From OWL-S Descriptions to Petri Nets*

Antonio Brogi, Sara Corfini, and Stefano Iardella

Department of Computer Science, University of Pisa, Italy

Abstract. While OWL-S advertisements provide a rich (ontological and behavioural) description of Web services, there are no tools that support formal analyses of OWL-S services. In this paper we present a translator from OWL-S descriptions to Petri nets which makes such analyses possible thanks to the many tools available for Petri nets.

1 Introduction

Service-oriented Computing [1] centers on the notion of service as the fundamental element for developing distributed software applications. Web service composition is receiving increasing attention as it fosters rapid application development via service reuse. WSDL [2] is the current standard for describing Web service interfaces, yet, WSDL descriptions do not include information on the interaction behaviour of services. This inhibits the possibility of *a priori* verifying important behavioural properties of service compositions, such as lock-freedom or replaceability.

Various efforts have been recently devoted to propose more expressive description languages capable of modelling service behaviour. The OWL-based Web Service ontology (OWL-S, [3]) is one of the major efforts in this direction. OWL-S is a computer-interpretable semantic mark-up language, where, for the first time, ontology-based descriptions of service functionality and of interaction service behaviour coexist. This is probably the major strength of OWL-S, which paves the way for the full automation of service discovery, invocation and composition. In particular, OWL-S service descriptions (e.g., the service *process model*) provide the needed information for the *a priori* analysis and verification of service invocations and compositions.

While some tools for describing OWL-S services (e.g., Protégé [4]) and for matching OWL-S service advertisements (e.g., OWLSM [5]) are already available, there is no tool – at the best of our knowledge – that supports formal analyses of OWL-S services.

The main objective of this work was to feature a tool – named OWLS2PNML – capable of translating OWL-S service descriptions into Petri nets, thus paving the way for analysing and verifying OWL-S services by exploiting some of the many tools available for Petri nets [6].

In particular, OWLS2PNML translates an OWL-S process model (expressing service behaviour) into a Petri net described by a PNML file. The Petri Net

* Work partially supported by the SMEPP project (EU-FP6-IST 0333563).

E. Di Nitto and M. Ripeanu (Eds.): ICSOC 2007 Workshops, LNCS 4907, pp. 427–438, 2009.

Mark-up Language (PNML, [7]) is an XML-based interchange format for describing Petri nets. There are two good reasons for choosing PNML:

- First, PNML is emerging as the *de facto* standard language for expressing (different types of) Petri nets, and various Petri net tools capable of manipulating PNML files are available (e.g., [8,9]).
- Second, there are translators available capable of converting other types of service descriptions into PNML. This is particularly interesting since it is not realistic to imagine that *all* service descriptions will be written in OWL-S. For instance, WS-BPEL [10] has recently become the OASIS standard for expressing Web service compositions. The availability of translators (like [11]) from WS-BPEL to PNML hence suggests the adoption of PNML as a *lingua franca* for expressing and for reasoning about heterogeneous service descriptions.

The two main contributions of OWLS2PNML can be hence summarised as follows:

1. OWLS2PNML makes it possible to animate OWL-S services. Indeed, after translating OWL-S process models into PNML files, the latters can be loaded into some available Petri net tool (e.g., [8]) and animated in order to observe the possible actual service behaviour.
2. OWLS2PNML makes it possible to reason with heterogeneous services. For instance, we can translate an OWL-S service and a WS-BPEL service into PNML, and then check their (behavioural) equivalence by exploiting the various existing (Petri net-based) algorithms and tools (e.g., [12]). The availability of suitable congruences for Web services expressed in terms of Petri nets (e.g., [13]) allows to address two crucial issues in Service-oriented Computing: service replaceability and modular service development.

The OWLS2PNML translator has been developed in the context of the SAM project [14]. SAM is a matchmaking system for discovering compositions of (OWL-S) semantic Web services. One of the main feature of SAM is the ability of performing a behaviour-aware matching, namely, SAM is able to discover (compositions of) services that feature a desired behaviour. SAM defines a Petri net-based methodology for checking the equivalence of service behaviour, hence, OWLS2PNML directly plugs-in into SAM by translating OWL-S services into Petri nets.

The rest of the paper is organised as follows. Section 2 provides a short introduction to OWL-S and to PNML. OWLS2PNML is described in Section 3, after presenting a Petri nets semantics for OWL-S. Finally, we discuss related work and we draw some concluding remarks in Section 4.

