Approaching Web Service Coordination and Composition by Means of Petri Nets. The Case of the Nets-within-Nets Paradigm*

P. Álvarez, J.A. Bañares, and J. Ezpeleta

Department of Computer Science and Systems Engineering,
Instituto de Investigación en Ingeniería de Aragón (I3A),
University of Zaragoza. María de Luna 3, E-50015 Zaragoza (Spain)
{alvaper, banares, ezpeleta}@unizar.es

Abstract. Web service coordination and composition have become a central topic for the development of Internet-based distributed computing. A wide variety of different standards have been defined to deal with the composition of Web services (usually represented as workflows) and the execution of coordination protocols. On the other hand, some relevant research proposals have already pointed to the use of the same formalism for both aspects, being Petri nets one of the adopted formalisms. In this work we present a case study showing how the adoption of the *Nets-within-Nets* paradigm helps in the modelling of complex coordination protocols and workflows. We first propose a Petri net model for a Web service peer able to run any workflow and to dynamically interpret the coordination required protocols. The execution of these protocols allows the peer to integrate functionalities offered by external peers. The *Linda* communication model has been used to support the integration among peers.

Keywords: Service Composition and Coordination, Formal Methods for Service-Oriented Architectures, Petri nets, Nets-within-Nets paradigm.

1 Introduction

In service-oriented computing, Web services are the basic building blocks to create new applications. Many efforts have been devoted to define some standards to access Web services. As pointed in [1], some research should be done on how to weave those services together and subsequently expose the resulting artifacts as new Web services, namely, service coordination and composition (choreography and orchestration terms are also used alternatively to refer to them). The cornerstone of this style of building Web-based applications is a communication

* This work has been partially supported by the Spanish Ministry of Education and Science through the project TIC2003-09365-C02-01 from the National Plan for Scientific Research, Development and Technology Innovation.

B. Benatallah, F. Casati, and P. Traverso (Eds.): ICSOC 2005, LNCS 3826, pp. 185–197, 2005.

middleware able to glue Web services using new interaction models, more complex than the provided by the client/server model (asynchronous, event-based communication, etc.).

Service composition is an aspect related to the implementation of a Web service whose internal logic involves the invocation of operations offered by other Web services. From this definition, it is obvious that composition requires interactions between different Web services: a Web service may require specific dialogs (sequences of interchanged messages) in order to respond to a service requested by another participant. A *conversation* is a dialog among two or more Web services participating in these complex interactions, whereas a *coordination protocol* describes a set of accepted conversations (the external observable behavior of involved Web services) [1].

Most coordination and composition standard initiatives have been launched with industry-wide support. For Web service coordination, behavioral descriptions (i. e., the set of protocols and their conversations) of Web services can be defined using high-level declarative languages, such as WCSI [2], WS-CDL [3] and OWL-S [4]. On the other hand, many composition tools are available in the marketplace, most of them offering some type of modelling mechanism based on the BPEL4WS specification [5]. In any case, despite all the efforts invested in the standardization of coordination and composition languages and tools, an important problem is the lack of a clear methodology to develop complex Web services. Another interesting question refers to the fact that both, composition and coordination, have quite similar aspects, which lead us to the question of why not to use the same tool/formalism to work with them.

Different solutions can be adopted to deal with this last point, being Petri nets a quite natural approach. Petri nets [6] are a well-known formalism in the world of concurrent systems, which easily fits into Web service environments to deal with composition and coordination aspects (see for instance [7, 8, 9, 10]). The Petri net family of formalisms are of interest for the Web service community because they provide a clear and precise formal semantics, an intuitive graphical notation and many techniques and tools for their analysis, simulation and execution [7]. However, Petri nets are not the unique formalism that can be used for that purposes: in [11, 12] alternative solutions can be found using automata-based specifications of the peers' behaviors, using queues as intermediate message stores; and, in [13] is investigated the use of process-calculus techniques for providing distributed protocols in a mobile agent-based environment.

