BPEL-RF Tool: An Automatic Translation from WS-BPEL/WSRF Specifications to Petri Nets

María Díaz, Valentín Valero, Hermenegilda Macià, Jose Antonio Mateo, Gregorio Díaz
Informatics Research Institute of Albacete (I3A) Albacete, Spain
Email: {Maria.DiazTello, Valentin.Valero, Hermenegilda.Macia, JoseAntonio.Mateo, Gregorio.Diaz}@uclm.es.

***Abstract*—Composite Web services technologies are widely used due to their ability to provide interoperability among services from different companies. Thus, orchestration languages like WS-BPEL have recently appeared to manage the interactions of multiple services in order to achieve a global aim. Web services are usually *stateless*, which means that no state is stored from the clients viewpoint. However, some new applications and services have emerged, which require to capture the state of some resources. Therefore, new standards to model Web services states have arisen, such as Web Services Resource Framework (WSRF). In this paper, we present a tool, which takes as input a specification in BPEL-RF (a language defined on the basis of both standards), and transforms it into a prioritised-timed coloured Petri net (PTCPN). These PTCPNs can be verified and validated with the well-known tool, CPNTools.**

***Keywords*-Web Service compositions; WS-BPEL; WSRF; Coloured Petri nets; Tool support; Stateful workflows**

I. INTRODUCTION

The development of software systems is becoming more complex with the appearance of new computational paradigms such as Service-Oriented Computing (SOC), Grid Computing and Cloud Computing. In January of 2004, several members of the *Globus Alliance* organization and the computer multinational *IBM* with the help of experts from companies such as *HP, SAP, Akamai, etc.* defined the basis architecture and the initial specification documents of a new standard to describe distributed resources, Web Services Resource Framework (WSRF) [10]. The WSRF elements that are considered in the language BPEL-RF are:

- **WS-ResourceProperties**: There is a precise specification to define WS-Resource properties, based on a Resource Properties Document (RPD), which represents the properties of the associated resource (disk size, processor capacity, etc.). Nevertheless, for simplicity, we only consider a single property for each resource, which is an integer value. Resources are identified by their EPRs (End-Point References); so, we will also use this mechanism for identification purposes, but, for simplicity, we will consider these references as static, instead of assuming a dynamic mechanism to assign them. As a shorthand notation, EPRs will also be used to denote the resource property values. Among the operations allowed by the standard are *GetResourceProperty* and *SetResourceProperty*, which are used to manipulate the resource property values.
- **WS-ResourceLifetime**: The WSRF specification does not provide a standard way to create resources. However, resources have an associated lifetime, which means that once this time has elapsed, the resource is considered to be destroyed. We have then included, for completeness, an operation to create resources, *createResource*, in which the initial value of the resource, its lifetime and the activity that must be launched upon its destruction are indicated. We also have an operation in order to modify the current resource lifetime, *setTimeout*.
- **WS-Notification**: Clients can subscribe to WSRF resources in order to be notified about some topics (resource conditions). We therefore include the *subscribe* operator, indicating the condition under which the subscriber must be notified, and the activity that must be executed upon that event.

WS-BPEL [3], for short BPEL, is an OASIS orchestration language for specifying actions within Web service business processes. BPEL is an orchestration language in the sense that it is used to define the composition of services from a local viewpoint, describing the individual behaviour of each participant. BPEL processes use *variables* to temporarily store data. Variables are, therefore, declared on a process or on a scope within that process. In our case, there will be a single scope (*root*); so, no nesting is considered here. Besides, for simplicity again, we will only deal with integer variables.

An orchestrator consists of a main activity, representing the normal behaviour of this participant. There are also event and fault activities, which are executed upon the occurrence of some events, or due to some execution failures, respectively. BPEL activities can be *basic* or *structured*. *Basic activities* are those which describe the elemental steps of the process behaviour, such as the assignment of variables (*assign*), empty action (*empty*), time delay (*wait*), invoke a service (*invoke*) and receive a message (*receive*), reply to a client (*reply*), and throw an exception (*throw*). We also have an action to *terminate* the process execution at any moment (*exit*). For technical reasons, we have also included a barred form of *reply* action, which is used when a service invocation expects a reply, in order to implement the synchronization with the *reply* action from the server. *Structured activities* encode control-flow logic in a nested way. The considered structured activities are the following: a *sequence* of activities, separated by a semicolon, the parallel composition, represented by two parallel bars ($\|$), the conditional repetitive behaviour (*while*), and a timed extension of the

 ISBN: 978-1-61208-230-1

receive activity, which allows to receive different types of messages with a time-out associated (*pick*).

