts,As,Ns,Cs,Es,Gs>.
- **ST** is a set of substitution transitions. A substitution transition identifies a Web service without any internal details on how it is performed. Similarly to transitions in a service colored net, substitution transitions in CN are with or without guard transition.
- **SA** is a function that assigns a Web service to a composite Web service. Indeed, each ST corresponds to SCN (SA: ST→SCN). In contrast with Jensen's substitution transition firing rules, we assume that firing a substitution transition depends on the firing of all the transitions that are present in the SCN. This means that Is-enabled(ST) function has to check each transition in the SCN and is related to the transition ST if it is enabled or not.

- **PP** $\subseteq$ P is the set of port places. Each SCN contains places that are tagged with either in, out, or i/o. These places are named port places and permit the communication of a SCN with its peers. Similarly, all places connected to a substitution transition model a Web service in a composition are defined as input or output socket places. As mentioned before, each substitution transition is related to a SCN. This is achieved by providing a port assignment, which describes how the port places of the SCN are related to the socket places of the substitution transition.
- **PT** defines the type of the port, PT: PP $\rightarrow$ {in, out, i/o}. Each place connected to the substitution transition has a specific type.
- **PA** is a port assignment function that describes how the port places of the SCN related to the socket places of the substitution transition. Formally, this is represented as follows: **PA: ST$\subseteq$(PP$_{source}$xPP$_{target}$)** where PP$_{source}$ is a set of port node (especially input or output socket places of ST transition) and where PP$_{target}$ is a set of places in the target page (SA (ST)).
- **FP** is the first page of the CN, i.e., it represents the composite Web service. For each substitution transition in the first page, a SCN is obtained.

3.3 Illustration with Anatole scenario

The composition net of Anatole scenario is shown in Figure 1. It consists of seven Web services designed as substitution transitions, which permits abstracting their behavior. The substitutions transitions are: **LookforCenter**, **LookforDoctor**, **LookforEmergency**, **TreatmentTransmission**, **UpdatePatientRecord**, **Localization**, and **ContactMobileCare**.

The boxes that are next to each substitution transition specify the SCN that contains the detailed description of the activity represented by the corresponding substitution transition. For example, the page modeling **LookforCenter** Web service is modeled by the substitution transition named **LookforCenter**. The SCN of this substitution transition is shown in Figure 3. In addition, the types of input places that connect each substitution transition are "message" and "context". For example, the substitution transition for **LookforCenter** Web service is broken up into three context places (CP1, CP2, CP3), as an input and a single message place (PTLFC). These places are the input socket places for this substitution transition. For illustration purposes, the following assumptions are made regarding the context and message places:

- CP1 contains the required memory, e.g., 1'128 in Megabits, for the execution of the Web service.
- CP2 contains the time-slot availabilities of the Web service for execution.
- CP3 contains French language using 1'french. This represents the language the Web service uses for returning responses to users.
- PTLFC models the message between the PEM and **LookforCenter** Web service.