2 A Short Introduction to OWL-S and PNML

The next two subsections provide a short recap on OWL-S and PNML. A complete specification of OWL-S and PNML can be found in [3] and [7], respectively.

2.1 OWL-S: Semantic Markup for Web Services

OWL-S [3] is an ontology-based language for semantically describing services. An OWL-S advertisement is structured in three parts, viz. the *service profile*, the *process model* and the *grounding*, each of them providing a different view of a service. Briefly, the *service profile* provides a high-level description of a service, which includes both functional (i.e., inputs/outputs) and extra-functional properties (i.e., service name, service category, cost/quality of service, and so on). The *process model* details the service behaviour, in particular, it describes how the service performs its component tasks. The *service grounding* explains how to access the service by specifying protocol and message format information.

The OWL-S process model is the starting point of the OWLS2PNML translator. More precisely, the process model describes a service as a composite process which consists, in turn, of composite processes and/or atomic processes. An atomic process can not be decomposed further and it executes in a single step (similarly to a *black box* providing a functionality), while a composite process is built up by using a few control constructs: `sequence` (i.e., sequential execution), `if-then-else` (conditional execution), `choice` (non-deterministic execution), `split` (parallel execution), `split+join` (parallel execution with synchronization), `any-order` (unordered sequential execution), `repeat-while` and `repeat-until` (iterative execution). Hence, for instance, an `if-then-else` process is a bag of two processes out of which one is chosen for execution according to the value of a condition, an `any-order` process is a bag of processes to be executed in some unspecified order but not concurrently, and a `repeat-until` process is a process to be executed at least one, until a condition becomes true.

2.2 PNML: Petri Net Markup Language

The Petri Net Markup Language (PNML) [7] is an XML-based interchange format for Petri nets, which is independent of specific tools and platforms. The PNML technology is structured in three (fixed) parts: the *meta model*, which defines the basic structure of a PNML file, the *type definition interface*, which allows the definition of new Petri net types, and the *feature definition interface*, which allows the definition of new features for Petri nets. The PNML technology is complemented by an evolving part, namely the *conventions document*, which contains the definition of a set of standard features of Petri nets.

The meta model of (basic) PNML defines those objects that basically represent the graph structure of Petri nets, that is, *places*, *transitions* and *arcs*. Each object may have *labels*, whose functionality is to assign further meaning to objects. For example, a label can represent the name of a place/transition, the initial marking of a place, or the inscription of an arc. The definition of all standard labels is provided (and maintained) by the conventions document, while the legal (combinations of) labels of a specific Petri net type are defined in a *Petri Net Type Definition* (PNTD) document. The XML syntax of the PNML meta model is straightforward: PNML objects (viz., places, transitions and arcs) are translated into *PNML elements* (viz., `<place>`, `<transition>` and `<arc>`).

Additional XML elements are defined for the labels defined by specific Petri net types. For example, the element `<name>` could be a label for a place object (i.e., `<name>` is a child XML element of `<place>`). Each element within a PNML file has a unique identifier, which can be used to refer to this element.

3 From OWL-S to PNML

3.1 A Petri Nets Semantic for OWL-S

As we anticipated in the introduction, OWLS2PNML has been designed in the context of a matchmaking system for discovering compositions of semantic Web services, called SAM (for Service Aggregation Matchmaking). SAM models service behaviour by means of CPR nets (for Consume-Produce-Read nets), a variant of standard condition/event[1] Petri nets that we defined in [13]. A feature of CPR nets is that they are equipped with two disjoint sets of places, namely, *control* places (to be consumed and produced) and *data* places (to be produced and read). Hereafter, we recall the formal definition of CPR nets.

Definition 1 (CPR net). *A* consume-produce-read *net (simply,* CPR net*) N is a tuple $(C_N, D_N, T_N, F_N, I_N)$ where*

- C_N *is a finite set of* control places*,*
- D_N *is a finite set of* data places *(disjoint from C_N),*
- T_N *is a finite set of* transitions*,*
- $F_N \subseteq (C_N \times T_N) \cup (T_N \times C_N)$ *is the* control flow relation*,*
- $I_N \subseteq (D_N \times T_N) \cup (T_N \times D_N)$ *is the* data flow relation*.*

When defining CPR nets in [13], we considered that an OWL-S atomic operation can be executed only if all its inputs are available and all the operations that must occur before its execution have been completed (according to [3]). Consequently, we mapped atomic operations into transitions, and we employed places and transition firing rules to model both the availability of data (viz., the *data flow*) and the executability of atomic operations (viz., the *control flow*). In particular, an atomic operation T is modelled as a transition t having an input/output data place for each input/output of T, an input control place to denote that t is executable, as well as an output control place to denote that t has completed its execution. The graphical notation of CPR nets is illustrated in Figure 1, where diamonds represent control places, while circles and rectangles represent data places and transitions, respectively.