The main motivation of this work is to provide a formal semantics for WS-BPEL+WSRF to manage stateful Web services workflows by using the existing machinery in distributed systems, and specifically a well-known formalism, such as prioritised-timed coloured Petri nets (PTCPN), which are a graphical model that also provide us with the ability to simulate and analyse the modelled system. In order to deal with the integration of BPEL plus WSRF in a proper way, we have realised that it is more convenient to introduce a specific semantic model, which covers properly all the relevant aspects of WSRF such as notifications and resource time-outs. The integration of both standards is not new; in the literature, there are a bundle of works defining this integration, but none of these works define a formal semantics in terms of Petri nets.

In [16], the integration of BPEL in Grid environments is considered, and the author discusses the benefits and challenges of extensibility in the particular case of OGSI workflows combined with WSRF-based Grids. Other two works centred around Grid environments are [8] and [11]. The first one justifies the use of BPEL extensibility to allow the combination of different GRIDs, whereas Ezenwoye et al. [8] share their experience on BPEL to create and manage WS-Resources that implement the factory/instance pattern in bioinformatics. On the other hand, Ouyang et al. [15] define the necessary elements for translating BPEL processes into Petri nets. Thus, they cover all the important aspects in the standard such as exception handling, dead path elimination, and so on. The model they consider differs from ours in that we formalise the whole system as a composition of orchestrators with resources associated, whereas they describe the system as a general scope with nested sub-scopes leaving aside the possibility of administering resources. Besides, we have also formalized the event handling and notification mechanisms. Following this translation, in [14], Ouyang et al present the tool WofBPEL and a companion tool, BPEL2PNML. The idea behind is to provide tool support for the analysis of BPEL processes. Related to π-calculus semantics, Dragoni and Mazzara [6] propose a theoretical scheme focused on dependable composition for the WS-BPEL recovery framework. In this approach, the recovery framework is simplified and analysed via a conservative extension of π-calculus. The aim of this approach clearly differs from ours, but it helps us to have a better understanding of the WS-BPEL recovery framework. In addition, we also consider time constraints. Moreover, we would like to highlight the work of Farahbod et al. [9] and Busi et al. [4]. In the first one, the authors extract an abstract operational semantics for BPEL based on abstract state machines (ASM) defining the framework BPEL$_{AM}$ to manage the agents who perform the workflow activities. In this approach, time constraints are considered, but they do not formalize the timed model. In the second one, they also define a π-calculus operational semantics for BPEL and describe a conformance notion. They present all the machinery to model web service compositions (choreographies and orchestrations). The main difference with our work is that we deal with distributed resources.

Finally, in the literature one can find several tools performing the opposite translation, i.e., from Petri nets into BPEL. In [2], van der Aalst and Lassen present the theory and implementation of a translation between WF-nets and BPEL. The implementation is performed via the tool WorkflowNet2BPEL4WS. This tool automatically translates coloured Petri nets, CPNs, into BPEL code. These CPNs are specified using CPN Tools [5]. Other similar proposal, WoPeD [7], is a Java-based tool that provides an easy-to-use software for modelling, simulating and analysing workflow processes and resource descriptions. WoPeD supports the CPN notation and the standard file format of WoPeD is PNML, allowing model exchange with other Petri net tools. After this introduction, Section II shows briefly the language BPEL-RF, whereas Section III presents indeed the tool. Section IV contains a case study so as to illustrate how the tool works. Finally, Section V finishes the paper with some conclusions and possible future work.

II. BPEL-RF LANGUAGE

In this section, we are going to present briefly the main characteristics of the language called BPEL-RF (Business Process Execution language for the Resource Framework). An operational semantics for this language was presented in our previous work [12], and the corresponding translation to prioritised-timed coloured Petri nets in [13]. Due to the lack of space, we omit here these transformations, so the interested reader can refer to [12], [13] for them.