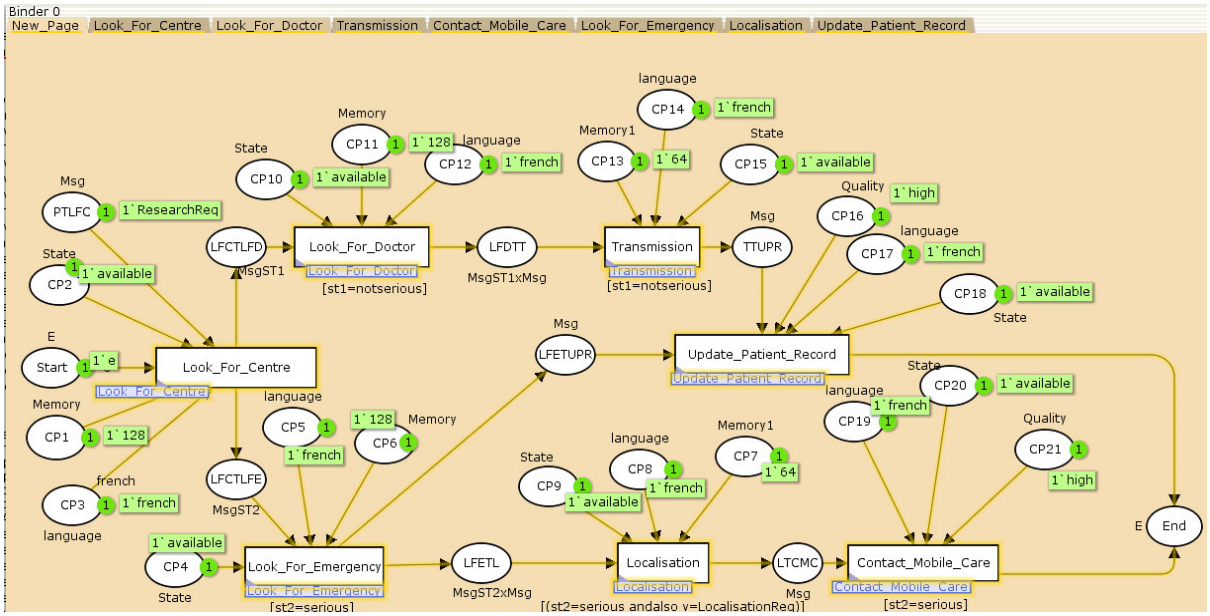

Figure 1 : The CN for Anatole scenario

The two output socket places for this substitution transition are:

- LFCTLFD models the message between **LookforCenter** and **LookforDoctor** Web services.
- LFCTLFE models the message between **LookforCenter** and **LookforEmergency** Web services.

Formally, the composition net of Anatole scenario is defined by the tuple **CN=<S,ST,SA,PP,PT,PA,PP>** where:

- **S**: six pages representing the existing Web services in Anatole scenario and the primary page.
- **ST**: six substitution transitions namely **LookforCenter**, **LookforDoctor**, **LookforEmergency**, **TreatmentTransmission**, **Localization**, and **ContactMobileCare**.
- $\mathbf{ST_{gu}}$**={LookforDoctor, LookforEmergency}**.
- $\mathbf{ST_{\overline{gu}}}$**={LookforCenter, TreatmentTransmission, Localization, ContactMobileCare}**.
- **SA**: six sub-pages are present in the composition net. Each sub-page represents a service composition net, which contains the detailed description of the Web services shown as substitution transition. It is assumed that each Web service composition's name has the same name of the substitution transition, for example **SA (LookforCenter)=SCNLFC.**
- **PP**: lists the distinct messages and context places that are associated with substitution transitions.
    - **LookforCenter:** Message places={PTLFC, LFCTLFD, LFCTLFE}, Context places={CP1, CP2, CP3}.
    - **LookforEmergency:** Message places={LFCTSE, SETL}, Context places={CP4, CP5, CP6}.

- **Localization:** Message places={LFETL, TTL, LTCS}, Context places={CP7, CP8, CP9}.

- **PT**: lists the different types of ports that are connected to each substitution transition that appears in the composition net of Anatole.
  - **LookforCenter**: input socket places={PTLFC, CP1, CP2, CP3}, output socket places={LFCTLFD, LFCTLFE}.
  - **LookforEmergency**: input socket places={LFCTLFE, CP4, CP5, CP6}, output socket place={LFETL}.
  - **Localization**: input socket places={LFETL, TTL, CP7, CP8, CP9}, output socket place={LTCMC}.
- **PA**: is the function that specifies the relationships between a substitution transition and its associated service colored net. This means describing how the port places of this net are related to the socket places of the substitution transition.
  - **LookforCenter: PP_{source}=**{PTLFC, CP1, CP2, CP3, LFTLFD, LFCTLFE}. The **SCN** of this substitution transition is named **LookforCenter**. The port nodes of this **SCN** are: PP_{target}={PTLFC, CP1, CP2, CP3, LFCTLFD, LFCTLFE}.These input ports are linked to the input socket places of **LookforCenter** substitution transition. Analogously, the two output ports LFCTLFD and LFCTLFE are related to the output socket places: LFCTLFD and LFCTLFE.
  - **LookforEmergency: PP_{source}=**{LFCTLFE, CP4, CP5, CP6, LFETL}. The **SCN** of this substitution transition is named **LookforEmergency**. Its port nodes are: PP_{target}**=**{LFCTLFE, CP4, CP5, CP6, LFETL}.
- **FP**: is the primary page as shown in Figure 3 .