As illustrated by the straight lines of Figure 1, data places can be read, produced but not consumed. This feature of CPR nets is motivated by SAM, which abstracts from the *multiplicity* of data in modelling services [14].

[1] In this paper as well as in [13] we consider Petri nets which describe the "life-cycle" of a single service session. Yet, the properties of C/E Petri nets avoid that (possible) different service sessions overlap.

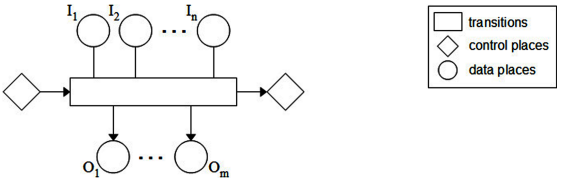

Fig. 1. Modelling atomic operations as CPR net transitions

We defined the formal encoding of OWL-S composite processes into CPR nets in [15]. Yet, for the lack of space, we include hereafter only a short description of how OWL-S composite processes can be directly mapped into CPR nets.

Let us define a service as a triple (i, P, f) where P denotes the CPR net representing the service, and i and f denote the initial and the final control places of P, respectively. To define *compositional* operators it is sufficient to properly coordinate the initial and final control places of the employed services. For instance, let us consider the *sequential composition* of two services $(i_1, P1, f_1)$ and $(i_2, P2, f_2)$. This is a CPR net consisting of the two services plus a transition whose starting control place is f_1 and the final control place is i_2. By doing so, $P1$ has to be completed before $P2$ can start.

The mapping of OWL-S composite processes into CPR nets is illustrated in Figure 2, where the PX-labelled boxes represent (i_x, P_X, f_x) services, the dark gray rectangles denote empty transitions, and the light gray diamonds denote the starting and final control places of the resulting nets. Note that, to simplify reading, we omitted data places from the nets of Figure 2.

Petri nets have been already employed to model Web services. In particular, we refer to the *WorkFlow nets* (WFN) defined by van der Aalst in [16] and designed to model the workflow of services. In [17] WFNs have been enriched (i.e., *open* WFNs) with communication places, which constitute the interface of the net. We introduced CPR nets since both WFNs and open WFNs abstract from the (whole) data flow of a service, that, yet, is crucial for matchmaking systems (e.g., [14]) which discover, compose and analyse *semantic* services relying on their ontology-annotated data.

3.2 A Petri Net Type Definition for CPR Nets

One of the major guiding principles of PNML is *flexibility*, that is the ability of representing any kind of Petri nets with its specific extensions and features. In order to define a new type of Petri net supported to PNML, it suffices: (1) to extend the conventions document with the specific feature of the new Petri net type, and (2) to determine the legal labels for the new Petri net type.

The presence of two disjoint sets of data and control places is the distinguishing feature of CPR nets. In particular, a data place is associated with the type of the data it represents (in case of OWL-S, a type is an ontology concept). We hence differentiate between data and control places, and define the PNML specification of CPR nets as a simple extension of the PNML specification of standard