In this paper we propose a formal model based on Petri nets to represent a Web service peer able to run workflows representing composed services, to dynamically interpret the coordination protocols required by them during the execution and to communicate with other external peers via an abstraction of a communication middleware. The use of Petri nets as the same formalism to represent together conversations and workflows and the adoption of the *Nets-within-Nets paradigm* allow a natural integration of the coordination and composition models, making their interactions easier. The model imposes a methodology for avoiding the confusion between workflows and conversations, and provides a co-

ordination space based on the *Linda* paradigm [14] to model the asynchronous communications among peers. The choice of *Linda* is motivated because its communication primitives are particulary well-suited for Web service environments allowing an uncoupled communication and requiring a minimum prior knowledge between the cooperating peers.

Our approach is similar to the one in [15]. Moldt et al. follow a more agent-centric view to cope with adaptability for workflows in the Web service field. They use *Nets-within-Nets* to deal with different aspects related with Web services, such as the deployment of Web services into physical hosts, the service container, and the internal and external service flows. Our proposal provides a more concrete model based on a *Linda*-like communication model. This simpler and narrower point of view allows us to provide a more detailed representation of peers, recovering the explicit separation between protocols (workflows) and conversations of agent models presented in [16].

The paper is organized as follows. Section 2 presents a brief introduction to the *Linda* communication model and the *Nets-within-Nets* paradigm. These formalisms constitute the framework for modelling Web service composition and coordination. Section 3 introduces our view of a Web service peer able to execute complex workflows involving complex conversations, which is then applied to the development of a concrete example from [17]. Finally, Section 4 contains some concluding remarks and future work directions.

2 Underlying Technologies: *Linda* and *Nets-Within-Nets*

Let us briefly introduce *Linda* and *Nets-within-Nets* as the underlying technologies used in the approach we are proposing.

2.1 *Linda* as the Communication Model

Linda is a coordination model based on generative communication. If two or more processes need to communicate, they cooperate exchanging messages through a shared memory. In *Linda*, messages are represented as *tuples*, while the common tuple repository is called a *tuple space* [14]. Informally speaking, a tuple is a list of untyped atomic values, as `("a string", 18)`, for instance. A few simple operations have been defined to insert (withdraw) tuples into (from) the tuple space: `out` places a tuple into the tuple space; `rd` returns a copy of a tuple from the space that matches with a template tuple (a template is a query tuple composed of values and wildcards, like `("a string", ???)`; the matching is free for the wildcard and literal for the constant values); finally, `in` works like `rd`, except that the matched tuple is removed from the space. If no matching tuples are into the space, the `rd` and `in` operations block the calling process until a convenient tuple appears (until a matching occurs). In this paper, the *Linda* operations have been renamed according to the point of view of the external processes: `write` (out), `read` (rd) and `take` (in).

The use of *Linda* in distributed and open environments is promising because it allows for an uncoupled cooperation in space and time and a flexible modelling of

interactions among processes without adapting or announcing themselves. However, due to the fact that processes distributed over Internet communicate exchanging XML-encoded data, and the reading operations can involve long waits if no matching tuple is available in the space, *Linda* must be extended for improving its data representation capabilities and the set of associated operations. In this sense, the definition of tuple has been broadened to be able to represent data according to the XML encoding-format by means of attribute/value pairs, and new non-blocking reading operations inspired by an event-based communication style have been added to coordinate Web services [18].

2.2 The *Nets-within-Nets* Paradigm

Assuming the reader knows about Petri nets, let us now briefly introduce the class of Reference nets [19], which is a subclass of the *Nets-within-Nets* family of Petri nets [20]. *Nets-within-Nets* are an extension of the Colored Petri net formalism [21]. They fall into the set of object oriented approaches. In classical Petri nets, the net structure is static, and tokens move inside the net. *Nets-within-Nets* have a static part (the environment, also called *system net*) and a dynamic part, composed of instances of *object nets* that move inside the system net. These instances can be created in a dynamic way. Each object net can have its own internal dynamic behavior and can also interact with the system net by means of *interactions*. The system net can also move (*transport*) object nets by its own.