3.4 Modeling a component Web service

Let us now consider the sub-page of Figure 3, which is about the detailed description of the activity that **LookforCenter** Web service carries out. The sub-page shows two operations captured by two transitions. We also consider **EvaluationState** transition in order to observe how the firing rules get initiated. **EvaluationState** transition is fired iff the following conditions are satisfied:

1. The color set of each place connected to EvaluationState transition is included in the color set of this transition. EvaluationState transition is only connected to P1 place, so the needed color-set for firing this transition is the Boolean color which corresponds to P1's colors set.
2. The arc's variables type matches the color of P1. As shown in Figure 2 , the arc connecting P1 place to EvaluationState transition only contains the ok variable. This variable has a Boolean color, which corresponds to the color of P1 place.

3. The ok variable must be assigned to a value with a specific type. Having a Boolean color, the ok variable can only receive true or false. Besides, this variable must have a type already defined in the color set of SCN_{LFC}.
4. EvaluationState transition is enabled if the ok variable in the guard condition evaluates to true.

In our case, the last condition depends on the value that is randomly assigned to the ok variable since all others conditions are satisfied.

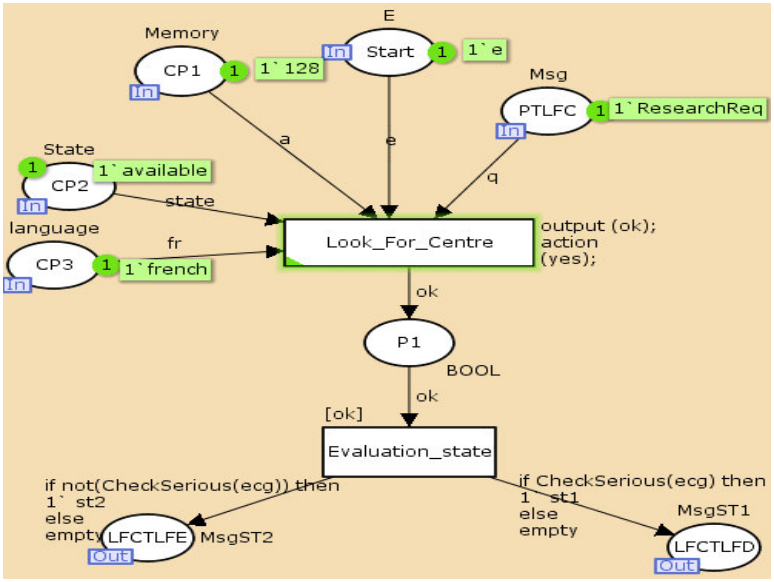

Figure 3 : SCN for LookforCenter Web service

3.5 Verifying a composite Web service

The use of high level Petri-Nets permits increasing the reliability level of composite Web services. Indeed, the checking of a composite Web service could happen prior to any deployment. The associated composition net could be subject to analysis using different techniques and computer tools for CPN [16]. These tools support interactive and automatic simulation. The most important analysis technique is known as a state space method [16]. It consists of designing a graph that has a node for each reachable marking (i.e., set of data value distribution in different places), as well as an arc for each occurring binding element. We suggest in the following, the definitions of some properties that can be checked using CPN:

- **Reachability** determines whether it is possible for a composition to achieve the desired results by using the state space method that generates all the reachable states.
- **Boundness** determines the minimal and the maximal number of tokens in the different places. If the type of a place is "context", then it can only contain a single token; otherwise errors are raised during the execution of the process. If the type of place is "message", then boundness checks buffer overflows.
- **Dead transition** determines the number of transitions which will never be enabled. If dead

transitions are initially detected, this means bad design.