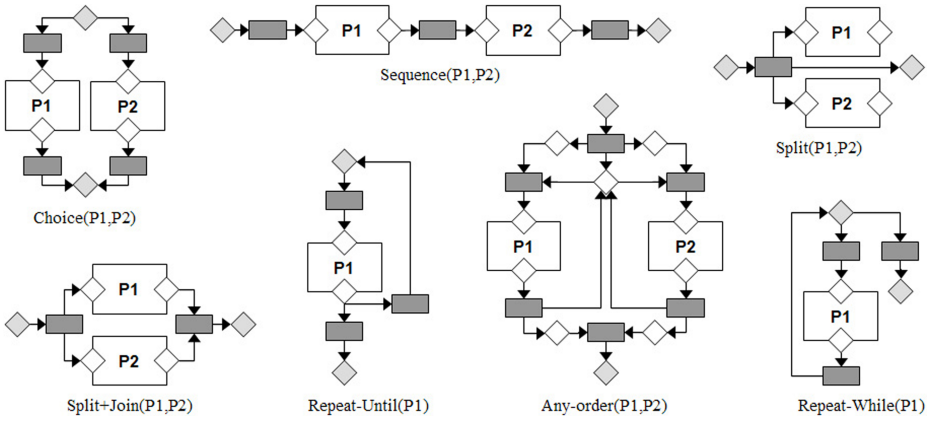

Fig. 2. Modelling OWL-S composite operations as CPR nets

condition/event nets. We first extended the conventions document with the definition of a new label `<ontology>`. The value of an `<ontology>` label is the URI identifying a specific ontology concept. Next, we established the legal labels for CPR nets in a new Petri Net Type Definition file. In particular, `<place>` elements can use labels `<name>`, `<ontology>` and `<initialMarking>`, `<transition>` elements can use `<name>` labels, and `<arc>` elements can use `<inscription>` labels. The presence of an `<ontology>` label allows us to distinguish between data and control places.

3.3 OWLS2PNML: Implementation Details

The OWLS2PNML prototype implements the Petri nets semantic for OWL-S described in [13] and summarised in Subsection 3.1.

A non-trivial issue in implementing OWLS2PNML was how to handle the sharing of the input/output data among different processes. Indeed, the OWL-S process model allows – via the mechanism of the input/output binding – to define when an input/output data has to be shared by multiple processes. OWLS2PNML addresses such an issue by performing a suitable analysis of OWL-S process models. Namely, for instance, if two atomic processes p_1 and p_2 share an input/output data d, OWLS2PNML creates a single data place d which is linked with both transitions p_1 and p_2.

OWLS2PNML has been implemented as a Java servlet and it is accessible from `http://www.di.unipi.it/~corfini/owls2pnml.html`. As one may note in Figure 3, the simple Web interface of OWLS2PNML requires as input an URL (or a file system path) pointing to an OWL-S service description, and returns as output the PNML code describing the Petri net representation of the given service.

The choice of Java as developing language has been mainly motivated by the availability of many supporting libraries, such as the MindSwap OWL-S API (`http://www.mindswap.org/2004/owl-s/api/`) to parse OWL-S descriptions, and

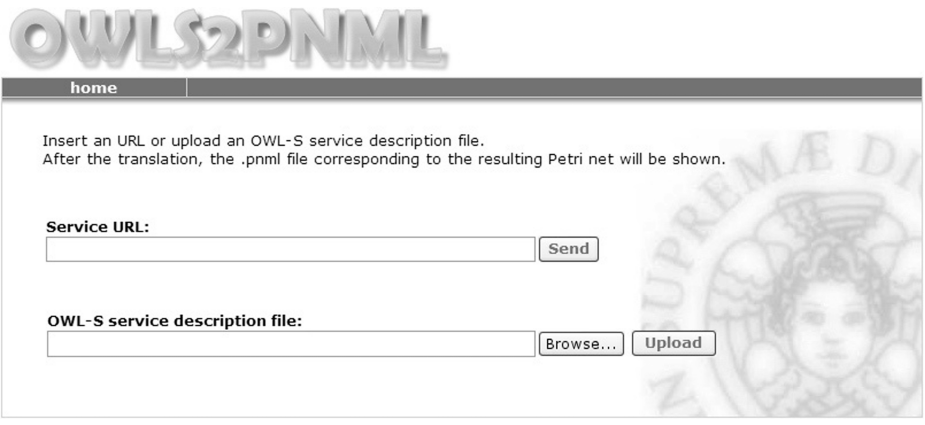

Fig. 3. The OWLS2PNML Web interface

the XML JDOM API (`http://www.jdom.org/`) to write PNML files. Although the MindSwap OWL-S API presents some limitations (e.g., it does not cope with OWL-S pre-conditions and effects[2]), it is the only one to deal with all the OWL-S control constructs. In spite of the availability of specific instruments to generate PNML code, such as the PNML framework (`http://www-src.lip6.fr/logiciels/mars/PNML/pfUsers.html`), we used the more widely known XML JDOM API.

3.4 Examples

In this subsection we illustrate the contribution of OWLS2PNML by presenting two examples which highlight the importance of animating (OWL-S) services and of checking service equivalence, respectively.

Let us first discuss the importance of animating OWL-S services to detect undesired behaviour. Consider the ExampleOne service, operating in the bank domain, which builds a credit offer, given the requested amount of credit, the balance and the guarantee provided by a customer. The full OWL-S code of the ExampleOne process model, as well as of all the services employed in this subsection, is available at `http://www.di.unipi.it/~corfini/owls/processmodels/`. For the convenience of the reader, we show in Figure 4 a more compact – yet manually built – tree representation of the ExampleOne process model, rather than listing the actual OWL-S code (which is more than 500 lines long).