We use the following notation: *ORCH* is the set of orchestrators in the system, *Var* is the set of integer variable names, *PL* is the set of necessary partnerlinks, *OPS* is the set of operations names that can be performed, *EPRS* is the set of resource identifiers, and *A* is the set of basic or structured activities that can form the body of a process. Note that each orchestrator uses its own variables despite we have not separated *Var* in its corresponding subsets.

The specific algebraic language, then, that we use for the activities is defined by the following BNF-notation:

$$
\begin{aligned}
&A ::= throw \mid \underline{receive}(pl, op, v) \mid invoke(pl, op, v_1) \mid exit \mid \\
&\ reply(pl, v) \mid \overline{reply}(pl, op, v_2) \mid assign(expr, v_1) \mid empty \mid \\
&\ A\,;A \ \mid \ A \,\|\, A \mid while(cond, A) \mid wait(timeout) \mid \\
&pick(\{(pl_i, op_i, v_i, A_i)\}_{i=1}^{n}, A, timeout) \mid \\
&createResource(EPR, val, timeout, O, A) \mid \\
&getProp(EPR, v) \mid setProp(EPR, expr) \mid \\
&setTimeout(EPR, timeout) \mid \\
&subscribe(O, EPR, cond', A)
\end{aligned}
$$

where $O \in ORCH, EPR \in EPRS, pl, pl_i \in PL, op, op_i \in OPS, timeout \in \mathbb{N}, expr$ is an arithmetic expression constructed by using the variables in *Var* and integers; v, v_1, v_2, v_i range over *Var*, and $val \in \mathbb{Z}$. A condition $cond$ is a predicate constructed by using conjunctions, disjunctions, and negations over the set of variables Var and integers, whereas $cond'$ is a predicate constructed by using the corresponding EPR (as the resource value) and integers. Notice that

setProp and *getProp* do not contain the property name since, for simplicity, we are only considering a single property for each resource. We therefore use its EPR as representative of this property, as we already observed in the introduction. Note that we do not take into consideration correlation sets, dynamic partnerlinks or instance creation, since we only deal with the static aspects of WS-BPEL. We plan as part of our future work an extension of this operational semantics enriched with these additional constructions, as well as with the inclusion of structured variables, instead of just considering all variables as integers. An orchestration is now defined as a tuple $O = (PL, Var, A, A_f, \mathcal{A}_e)$, where A and A_f are activities defined by the previous syntax and $\mathcal{A}_e$ is a set of activities. Specifically, A represents the normal workflow, A_f is the orchestrator fault handling activity and $\mathcal{A}_e = \{A_{e_i}\}_{i=0}^{m}$ are the event handling activities.

III. BPEL-RF TOOL

As WS-BPEL and WSRF are XML-based languages, and the PTCPNs supported by CPNTools are also represented by XML files, we have used XSLT stylesheets to transform the BPEL-RF document into another XML document representing the PTCPN in a format supported by CPNTools. These XSL stylesheets are created using a XSLT editor. The obtained XML document can be visualized, simulated and verified with CPNTools. As the tool has been developed in Java, it is multiplatform, i.e., runs on Windows/Linux/Mac systems under the Java virtual machine® (the tool is available at [1]). The XSLT transformation sheets (*eXtensible Stylesheets Language/Transform*) are a W3C declarative language to transform XML documents into other XML documents or to some other kind of documents. The XSLT stylesheets are widely used, as an easy way to apply transformation rules to a source document in order to obtain the corresponding output documents. Nowadays, XSLT is widely recommended in web edition area, due to its ability to generate HTML or XHTML sheets.

For making that transformation, XSLT allows to convert the input in two ways: On the one hand, the programmer can manipulate the contents of the document to organize them without changing the document format, whereas, on the other hand, the programmer can use XSLT sheets to transform the contents into other different formats.

We have then defined a number of rules to extract the PTCPN elements from the choreography defined as a composition of WS-BPEL documents. Thus, our tool, BPEL-RF, is used to achieve this transformation in an automatic way, presenting to the user a *.cpn file*, which can be opened with CPNTools. After doing this, the user can analyse and verify the model by using the features of CPNTools.