- **Dead marking** is a marking with no enabled transitions. If the number of dead markings reported by the state space analysis is more than expected, then there must be an error in the composition design.

The verification of a composite Web service needs to take into account the current context that reflects the progress of this composite Web service. To apply HCPN modeling to a Web services composition according to a specific context, we introduce the concept of view.

4 The concept of view

4.1 Formal definition

We recall that a view is a dynamic snapshot over the specification of a composite Web service according to a certain context. We suggest below that a view is extracted out of the specification of a composite Web service using this time high level Petri-Nets. To this purpose, a set of definitions is provided.

- **Initial composition net definition.** Let us suppose that there is an Initial Composition Net (**ICN**) and a Derived Composition Net **(DCN)** and suppose also that **CN=ICN** $\cup$ **DCN**. With a high level abstraction, some attributes of the CN can be discarded. Hence, ICN is defined as the following triplet: **ICN=<S,ST**$_{\overline{gu}}$**,ST$_{gu}$>** where: **S** is the set of pages that are included in the CN where $\forall$ s$\in$S, s is an SCN, and ST$_{\overline{gu}}$ is and ST$_{gu}$= are like previously defined.
- **Context template definition.** Noted CT, it is the formal model of the corresponding context during view extraction. The CT includes two types of context: user (U-context) and Web service (W-context), **CT={U-context** $\cup$ **W-context}**. Additional details on contexts of user and Web service are given in [17].
- **Derived composition net definition**. The extraction of a view according to a certain context over an initial or derived composition specification permits obtaining a **DCN. DCN** is defined with the following triplet: a **DCN**=<**S'**,**S'T**$_{\overline{gu}}$**,S'T$_{gu}$>** where:
  - **S'**: is the derived specification that does not accept any additional Web services through their pages.
  - **S'T**$_{\overline{gu}}$**={ st' |** $\exists$ **st**$\in$ **ST**$_{\overline{gu}}$ $\wedge$ **Is-enabled (st)=*true*}** is the new set of substitution transitions without guard conditions.
  - **ST$_{gu}$={ st' |** $\exists$ **st**$\in$ **ST$_{gu}$** $\wedge$ **Is-enabled (st)=*true*}** is the new set of substitution transitions with guard conditions. **Is-enabled** function is already defined. It checks if the conditions relating to the different transitions (with or without guard condition) are satisfied in the

context running **CT**.

4.2 Application to Anatole scenario

Let us assume that Anatole's context returns details on his physical state and localization. Others details on Web services contexts are also returned. The context template is as follows:

- CT={U-context ∪ W-context}.
- SU-context={Identity="Anatole", Age="60", Gender="Male"}.
- DU-context={PsychologicalState="stressed", PhysicalState="serious", Localization="Fourvière Cathedral"}.

An example of Web service context is the context of **ContactMobileCare** Web service:

- W-context={SW-context ∪ DW-context}.
- SW-context={Name= "ContactMobileCare", Memory= "128", language= "French"}.
- DW-context={availability="no"}.

In Figure 1, the CN of Anatole scenario is illustrated. The net is a triple CN = <S, $ST_{\underline{gu}}$, ST_{gu}> where

- **S**={SCN_{LFC}, SCN_{LFD}, SCN_{T}, SCN_{UR}, SCN_{LFE}, SCN_{L}, SCN_{CS}}.
- **ST_{gu}**={LookforDoctor, LookforEmergency, Localization, ContactMobileCare }.
- **$ST_{\underline{gu}}$** = {LookforCenter, UpdatePatientRecord}.