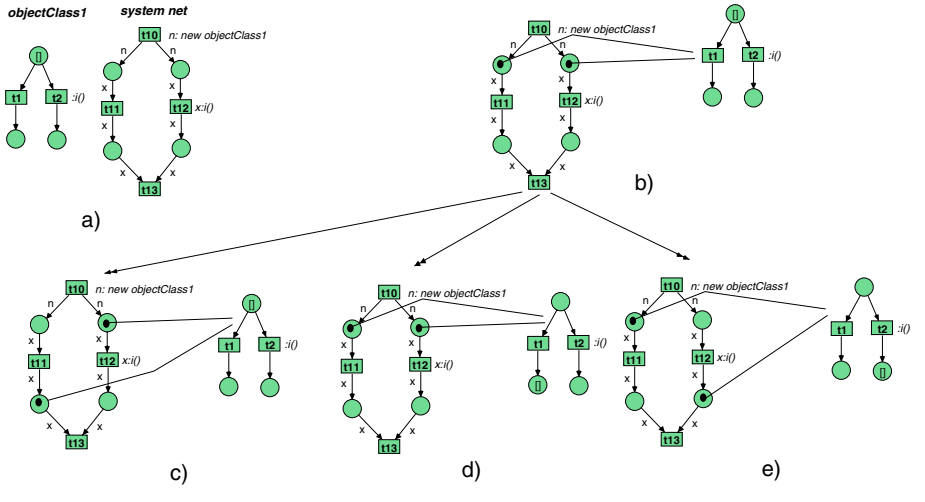

Fig. 1. a) A reference *Net-within-Net* example with the *system net* and an *object net*. b) The previous systems once transition `t10` has been fired. c) Evolution from the state in Figure-b) when `t11` fires (*transport*). d) Evolution from the state in Figure-b) when `t1` fires (*autonomous object event*). e) Evolution from the state in Figure-b) when the synchronized firing of `t12` and `t2` occurs (*interaction*).

Reference nets are a special subclass of *Nets-within-Nets* in which tokens in the system net, instead of object nets, are references to object nets, so that it is possible for different tokens to refer to the same object net. Figure 1-a) depicts a system net and an object net class. Firing transition `t10` creates two references to a new instance of `objectClass1`, moving the system to the state in Figure 1-b). In *Nets-within-Nets* three different types of transition firings are possible. The first one corresponds to the case in which an object instance executes an *object autonomous action*: in the state in Figure 1-b), transition `t1` of the object net is enabled, and can fire independently of the system net, leading to the state in Figure 1-d). The second one corresponds to the initiative of the system net: in the state in Figure 1-b), transition `t11` of the system net is enabled, and can fire moving the reference from the input place of transition `t11` to its output place, leading to the state in Figure 1-c) (notice that nothing has changed in the internal state of the object net). This is the reason why these firings are called *transports*. The last case corresponds to the synchronized firing of a transition of the system net with a transition of an object: in the state in Figure 1-b), transitions `t12` and `t2` can synchronize their firings (this is indicated by the common part in their inscriptions, `:i()`), whose firing will give the state in Figure 1-e). This way of firing is called an *interaction*. It is important to remark that the mentioned inscriptions may optionally consist of a comma-separated list of parameters (e.g., `:i(x,y,z)`), which are used to communicate values between the synchronized nets.

Reference nets have a powerful tool called `Renew` [22] that allows to execute reference nets. It is developed in Java, and allows an easy integration of reference nets and Java code associated to transitions (it is possible to access Java code from the net, but also to access the net from Java code). This makes Renew to become a very interesting and useful tool to work on Web services environments.

3 *Nets-within-Nets* for Web Services Composition and Coordination

In this section we describe the architectural design of the approach we propose. In order to introduce the way we propose to integrate Web service composition and coordination, an example from the literature is developed.