The XSLT stylesheet document starts with the instruction $\langle\ ?xml\ version =' 1.0'?\rangle$. The element root is a stylesheet, which contains all other elements. In an XSLT stylesheet, the name of reserved elements by the specification comes from the same namespace, so they must be written preceded by the appropriate alias that must point to the URL: http://www.w3c.org/1999/XSL/Transform.
In Fig. 1, we show a piece of the structure of the XSLT document.
Once we have located the initial and final mark of the root element "xsl:stylesheet", we define the transformation rules:

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

```
<?xml version="1.0" ?>
<xsl:stylesheet
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform
version="1.0">
<xsl:output indent="yes" />
<xsl:template match="/">
<workspaceElements>
<generator tool="CPN Tools" version="3.2.2" format="6" />
<cpnet>
...
<page id="ID6">
<template>
<xsl:for-each select="//process">
<xsl:for-each select="child::*">
<xsl:if test="(name()='pick')">
<xsl:call-template name="pick" />
<xsl:call-template name="picktrans" />
</xsl:if>
....
</template>
</page>
...
</cpnet>
</workspaceElements>
</template>
</stylesheet>
```

Figure 1. Illustration of an XSLT template

For the sake of simplicity, BPEL-RF Tool has a very simple and intuitive interface shown in Fig. 2. It consists of a main frame with separated elements such as a file menu and the transformation panel. The file menu has three different submenus, namely: *File*, *CPN Tools* and *Help*. The *File* submenu offers two options. The first one, *Open WS-BPEL WSRF File*, opens a BPEL-RF document previously edited and saved with the tool; whereas the second one, *Exit*, exits the program. The *CPN Tools* submenu only offers one option, *Save Coloured Petri Net*, which saves the translated XML code to a .cpn file. Finally, the last submenu, *Help*, consists of two options *Help* and *About*. The option *About* only informs users about the tool version, the option *Help* offers users a wide user manual with the possibility of searching through the information using either a table of contents or a search option.

 ISBN: 978-1-61208-230-1

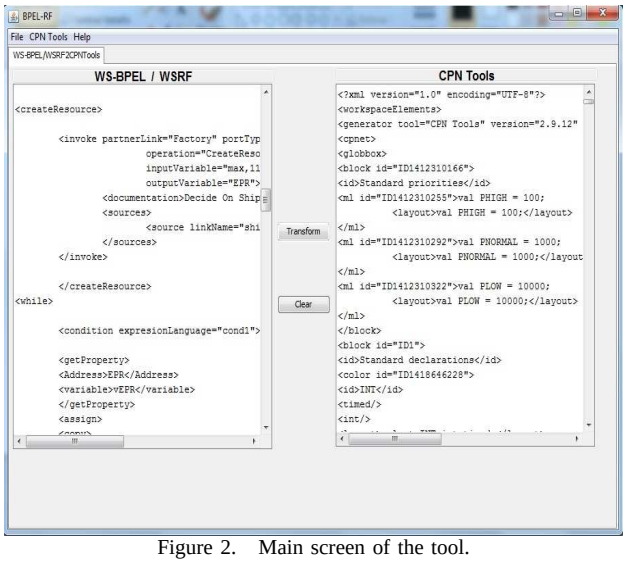

Figure 2. Main screen of the tool.

The main elements of the interface are:

- The **WS-BPEL / WSRF Textbox** permits users to introduce XML code following the specification given by WS-BPEL and WSRF. This XML is used as the source code to be translated into PTCPN. This code can be introduced in two ways; either by writing the XML code by hand or by loading a previously saved document using the *Open WS-BPEL WSRF File* submenu mentioned above. A dialog window will be shown to the user asking him to select the document to be opened. If the file is not valid, an error message will be displayed on the screen.
- In the **CPNTools Textbox**, after clicking on the button "Transform", the corresponding Petri Net XML specification is shown. To save this specification, the user must click on the *Save Colored Petri Net File* option in the CPN Tools menu. A dialog window will be shown to the user to choose the destination folder.