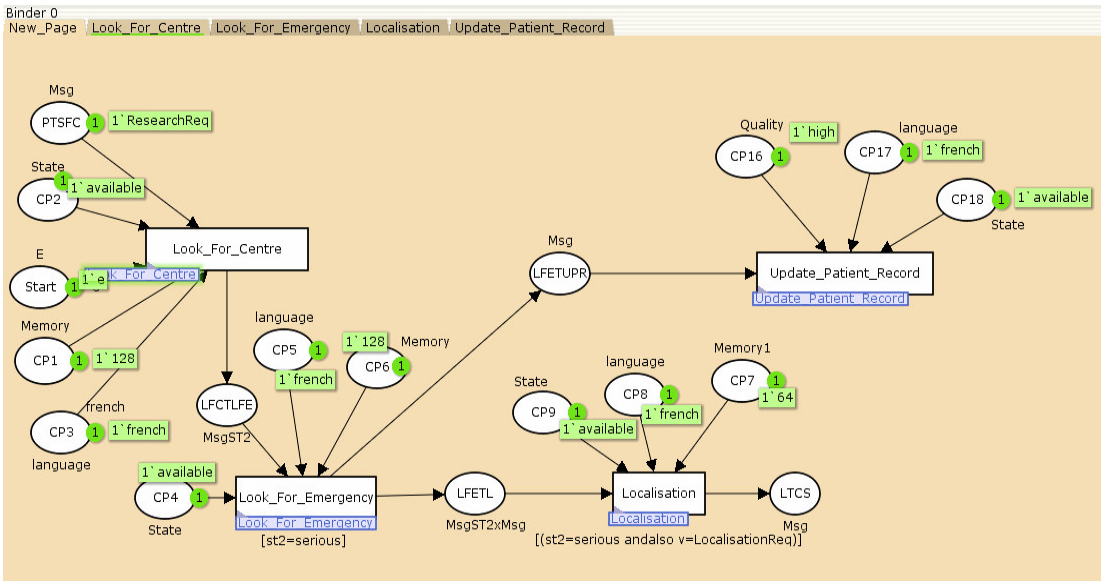

Figure 4 : DCN for Anatole scenario

Figure 4 shows the derived composition net that is extracted out of the composition net of Figure 1 according to the defined context template. The DCN is defined by the triple <S',S'$T_{\underline{gu}}$, S'T_{gu}> where

S'={SCN_{LFC}, SCN_{LFE}, SCN_{UR}, SCN_{L}}, ST_{gu}={Localization, UpdatePatientRecord}, and $ST_{\underline{gu}}$={LookforCenter, LookforEmergency}. The testing of the various concepts that were discussed in this paper is presented in the next section.

5 Prototype

A prototype that supports our proposal for using high level Petri-Nets in the specification and verification of views over composite Web services is fully operational. We used Java to implement the needed functionalities for context collection and generation as well as for view extraction. The main architecture of the prototype is shown in Figure 5. The prototype comprises two modules that a Java program orchestrates. The first module is about the context generator and the second is the view extraction.