3.1 The System Components

Figure 2 depicts the Petri net model we propose for a Web service peer. Basically, a Web service peer has the following elements. First, it contains a *work-space*. This is a kind of process space where the services in which the peer is involved are being executed. These can be either simple ("atomic" services) or composite services. Composite services are described by means of workflows where a component is a service provided by either an external peer or the peer itself. On the other hand, the interactions between the peer and other peers usually require the execution of (simple or complex) interaction protocols (choreographies) whose

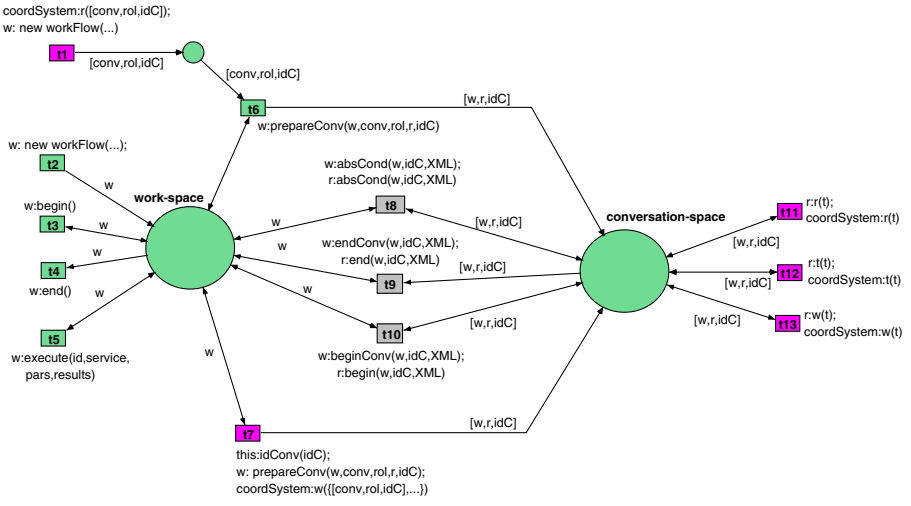

Fig. 2. The general architecture of a Web service peer

possible executions correspond to possible conversations. Conversations may involve two or more peers and, in each conversation, each peer has to execute a given part (the peer must play a given "role"). The set of roles that a peer is playing at a given moment stay in the *conversation-space*.

Adopting the *Nets-within-Nets* paradigm, Figure 2 corresponds to the "system net". Place *work-space* corresponds to the work space as described in the precedent paragraph, while place *conversation-space* contains the active roles in which the peer is involved in. Tokens inside these places will be Petri nets (*object nets* in *Nets-within-Nets* terminology) corresponding to workflows and roles in execution, respectively. Let us first take a closer look at the system net. Notice that transitions `t1` and `t2` both contain the `new workFlow(...)` action; these transitions are the way of starting the execution of new workflows. The main difference is that `t2` corresponds to the case of a new workflow generated by the initiative of the peer itself, while transition `t1` corresponds to the case of a workflow started in response to the requirement of a service initiated by another peer

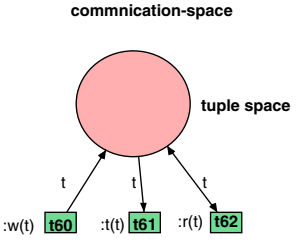

Fig. 3. A *Linda* coordination system

(accepting to participate in a conversation demanded by another peer requires to be able to execute a given role of a conversation).

A workflow whose execution has started can execute a set of different actions, corresponding to transitions around the work-space place:

- The workflow must execute a `begin` action (transition `t3`) as a first step after creation and, once terminated, it must execute an `end` action (transition `t4`).
- The workflow may require the execution of inner services, this is the task of transition `t5`. For that, a call to a local service or application is generated with a set of input and output arguments. For instance, once a set of data has been received, some local processing can be necessary before continuing an active conversation.
- Transitions `t6` and `t7` are the ones that generate new conversations, which are inserted into the conversation space. There are two different situations in which insertion of new conversations can occur (in fact, new roles corresponding to either a new conversation or an existing one): 1) the workflow requires the new conversation and starts it (transition `t7`); this means that a new conversation correlator `idC` is generated, a role (or set of roles) is assumed by the initiating workflow and a set of peers are demanded putting the corresponding service demands on the coordination system; 2) the workflow decides to meet the requirements of another peer to participate in an started conversation, in which a required role is assumed (transition `t6`).
- Once a given in-execution workflow starts the execution of a role of a conversation, some interactions are needed between the workflow and the conversation (these typically involve the necessity of passing information between the workflow and the role in order to be able to execute the conversation). They must synchronize the conversation begin and end points (transitions `t10` and `t9`, respectively). On the other hand, the execution of a role of a given conversation by a peer sometimes requires the invocation of proper services, whose results may also be needed to continue the conversation (for instance, to do some calculations, to take some decisions, etc.). This approach is similar, but more flexible, to *abstract properties* (WSCI notation [2]) or *variables* (WS-CDL notation [3]) used by declarative XML-based coordination languages for the evaluation of conditions against the internal implementation of the service. This is the task of transition `t8`.