Moreover, we have another two buttons on the screen:

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

IV. Case Study: Automatic Management System for Stock Market Investments

The case study concerns a typical automatic management system for stock market investments, which consists of *n+1* participants: the online stock market system and *n* investors, A_i, $i = 1, \ldots, n$. Here, the resource will be the stocks of a company that the investors want to buy just in case the price falls below an established limit, which the investors fix previously by means of subscriptions, i.e., an investor subscribes to the resource (the stocks) with a certain guard (the value of the stocks he/she want to pay for it). The lifetime *lft* will be determined by the stock market system and the resource price will be fluctuating to simulate the rises/drops of the stock. Notice that we do not take into account the stock buy process since our aim is to model an investors' information system. Thus, the participants will be notified when their bids hold or the resource lifetime expires. Let us consider the choreography $C = (O_{sys}, O_1, \ldots, O_n)$, where $O_k = (PL_k, Var_k, A_k, A_{f_k}, \mathcal{A}_{e_k})$, k=sys, 1,..., n; $Var_{sys} = \{at, vEPR\}$, $Var_i = \{v_i\}$, $A_{f_k} = exit$. Variable $vEPR$ serves to temporarily store the value of the resource property before being sent; v_i is the variable used for the interaction among participants, and, finally, at controls the period of time in which the auction is active. Note that the value *x* indicates the resource value at the beginning, *at0* is the time that the "auction" is active, and, finally, x_i is the value of the stocks that he/she wants to pay for. Suppose that all the variables are initially 0:

$$
\begin{aligned}
Asys = {} & assign(x+1, vEPR); assign(at0, at); \\
& CreateResource(EPR, lft, x, empty); \\
& while(actualTime() <= at, Abid) \\
Abid = {} & getProp(EPR, vEPR); assign(vEPR + bid(), vEPR); \\
& setProp(EPR, vEPR); wait(1, 2) \\
A_i = {} & wait(1, 2); subscribe(O_i, EPR, EPR < x_i, Acond_i); \\
& pick((pl_i, buy, v_i, empty), empty, at0) \\
Acond_i = {} & getProp(EPR, vEPR); invoke(pl_i, buy, vEPR)
\end{aligned}
$$

Here, the function *bid* is used to increase/decrease the stocks value simulating the fluctuation of the stocks price.

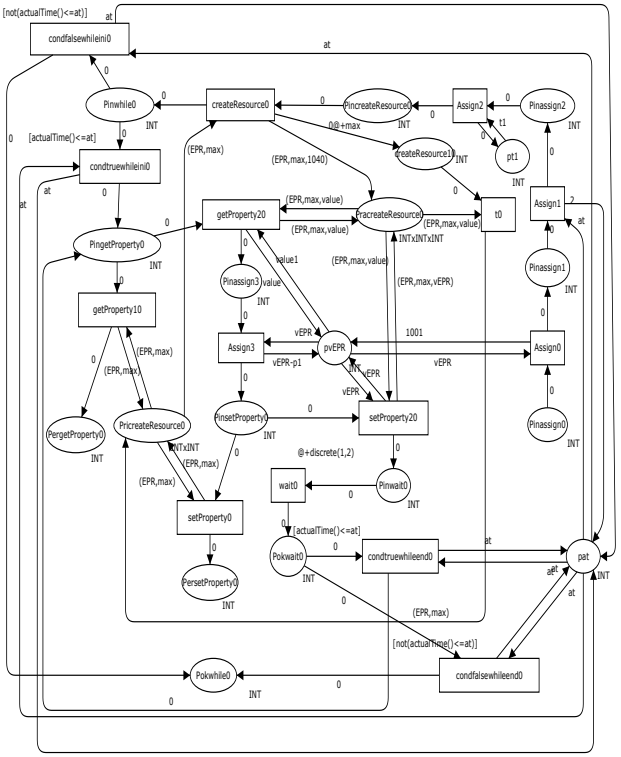

Figure 3. PTCPN of the online stock market.

In Figs. 3 and 4, the PTCPNs for one buyer and for the system are depicted. These figures have been obtained automatically by using our tool.