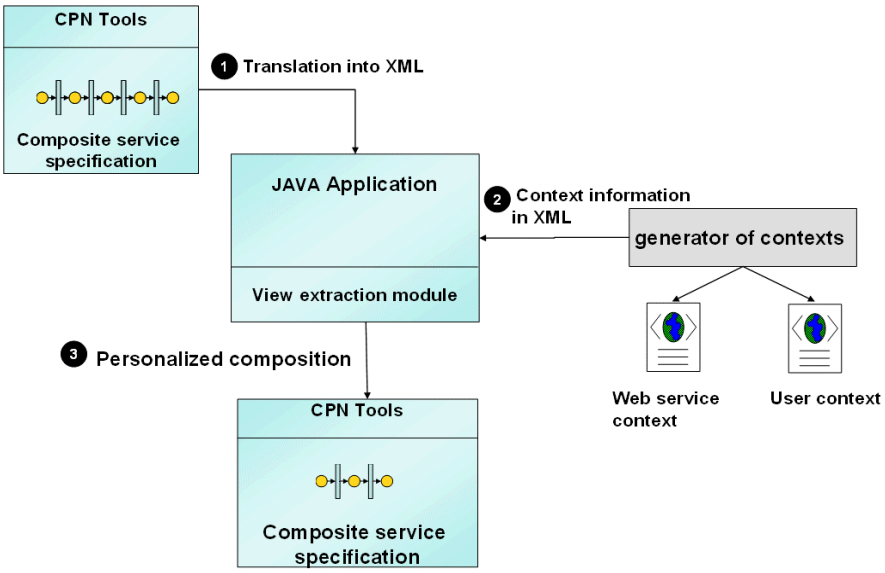

Figure 5: Architecture of the prototype

For demonstration purposes, we assumed that the context generator provides upon request several contextual details related to users and Web services. To this purpose, two XML files are obtained out of the context generator. Both files are then submitted to the view extraction module, which requires an additional input, which is the XML file that contains the composite Web service specification in a high level Petri-Net form, i.e., the initial composition net. We used CPN Tools, which is a tool for editing, simulating and analyzing Colored Petri-Nets. The extraction of a view consists of comparing the expected contextual elements that are associated with this specification to the current contextual details that are obtained out of the context generator. The result of the comparison is an XML file that describes the derived composition net. The XML file corresponds to the view that can now be visualized as a Petri-Net using the CPN Tools and verified using the various properties we listed in Section 3.4.

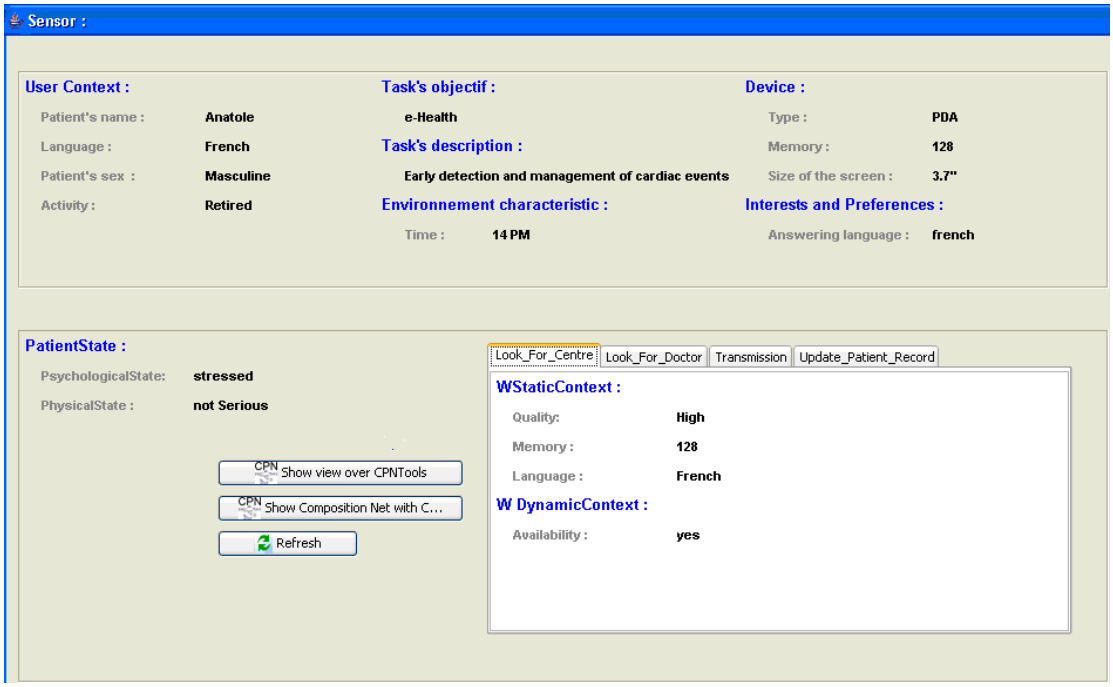

Figure 6 : Snap shot of the developed prototype

In Figure 6, the context generator provides details on the context of Anatole (e.g., PhysicalState="not serious", PsychologicalState="stressed") as well as details on the context of Web services such as **LookforCenter.** This Web service is available and uses French to return responses to users.