Obviously, Web services must interact, usually in an asynchronous way. This means that some mechanism must be provided allowing Web services to interact. Its asynchronous nature made interesting to adopt the *Linda* coordination system [14]. In fact, independently of its implementation, a *Linda* system will be considered along all the paper. Figure 3 is a model of a *Linda* coordination space. The three transitions `t60,t61,t62` correspond to the `write`, `take` and `read` *Linda* operations, respectively, while place *tuple space* holds the tuples inserted into the *Linda* space. The peer will execute the communication operations by firing transitions `t11,t12,t13`, which will synchronize with `t62,t61,t60`, respectively.

Workflows and conversations have some common elements, which pushed us to model both using the same formalism. Among other common elements, the following can be considered: a) there may have some partial ordering to be imposed (on the way services are composed and also on the way messages are exchanged); b) the same way as some services can be satisfied in parallel, a peer, inside a given conversation, can dialog with a set of involved peers (even if the different dialogs would be executed in an interleaved way). This leads to the use of Petri nets for both types of elements. Since these elements must live and evolve inside a peer, to model them by means of object nets in the *Nets-within-Nets* paradigm is a quite natural approach. Let us introduce in the next subsection an example and to explain how it can be viewed from our perspective.

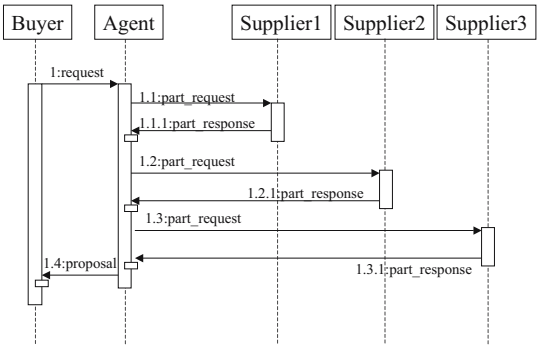

Fig. 4. The case study sequence considered in [17]

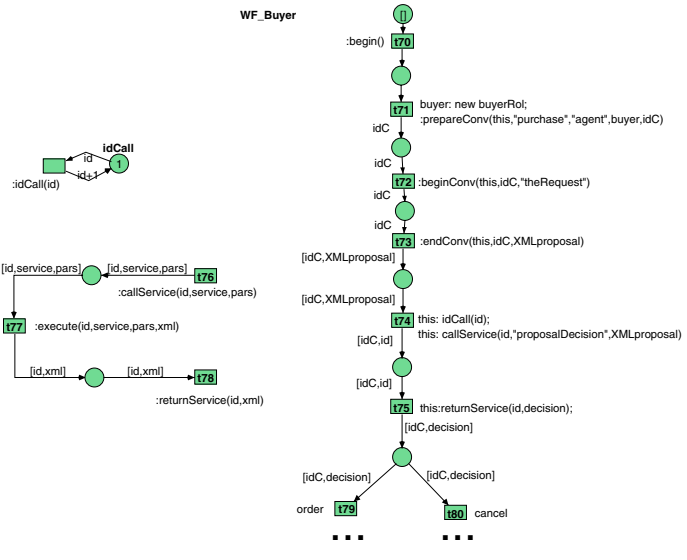

Fig. 5. The (partial) workflow of the buyer peer

3.2 An Example

Let us use the same case study in [17] to show how Petri nets can be used for composition and coordination, and how the *Nets-within-Nets* paradigm allows a natural integration with the previously defined architecture. The case corresponds to a PC manufacturer which needs to build a set of PC machines with different configurations, and using a list of available component suppliers. In the process, a buyer uses a purchasing agent to fulfill the inventory requests. The purchasing agent communicates with a set of suppliers, each of them offering specific components needed to build the PC machines. Once a complete configuration can be build (using components of one or multiple suppliers), a proposal is constructed and sent to the buyer, which can either place the parts order or cancel the request. Figure 4 shows a possible view of the process just described.