As one can note in Figure 4, ExampleOne consists of an `any-order` process composed, in turn, of a `sequence` process and of an atomic process. An `any-order` process is a bag of processes to be executed in some unspecified order but not concurrently. In particular, for the ExampleOne service, this means that if the execution of the `any-order` process starts from the `sequence` process,

[2] Such limitation was not a problem for developing the OWLS2PNML prototype as OWL-S pre-conditions and effects are not present in the service descriptions translated by OWLS2PNML.

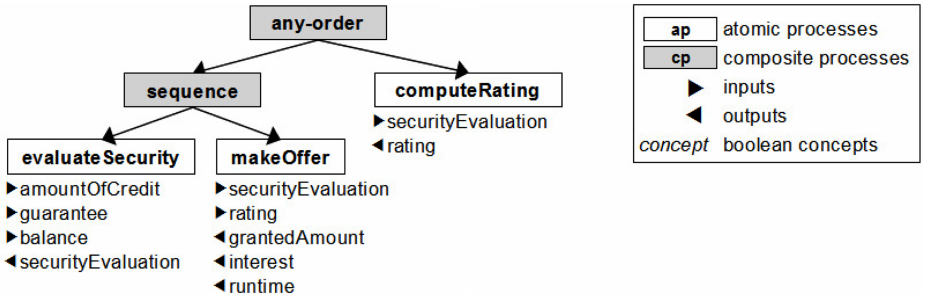

Fig. 4. The OWL-S process model of the ExampleOne service

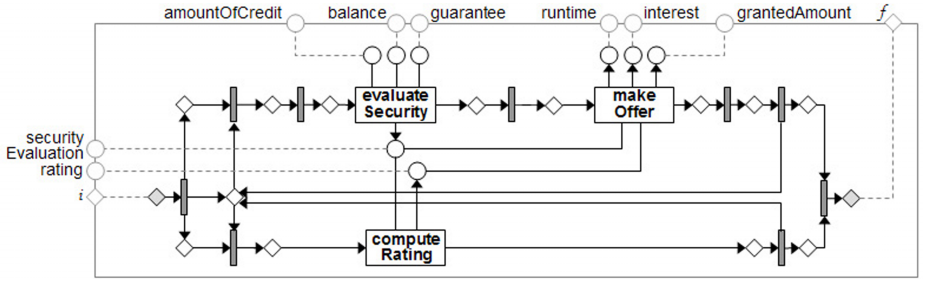

Fig. 5. CPR net representation of the ExampleOne service

all the child processes of the `sequence` (i.e., `evaluateSecurity` and `makeOffer`) have to be completed before executing the computeRating atomic process. Yet, we may note that this behaviour (i.e., the sequential execution of evaluateSecurity, makeOffer, computeRating) leads to a dead-lock. Indeed, given as input amountOfCredit, balance and guarantee, the evaluateSecurity atomic process is executed, yet, makeOffer can not execute, as it needs rating as input, which is provided by computeRating, that, in turn, can not be executed before makeOffer completes. Regrettably, this undesired behaviour is hard to be detected by directly analysing the long "static" OWL-S code. OWLS2PNML enables the possibility of animating OWL-S services, so to verify the actual service behaviour.

OWLS2PNML generates a Petri net representation of a given OWL-S process model, that can be next simulated by exploiting some of the many existing tools for simulating nets (e.g., WoPeD [8]). Figure 5 illustrates the Petri net generated by OWLS2PNML for the ExampleOne service. By animating the net, namely, by inserting tokens in the data places corresponding to amountOfCredit, balance and guarantee, it is easy to detect the above described undesired behaviour. Indeed, the net simulation stops after executing evaluateSecurity.