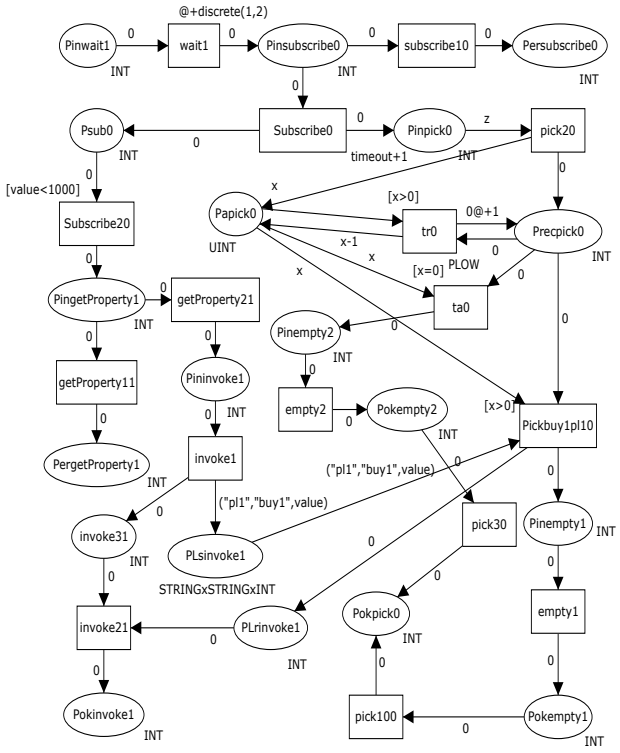

Figure 4. PTCPN of one buyer.

A. Analysis

CPNTools offers us two forms to check the correctness of our system: formal verification and simulation. First, the simulation helps designers to understand how the system exactly works and it is a mean to detect possible errors in early stages of the development process in order to refine the model according the clients' requirements. Besides, formal verification through state space analysis could be done in order to ensure that our system achieves some formal properties such as liveness, deadlock-freeness, and so on. In this way, Table I shows the results obtained considering 1, 2, 3, 4 or 5 investors. Note that we have considered the following assumptions:

- The "auction" time *at0* is limited to 10 time units.
- The resource is active during 15 time units (*lft=15*).
- The resource value *x* is 100 money units.
- The value of subscription of each investor *i*, x_i, is $x-(9+i)$, that is, if the system has only one investor its subscription guard will be $x < 90$, whereas with 5 investors, the last investor will have a subscription guard of $x < 86$.
- The function *bid* will fluctuate the stocks price between -2 and 1 in order to simulate that the price only can rise 1 and drop 2 at most each time unit.

We will focus on deadlock-freeness to ensure that the system never gets stuck while the participants have activities to do in their workflow. We have leveraged the functions offered by CPNTools to demonstrate that in all dead markings of the system the final place is marked, which leads us to conclude the system has finished correctly. This final place, *Pokfinal0*, is marked by a transition when all the participants have finished their workflow. For the sake of clarity, we have not drawn this place in each figure. Thus, the next SML code checks when this situation occurs: fun DesiredTerminal n =((Mark.PetriNet'Pokfinal0 1 n) == 1'true), which returns *true* if the place *Pokfinal0* is marked. In addition, it is needed to evaluate the following predicate: PredAllNodes DesiredTerminal=ListDeadMarkings(), to check
that the list of dead marking contains the marking of the *Pokfinal0* place.

Properties	Number of investors				
	1	2	3	4	5
State Space Nodes	3561	7569	16983	50350	89879
State Space Arcs	5203	12843	33271	112101	262215
Time (s)	2	7	23	146	1140
Dead Markings	124	244	454	1108	874

Table I
STATE SPACE ANALYSIS RESULTS

In Fig. 5, we show the results offered by CPNTools to our queries for the case of **three** investors. Here, it can be appreciated that all dead markings hold the predicate *DesiredTerminal*, and, therefore, when the system reaches a dead marking is because system has terminated, which demonstrates the absence of deadlocks in our case study.

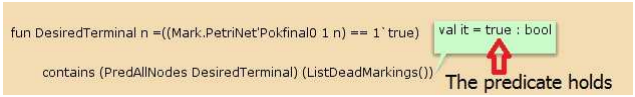

Figure 5. Result of the queries in CPNTools.

V. CONCLUSIONS AND FUTURE WORKS

In this paper, a tool which permits the automatic translation between BPEL-RF specifications and PTCPNs supported by CPNTools has been presented. This is a great advantage with respect to our previous works in such a way the user only needs to provide the XML code for the orchestration and the tool will extract automatically the corresponding translation in order to effectuate the formal analysis of the system. This analysis can be done by simulation or by formal verification. In the case study, we have centred on formal verification looking for the absence of deadlocks in the model. Finally, as future work, we plan to extend our work with additional features of both WS-BPEL and WSRF, as the discovery of existing resources. We are also working on the demonstration of the equivalence between the operational semantics of [12] and the Petri nets semantics of [13].