Figure 5 is a Petri net model of the workflow a buyer executes (by now, do not pay attention to the left small Petri nets in the figure: they are just technical elements to get a unique local identifier for local service calls, the upper one, and to implement the local service calls, the bottom one). Firing transition `t2` in Figure 2 generates a new instance of this workflow, which is inserted into the work-space place. The synchronized firing of transitions `t3` and `t70` makes the execution of the workflow to start. The synchronized firing of transitions `t7` and `t71` makes a lot of work. First, a net instance of the role `"buyer"` (left part of

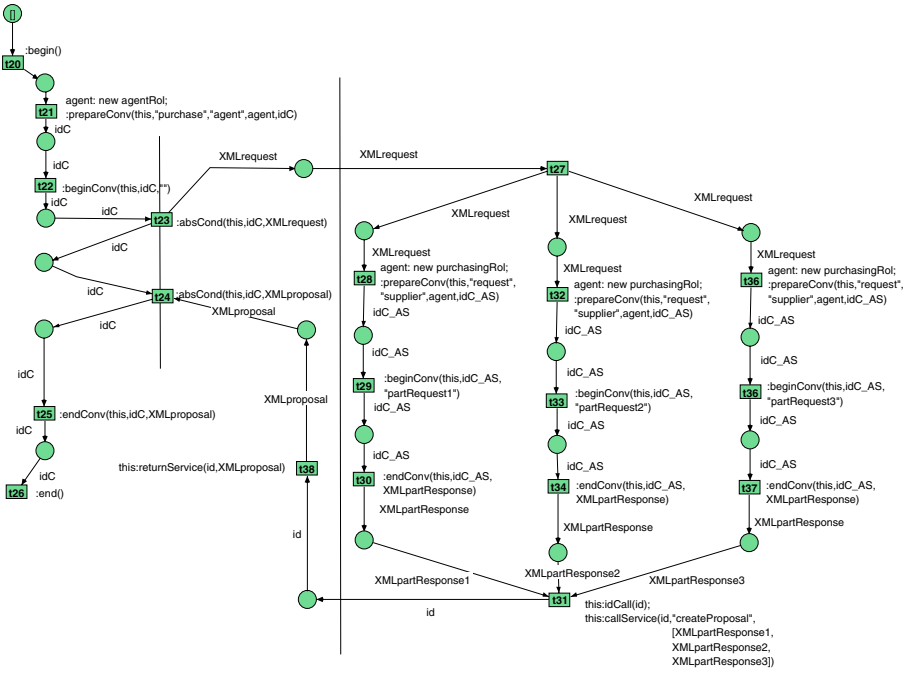

Fig. 6. The workflow of the agent peer

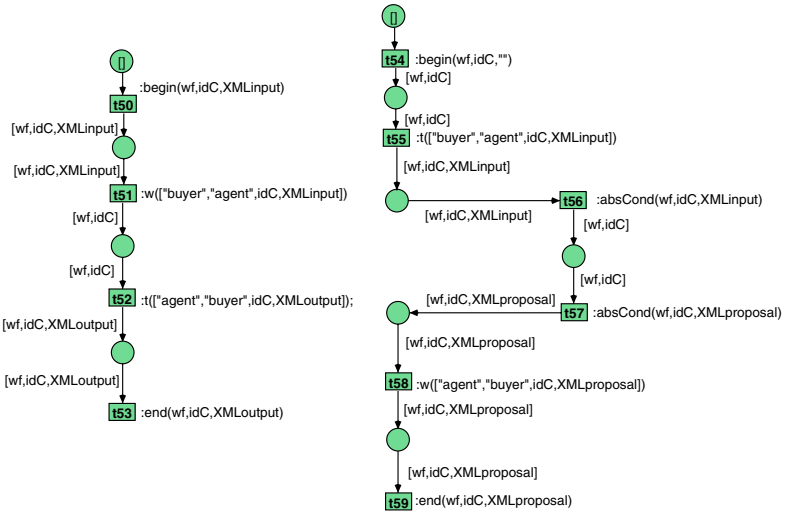

Fig. 7. A PN model of the buyer-agent interaction protocol (the `"purchase"` protocol). The left part corresponds to the `"buyer"` role, while the right one to the `"agent"` role.

