A Petri-Net Based Formalisation of Interaction Protocols Applied to Business Process Integration

Djamel Benmerzoug[1], Fabrice Kordon[2], and Mahmoud Boufaida[1]

[1] LIRE Laboratory, Computer Science Department,
Mentouri University of Constantine 25000, Algeria
{benmerzougdj,boufaida_mahmoud}@yahoo.fr
[2] LIP6 Laboratory, Pierre et Marie Curie University,
4, place Jussieu, 75252 Paris Cedex 05 France
fabrice.kordon@lip6.fr

Abstract. This paper presents a new approach for Business Process Integration based on Interaction Protocols. It enables both integration and collaboration of autonomous and distributed business processes modules. We present a semantic formalisation of the interaction protocols notations used in our approach. The semantics and its application are described on the basis of translation rules to Coloured Petri Nets and the benefits of formalisation are shown. The verified and validated interaction protocols specification is exploited afterwards with an intermediate agent called « Integrator Agent » to enact the integration process and to manage it efficiently in all steps of composition and monitoring.

Keywords: Business Processes Integration, Interaction Protocols, Coloured Petri Nets, Multi-agent Systems.

1 Introduction

Unlike traditional business processes, processes in open, Web-based settings typically involve complex interactions among autonomous, heterogeneous business partners. In such environments there is a clear need for advanced business applications to coordinate multiple business processes into a multi-step business transaction. This requires that several business operations or processes attain transactional properties reflecting business semantics, which are to be treated as a single logical unit of work [1]. This orientation requires distilling from the structure of businesses collaboration the key capabilities that must necessarily be present in a Business Process Integration (BPI) scenario and specifying them accurately and independently from any specific implementation mechanisms.

Web services are a promising technology to support business processes coordination and collaboration [2][3]. They are an XML-based middleware that provides RPC-like remote communication, using in most cases SOAP over HTTP. Web services are designed to allow machine-to-machine interactions. This interaction takes place over a network, such as the Internet, so Web services are by definition distributed, and operate in an open and highly dynamic environment.

Heterogeneity, distribution, openness, highly dynamic interactions, are some among the key characteristics of another emerging technology, that of intelligent agents and

J.L.G. Dietz et al. (Eds.): CIAO! 2008 and EOMAS 2008, LNBIP 10, pp. 78–92, 2008.

Multi-Agent Systems (MAS). M. Luck et al. [4] propose the following definition: "an agent is a computer system that is capable of flexible autonomous action in dynamic, unpredictable, typically multi-agent domains."

We already proposed a new approach based on Web services and agents for integrating business processes [5]. The BPI modeling is based on Interaction Protocols (IP) that enable autonomous, distributed business process management modules to integrate and collaborate.

IP are a useful way for structuring communicative interaction among business process management modules, by organizing messages into relevant contexts and providing a common guide to all parties. The value of IP-based approach is largely determined by the interaction model it uses. The presence of an underlying formal model supports the use of structured design techniques and formal analysis and verification, facilitating development, composition and reuse.

Most IP modeling projects to date have used or extended finite state machines (FSM) and state transition diagram (STD) in various ways [8]. FSM and STD are simple, depict the flow of action/communication in an intuitive way, and are sufficient for many sequential of interactions. However, they are note adequately expressive to model more complex interactions, especially those with some degree of concurrency. In the other hand, Coloured Petri Nets (CPN) [9] are a well known and established model of concurrency, and can support the expression of a greater range of interactions. In addition, CPN like FSM, have an intuitive graphical representation, are relatively simple to implement, and are accompanied with a variety of techniques and tools for formal analysis and design.

Unfortunately, the existing works on the use of formal models to represent IP leave open several questions [8], [16], [19], [21]. Most previous investigations have not provided a systematic comprehensive coverage of all issues that arise when representing complex protocols such as intra-Enterprise Application Integration (EAI) as well as the inter-enterprise integration (B2B, for Business to Business).