ACKNOWLEDGMENT

This work has been partially supported by CICYT project TIN2009-14312-C02-02, and JCCM project PEII09-0232-7745.

REFERENCES

[1] [retrieved:September,2012] BPEL-RF tool web site, http://www.dsi.uclm.es/retics/BPELRF/

[2] W. M. P van der Aalst and K. B. Lassen. Translating unstructured workflow processes to readable BPEL: Theory and implementation, Journal of Information Software Technology, vol. 50, number 3, pp. 131-159, 2008.

[3] [retrieved:September,2012] Alexandre Alves, Assaf Arkin, Charlton Barreto, Ben Bloch,

Francisco Curbera, and Rania Khalaf. Web Services Business Process Execution Language, http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

[4] N. Busi, R. Gorrieri, C. Guidi, R. Lucchi, and G.Zavattaro, Choreography and Orchestration: A Synergic Approach for System Design. In International Conference of Service Oriented Computing (ICSOC), Lecture Notes in Computer Science, vol. 3826, pp. 228-240, 2005.

[5] [retrieved:September,2012] CPNTools official web site, http://cpntools.org.

[6] N. Dragoni and M. Mazzara, A formal Semantics for the WS-BPEL Recovery Framework - The pi-Calculus Way. In International Workshop on Web Services and Formal Methods (WS-FM). Lecture Notes in Computer Science, vol. 6194, pp. 92-109, 2009.

[7] A. Eckleder and T. Freytag, WoPeD 2.0 goes BPEL 2.0. In 15th German Workshop on Algorithms and Tools for Petri Nets, Algorithmen und Werkzeuge für Petrinetze (AWPN 2008). CEUR Workshop Proceedings, vol. 380, pp. 75-80, 2008.

[8] O. Ezenwoye, S.M. Sadjadi, A. Cary, and M. Robinson, Grid Service Composition in BPEL for Scientific Applications. In OTM Conferences, pp. 1304-1312, 2007.

[9] R. Farahbod, U. Glässer, and M. Vajihollahi, A Formal Semantics for the Business Process Execution Language for Web Services. In Joint Workshop on Web Services and Model-Driven Enterprise Information Services (WSMDEIS), pp. 122-133, 2005.

[10] [retrieved:September,2012] I. Foster, J. Frey, S. Graham, S. Tuecke, K. Czajkowski, D. Ferguson, F. Leymann, M. Nally, T. Storey, and S. Weerawaranna, Modeling Stateful Resources with Web Services, http://www.globus.org/wsrf/.

[11] F. Leymann, Choreography for the Grid: towards fitting BPEL to the resource framework. Journal of Concurrency and Computation : Practice & Experience, vol. 18, issue 10, pp. 1201-1217, 2006.

[12] J.A. Mateo, V. Valero, and G. Diaz, An Operational Semantics of BPEL Orchestrations Integrating Web Services Resource Framework. In International Workshop on Web Services and Formal Methods (WS-FM), 2011.

[13] [retrieved:September,2012] J.A. Mateo, V. Valero, H. Macià, and G. Diaz. A Coloured Petri Net Approach to Model and Analyse Stateful Workflows Based on WS-BPEL and WSRF. Technical Report DIAB-12-04-2, University of Castilla-La Mancha. Available at: http://www.dsi.uclm.es/trep.php?codtrep=DIAB-12-04-2

[14] C. Ouyang, E. Verbeek, W. M. P. van der Aalst, S. Breutel, M. Dumas, and A. ter Hofstede, WofBPEL: A Tool for Automated Analysis of BPEL Processes., In Third International Conference on Service-Oriented Computing (ICSOC 2005). Lecture Notes in Computer Science, vol. 3826, pp. 484-489, 2005.

[15] C. Ouyang, E. Verbeek, W.M.P. van der Aalst, S. Breutel, M. Dumas, and A.H.M. ter Hofstede. Formal semantics and analysis of control flow in WS-BPEL. Science of Computing Programming, vol. 67, issue 2-3, pp. 162-198, 2007.

[16] A. Slomiski. On using BPEL extensibility to implement OGSI and WSRF Grid workflows. Journal of Concurrency and Computation : Practice & Experience, vol. 18, pp. 1229-1241, 2006.

 ISBN: 978-1-61208-230-1