Figure 7) of the coordination protocol `"purchase"` is created (the `buyer: new buyerRol` is executed); then the coordination protocol is prepared (a new correlator, `idC`, is generated and associated to the pair `(workflow,coordination protocol)`). The generated instance of the buyer role in the coordination protocol is then inserted into the conversation space. Finally, a request of the form `["purchase","agent",idC]` is sent to the coordination system, which means that a peer playing the `"agent"` role of a coordination protocol of type `"purchase"` is looked for. Firing transition `t72` starts the execution of the `buyer` role of the coordination protocol.

On the other side, Figure 6 is a Petri net model of the workflow an agent must execute as the answer to the request just commented. Notice that firing transition `t21` prepares an instance of the `agentRol`; the conversation between the buyer and the agent takes place while both roles are executing. Figure 7 is the communication protocol where transitions `t51`,`t52`, `t55` and `t58` execute asynchronous communication operations on the *Linda*-like asynchronous communication system.

Let us now concentrate on the agent workflow to remark some important elements. Once the instance of the agent role net has been created, the synchronized firing of transitions `t10`, `t22` and `t54` is possible, which means that the agent part of the `"purchase"` conversation can start. When possible, the synchronized firing of `t9`, `t25` and `t59` will terminate that conversation. But previously, a lot of things must occur. The synchronized firing of `t8`, `t23` and `t56` allows the agent role part of the `"purchase"` coordination protocol to pass to the associated workflow, in form of an XML string/file, for instance, the description of the request. With this information, that workflow initiates, in parallel, a

set of `request` coordination protocols with the three suppliers (transition `t27`). Once the offers from the suppliers arrive (firing transition `t31`), the local service *createProposal* is called, using the set of offers as parameter. By firing transition `t24`, the workflow of the agent peer communicates the proposal elaborated from the set of offers to its role (`"agent"` role) of the current conversation. The mentioned proposal is sent to the buyer by firing transition `t58`. The conversation between the buyer and the agent is then terminated, as well as the agent workflow (synchronized firing of transitions `t9`, `t25` and `t59`).

4 Conclusions

Most of the XML-based languages for modelling coordination protocols and workflows are declarative and cannot, by itself, be executed. Moreover, different languages are used as appropriate. In this work, Petri nets and the *Nets-within-Nets* paradigm are used as the same formalism to deal with complex coordination protocols and workflows. Its intuitive graphical notation, its formal semantics and the possibility of doing some properties analysis offer interesting advantages.

We have proposed a Petri net model for a Web service peer able to execute workflows and dynamically interpret the required coordination protocols. This proposal allows the development of complex Web based systems by the description of the workflows and protocols of the involved services, and subsequently, their distributed execution over a set of physical hosts. The cooperation among these distributed peers has been done by a *Linda*-like communication model, which is orthogonal to the formalism in which it is embedded. The decision of using a more concrete communication model, and a more concrete and delimited view of a Web services architecture which, as a counterpart, looses generality with respect to the proposal in [15], distinguishes our proposal from the approach followed by Moldt et al.

At present, we have also implemented *Linda* as a Petri net of the class of the *Nets-within-Nets*, so that we are able, using the Renew tool, to execute Web service peers and *Linda*-like coordination primitives on distributed physical hosts. We will study, in further work, the incorporation of some new functionalities to use Renew as a well-adapted tool for the modelling and prototyping of Web Services. It should include some important aspects as the automatic translation OWL-S into Petri nets, as pointed in [16], the addition of some horizontal coordination protocols, and the inclusion of transaction processing or time related aspects, as timeouts, for instance.