Let us now discuss the issue of *service replaceability*, that is, the ability of verifying whether a service taking part in a complex application can be replaced with a different service, without altering the behaviour of the whole application. Consider, for instance, the RatingOne service taking part in a credit banking

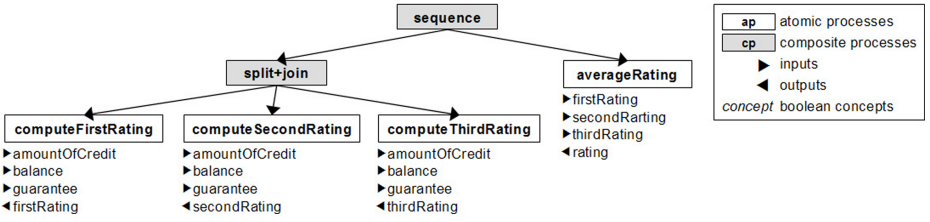

Fig. 6. The OWL-S process model of the RatingOne service

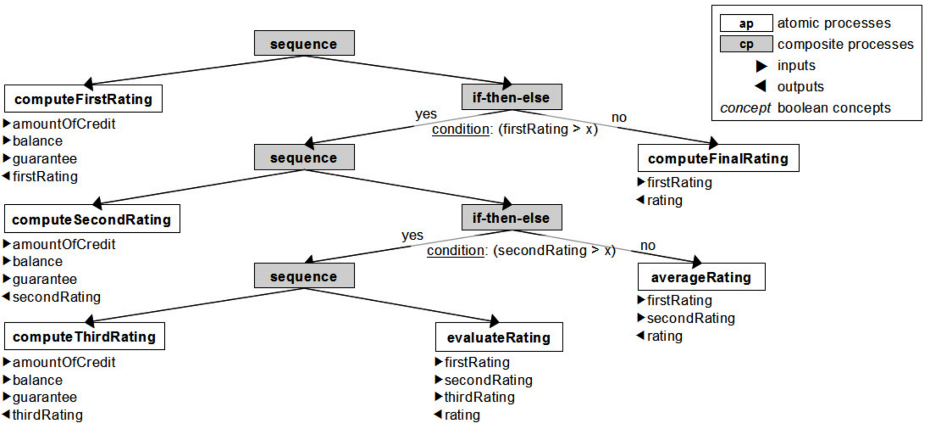

Fig. 7. The OWL-S process model of the RatingTwo service

system. RatingOne takes as input the requested amount of credit, the balance and the provided guarantee of a customer, it computes three separate evaluations of the customer and next it returns an average rating. For the convenience of the reader, we give in Figure 6 a tree-view of the OWL-S process model describing RatingOne, thus avoiding to list the complete OWL-S code. Let us now suppose that the banking system wants to substitute RatingOne with the RatingTwo service. RatingTwo, illustrated in Figure 7, may be more convenient for the bank, as it does not always compute three separate and expensive customer evaluations, e.g., it computes a second and next a third customer evaluation only if the previous rating exceeds a threshold value.

The problem for the bank is hence to establish if RatingOne can be replaced with RatingTwo, without altering the behaviour of its credit system. Yet, by concentrating on the OWL-S descriptions (i.e., XML-based code) of RatingOne and RatingTwo, there is no the possibility of easily verifying the behavioural equivalence of the two services, so to establish that RatingTwo can replace RatingOne.

OWLS2PNML enables the analysis of OWL-S services, since the translation of OWL-S services into Petri nets allows the re-use of the many methodologies, algorithms and tools (e.g., [12]) developed to check the equivalence of Petri nets. For example, let us consider the behavioural congruence for Web services defined in [13], where service behaviour is modelled by means of CPR nets.