6 Conclusion

In this paper, we presented a high level Petri-Net approach for the specification and verification of composite Web services. Our literature review has shown the importance of developing composite Web services and making them reliable and efficient, i.e., error-free. This definitely calls for formal techniques to use as part of the verification process of developing composite Web services prior to any deployment. Our literature review has also shown that few initiatives dealt with context during the specification exercise of a composite Web service. Therefore, we proposed a high level Petri-Net approach that integrates context during specification, maps this specification onto a Petri-Net, and finally checks the obtained Petri-Net with some automatic tools. Furthermore, we discussed in this paper how the execution of a composite Web service is tracked using the concept of view. We illustrated and prototyped the dual use of Petri-Nets and views with a patient-related scenario. Although this scenario was fictitious and simple, it revealed the challenges that need to be taken up when deploying Web services in critical domains such as healthcare. Our next work aims at proposing extensions for BPEL4WS so that user and Web service contexts are included. In addition, we aim at developing a tool that converts an extended BPEL specification into a colored Petri-Net for automatic verification purposes of this specification.

References

[1] F. Daniel and B. Pernici, "Insights into Web Service Orchestration and Choreography," *International Journal of E-Business Research, The Idea Group Inc.*, vol. 1, pp. 58 - 77, 2005.

[2] F. Leymann, "Web services flow language (WSFL1.0) ", IBM Software Group May 2001.

[3] D. Levy, "Coordination of web services : langages de description et plate-formes d'exécution," Septembre 2002.

[4] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, F. J. Klein, K. L. Leymann, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S.Weerawarana, "Business Process Execution Language for Web Services (BPEL4WS) version 1.1", May-2003.

[5] J. Mendling and M. Müller, "A Comparison of BPML and BPEL4WS," in *R. Tolksdorf, R. Eckstein (eds.): Proceedings of the 1st Conference "Berliner XML-Tage"*. Berlin, Germany, October 2003.

[6] Y. Yang, Q. Tan, J. Yu, and F. Liu, "Transformation BPEL to CP-nets for verifying Web services composition", in *Proceedings of The International Conference on Next Generation Web Services Practices (NWeSP'05)*. Seoul, Korea, 2005.

[7] Y. Yang, Q. Tan, J. Yu, and F. Liu, "Verifying Web Services composition Based on Hierarchical Coloured Petri Nets", in *Proceedings of the first international workshop on Interoperability of heterogeneous information systems*. Bremen, Germany, 2005.

[8] B. Medjahed, A. Bouguettaya, and A. K. Elmagarmid, "Composing Web services on the Semantic Web," *International Journal on Very Large Data Bases*, vol. 12, pp. 333-351, 2003.

[9] Z. Maamar, S. K. Mostefaoui, and H. Yahyaoui, "Toward an agent-based and context-oriented approach for Web services composition", *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, pp. 686-697, 2005.

[10] Z. Maamar, D. Benslimane, C. Ghedira, and M. Mrissa, "On tracking personalized Web services using views", in *Proceedings of The IEEE International Conference on e-Technology, e-Commerce and e-Service, (EEE'05),* Hong Kong, China, 2005.

[11] K. Jensen, *Coloured Petri Nets: Basic Concepts, Analysis Methods, and Practical Use*, 2nd ed. Berlin; New York: Springer, 1997.

[12] C. Petri, "Kommunikation mit Automaten". Germany: University of Bonn, 1962.

[13] B. Benatallah and H. Rachid, "A Petri net-based model for web service composition", in *Proceedings of the Fourteenth Australasian database conference on Database technologies*. Adelaide, Australia, 2003.

[14] Y. Xiaochuan and K. Krys J, "Process Composition of Web Services with Complex Conversation Protocols: a colored Petri Nets Based Approach", in *Proceedings of The Design, Analysis and Simulation of Distributed Systems Conference*, 2004.

[15] H. Bing, "Choreography Modelling and Analysis of a Travel Reservation Web Service", in *Proceedings of The Fifth International Joint Conference on Autonomous Agents & Multi-Agent Systems*. Hakodate, Japan, 2006.

[16] CPN Tools, "http://wiki.daimi.au.dk/cpntools/cpntools.wiki", Aarhus University, 2006.

[17] C. Ghedira and H. Mezni, "Through Personalized Web Service Composition Specification: From BPEL to C-BPEL", *Electronic Notes in Theoretical Computer Science*, vol. 146, pp. 117-132, 2006.