References

1. Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web Services. Concepts, Architectures and Applications. Springer Verlag (2004)
2. A. Arkin et al.: Web Service Choreography Interface (WSCI). Technical report, World Wide Web Consortium (W3C) (2002)

3. N. Kavantzas et al.: Web Service Choreography Description Language (WS-CDL). Technical report, World Wide Web Consortium (W3C) (2004)
4. D. Martin et al.: Bringing Semantics to Web Services: The OWL-S Approach. Number 3387 in Lecture Notes in Computer Science. In: First International Workshop, SWSWPC 2004. Revised Selected Papers. Springer Verlag (2004) 26–42
5. T. Andrews et al.: Business Process Execution Language for Web Services (BPEL4WS). Technical report, BEA Systems & IBM & Microsoft & SAP AG & Siebel Systems (2003)
6. Murata, T.: Petri nets: Properties, analysis and applications. In: Proceedings of IEEE. Volume 77. (1989) 541–580
7. Aalst, W., Hee, K.: Workflow Management: Models, Methods, and Systems. MIT Press, Cambridge, MA, USA (2004)
8. M. Mecella, F.P. Presicce, B.P.: Modeling E-Service Orchestration Through Petri Nets. Number 2444 in Lecture Notes in Computer Science. In: Proceedings of the 3rd VLDB International Workshop on Technologies for e-Services (VLDB-TES 2002). Springer Verlag (2002) 38–47
9. Hamadi, R., Benatallah, B.: A Petri net-based model for Web service composition. In: CRPITS'17: Proceedings of the Fourteenth Australasian database conference on Database technologies 2003, Darlinghurst, Australia, Australia, Australian Computer Society, Inc. (2003) 191–200
10. Yi, X., Kochut, K.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 Symposium (DASD'04), Advanced Simulation Technology Conference 2004. (2004) 141–148
11. Hull, R., Benedikt, M., Christophides, V., Su, J.: E-services: a look behind the curtain. In: PODS '03: Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, New York, NY, USA, ACM Press (2003) 1–14
12. Fu, X., Bultan, T., Su, J.: Analysis of interacting BPEL Web services. In: WWW '04: Proceedings of the 13th international conference on World Wide Web, New York, NY, USA, ACM Press (2004) 621–630
13. Fournet, C., Gonthier, G., Lévy, J.J., Maranget, L., Rémy, D.: A calculus of mobile agent. In: Proc. of CONCUR'96. Volume 1119 of Lecture Notes in Computer Science. Springer-Verlag, Berlin (1996) 406–42
14. Carriero, N., Gelernter, D.: Linda in context. Communications of the ACM **32** (1989) 444–458
15. Moldt, D., Offermann, S., Ortmann, J.: Proposal for Petri Net Based Web Service Application Modeling. Number 3140 in Lecture Notes in Computer Science. In: Web Engineering: 4th International Conference, ICWE 2004. Springer Verlag (2004) 93–97
16. Moldt, D., Ortmann, J.: A Conceptual and Practical Framework for Web-based Processes. Unpublished manuscript (2004)
17. Peltz, C.: Web Service Orchestration and Choreography. A look at WSCI and BPEL4WS. Web Services Journal (2003) 1–5
18. Álvarez, P., Bañares, J.A., Muro-Medrano, P.: An Architectural Pattern to Extend the Interaction Model between Web-Services: The Location-Based Service Context. Number 2910 in Lecture Notes in Computer Science. In: First International Conference on Service Oriented Computing –ICSOC 2003. Springer Verlag (2003) 271–286
19. Kummer, O.: Introduction to Petri Nets and Reference Nets. Sozionik Aktuell (**1**)

20. Valk, R.: Petri nets as token objects - an introduction to elementary object nets. Lecture Notes in Computer Science: 19th Int. Conf. on Application and Theory of Petri Nets, ICATPN'98, Lisbon, Portugal, June 1998 **1420** (1998) 1–25
21. Jensen, K.: Colored Petri nets: A high level language for system design and analysis. In Rozenberg, G., ed.: Advances in Petri Nets 1990. Volume 483 of Lecture Notes in Computer Science. Springer Verlag, Berlin (1991) 342–416
22. Kummer, O., Wienberg, F.: Renew - the reference net workshop. In: Tool Demonstrations, 21st International Conference on Application and Theory of Petri Nets, Computer Science Department, Aarhus University, Aarhus, Denmark (2000) 87–89