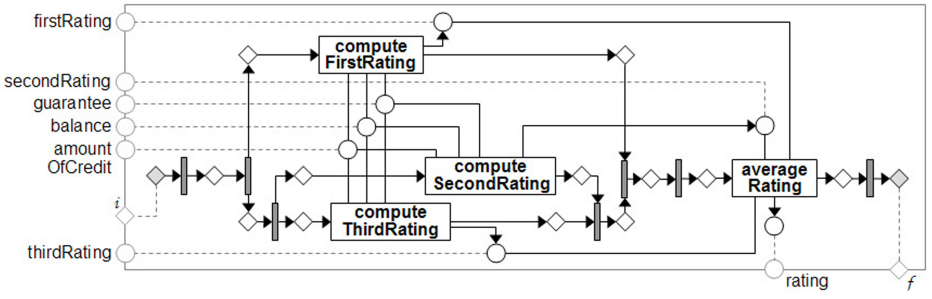

Fig. 8. CPR net representation of the RatingOne service

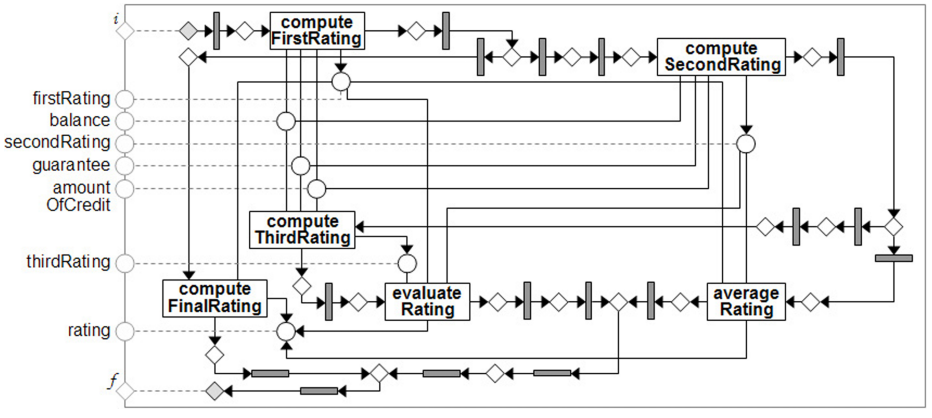

Fig. 9. CPR net representation of the RatingTwo service

The behavioural congruence of [13] defines equivalent two given nets N_1 and N_2 if each state of N_1 can be reached by N_2 (and vice versa). A state of a CPR net is the marking of its *observable* places, that is, those places that can be externally observed and that can interact with the external environment. Hence, such a congruence abstracts from internal transition steps by equating structurally different, yet externally indistinguishable services.

Figures 8 and 9 illustrate the (graphical representation of the) CPR nets achieved by executing OWLS2PNML on RatingOne and RatingTwo. Suppose that the bank is interested only in observing the final rating, not the intermediate ones. The observable places are hence the following: amountOfCredit, balance, guarantee and rating. Furthermore, note that the initial control place i as well as the final control place f have to be observed, in order to check the correct termination of both services. We can hence equate the nets of Figures 8 and 9 without observing firstRating, secondRating and thirdRating. By doing so, the nets are equivalent, indeed, although structurally different, they result externally indistinguishable. It is worth observing again that it is not possible to verify such an equivalence by directly reasoning on OWL-S descriptions.

4 Concluding Remarks

In this paper we presented a tool – named OWSL2PNML – capable of translating OWL-S service descriptions into Petri nets. As already anticipated in the introduction, the main contribution of OWSL2PNML is to make possible to analyse and verify OWL-S services, as it allows to employ some of the many tools available for Petri nets. OWSL2PNML, which implements the Petri net semantics for OWL-S that we described in [13], has been developed in the context of the SAM project [14], whose objective is the discovery of compositions of OWL-S semantic Web services featuring a specific behaviour.

The first Petri nets semantics for DAML-S (the predecessor of OWL-S) was defined by Narayanan and McIlraith in [18]. Yet, their semantics is not compositional, and it does not deal with the `any-order` control construct. The Petri nets semantics of [18] was implemented by the DaGen tool [19], which translates DAML-S descriptions into reference Petri nets. DaGen is a plug-in for the Reference Net Workshop (Renew, `http://www.renew.de`), a Petri net simulator that allows for the graphical drawing and for the execution of reference nets. Yet, DaGen does not generate intermediary files describing the Petri net resulting from the translation process. On the other hand, as we stated in the introduction, we believe that the ability of OWLS2PNML to describe nets by means of PNML files is suitable for two good reasons: (1) various Petri nets tools capable of manipulating PNML files are available [8,9], and (2) PNML can be used as *lingua franca* for expressing and for reasoning about service descriptions, since the availability of translators, e.g., from WS-BPEL into PNML [11].

It is also worth mentioning the work of Elenius et al. that presented in [4] an OWL-S editor, implemented as a plug-in of the Protégé OWL ontology editor (`http://protege.stanford.edu`). An interesting under development feature of [4] is the ability of executing OWL-S services, currently restricted to atomic processes only. Indeed, [4] is partially able to process the OWL-S grounding, which describes how to concretely invoke the service operations. Note that OWSL2PNML does not analyse the OWL-S grounding as it is the objective of other components of the SAM project, where OWSL2PNML takes part.

OWSL2PNML is still a prototype, and as such it has some known limitations. One of them is that OWSL2PNML requires correct OWL-S descriptions as input. We plan to add a validation step in order to verify the correctness of the OWL-S descriptions to be translated. A second line for future work can be the handling of graphical information, which can be associated to each element (i.e., place, transition, arc) of the net, in order to allow Petri net tools to visualise the "best-view" of the loaded net. Another important direction of future work is to complement OWLS2PNML with a tool capable of checking the behavioural equivalence of different services expressed as CPR nets, in particular by implementing the definifion of CPR net bisimulation presented in [13]. Finally, an interesting (although perhaps non-trivial) issue is to convert Petri nets into OWL-S descriptions. For instance, a Petri net-based matchmaking system (e.g., [14]) may exploit such a translator to generate a OWL-S description of the result of its discovery process.

References

1. Papazoglou, M.P., Georgakopoulos, D.: Service-Oriented Computing. Communications of the ACM 46(10), 24–28 (2003)
2. WSDL Coaltion: Web Service Description Language (WSDL) version 2.0 (2007), http://www.w3.org/TR/wsdl20/
3. OWL-S Coalition: OWL-S: Semantic Markup for Web Service (2006), http://www.ai.sri.com/daml/services/owl-s/1.2/overview/
4. Elenius, D., Denker, G., Martin, D., Gilham, F., Khouri, J., Sadaati, S., Senanayake, R.: The OWL-S Editor - A Development Tool for Semantic Web Services. In: Gómez-Pérez, A., Euzenat, J. (eds.) ESWC 2005. LNCS, vol. 3532, pp. 78–92. Springer, Heidelberg (2005)
5. Jaeger, M., et al.: OWLSM (2004), http://owlsm.projects.semwebcentral.org/
6. (Petri nets World), http://www.informatik.uni-hamburg.de/TGI/PetriNets/
7. Billington, J., Christensen, S., van Hee, K.M., Kindler, E., Kummer, O., Petrucci, L., Post, R., Stehno, C., Weber, M.: The Petri Net Markup Language: Concepts, Technology, and Tools. In: van der Aalst, W.M.P., Best, E. (eds.) ICATPN 2003. LNCS, vol. 2679, pp. 483–505. Springer, Heidelberg (2003)
8. The WoPeD Team: Workflow Petri Net Designer (2007), http://www.woped.org/
9. Research Group Petri Net Technology: Petri Net Kernel (2002), http://www2.informatik.hu-berlin.de/top/pnk/
10. BPEL Coalition: WS-BPEL 2.0 (2006), http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf
11. Ouyang, C., Verbeek, E., van der Aalst, W., Breutel, S., Dumas, M., ter Hofstede, A.: WofBPEL: A Tool for Automated Analysis of BPEL Processes. In: Benatallah, B., Casati, F., Traverso, P. (eds.) ICSOC 2005. LNCS, vol. 3826, pp. 484–489. Springer, Heidelberg (2005)
12. Fernandez, J.C., Mounier, L.: On the Fly verification of behavioural equivalences and preorders. In: Larsen, K.G., Skou, A. (eds.) CAV 1991. LNCS, vol. 575, pp. 181–191. Springer, Heidelberg (1992)
13. Bonchi, F., Brogi, A., Corfini, S., Gadducci, F.: A Behavioural Congruence for Web services. In: Arbab, F., Sirjani, M. (eds.) FSEN 2007. LNCS, vol. 4767, pp. 240–256. Springer, Heidelberg (2007)
14. Benigni, F., Brogi, A., Corfini, S.: Discovering Service Compositions That Feature a Desired Behaviour. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 56–68. Springer, Heidelberg (2007)
15. Bonchi, F., Brogi, A., Corfini, S., Gadducci, F.: Compositional Specification of Web Services via Behavioural Equivalence: A Case Study. Technical Report TR-08-01, Computer Science Department, University of Pisa (2008), http://compass2.di.unipi.it/TR/Files/TR-08-01.pdf.gz
16. van der Aalst, W.M.P.: The Application of Petri Nets to Workflow Management. Journal of Circuits, Systems, and Computers 8(1), 21–66 (1998)
17. Massuthe, P., Reisig, W., Schmidt, K.: An operating guideline approach to the SOA. Annals of Mathematics, Computing & Teleinformatics 1(3), 35–43 (2005)
18. Narayanan, S., McIlraith, S.A.: Simulation, verification and automated composition of web services. In: WWW 2002, pp. 77–88. ACM Press, New York (2002)
19. Moldt, D., Ortmann, J.: DaGen: A Tool for Automatic Translation from DAML-S to High-Level Petri Nets. In: Wermelinger, M., Margaria-Steffen, T. (eds.) FASE 2004. LNCS, vol. 2984, pp. 209–213. Springer, Heidelberg (2004)