This paper presents a generic approach for the BPI based on interaction protocols. Translation rules of IP based on AUML/BPEL4WS [13],[14] notations into CPN are proposed, enabling their formal analysis and verification. We provide interactions building blocks allowing this translation to model complex e-business applications that enable autonomous, distributed business process management modules to integrate and collaborate.

This CPN-based representation can be used to essentially cover all the features used in IP standards, including communicative act attributes (such as message guards and cardinalities) and protocol nesting. Also, we present a skeleton automated procedure for converting an IP specification to an equivalent CPN, and demonstrate its use through a case study.

In the next section we, briefly present our approach. Section 3 describes a CPN based representation of IP. In section 4, we provide a skeletal algorithm for converting BPI based on interaction protocols in AUML/BPEL4WS to Coloured Petri nets. Section 5 shows how the verified and the validated IP specification can be exploited by the MAS to enact the BPI. Related work is discussed in section 6 and conclusions are drawn in section 7.

2 An Overview of the Proposed Approach

In recent years, BPI modeling and reengineering have been longstanding activities in many companies. Most internal processes have been streamlined and optimized, whereas the external processes have only recently become the focus of business analysts and IT middleware providers. The static integration of inter-enterprise processes as common in past years can no longer meet the new requirements of customer orientation, flexibility and dynamics of cooperation [10].

In [6],[7] we have developed an agent-based method for developing cooperative enterprises information systems. This method permits to explicitly map the business process into software agents. In [5], we have described the use of IP to define and manage public processes in B2B relationships. This process is modelled using AUML (Agent UML [13]) and specified with BPEL4WS [14].

In this approach, we consider two types of business processes, the private processes and the public ones. The first type is considered as the set of processes of the company itself and they are managed in an autonomous way. Private processes are supported within companies using traditional Workflow Management Systems, Enterprise Resources Planning systems or proprietary systems. These systems were intended to serve local needs. In other hand, public processes span organizational boundaries. They belong to the companies involved in a B2B relationship and have to be agreed and jointly managed by the partners.

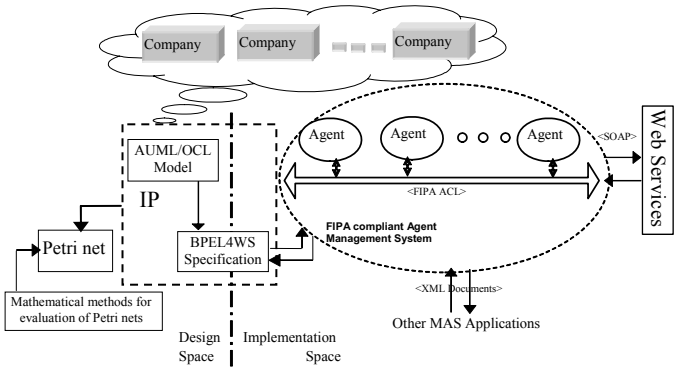


Fig. 1. The proposed approach

The B2B integration scenarios typically involve distributed business processes that are autonomous to some degree. Companies participating in this scenario publish and implement a public process. The applications integration based on public process is not a new approach. The current models for BPI are based on process flow graphs [11], [12]. A process flow graph is used to represent the public process. This approach lacks the flexibility for supporting dynamic B2B integration. In contrast, our approach (figure 1) presents an incremental, open-ended, dynamic, and personalizable model for B2B integration.

The use of IP to define public processes enables a greater autonomy of companies because each company hides its internal activities, services and decisions required to support public processes. In this way, the IP provide a high abstraction level in the modelling of public processes. The AUML model is mapped to a BPEL4WS specification, which represents the initial social order upon a collection of agents (figure 1). Since BPEL4WS describes the relationships between the Web services in the public process, agents representing the Web services would know their relationships a priori. Notably, the relationships between the Web services in the public process are embedded in the process logic of the BPEL4WS specification.

This relationship entails consistency problems, which can at best be solved at the level of models. Indeed, we used the BPEL4WS specification to generate a validation tool that can check that a BPEL4WS document is well-formed (the BPEL4WS preserves the business constraints, which are specified by means of OCL (Object Constraint Language [23])). In this work, we have exploited the Sun Microsystem Web Services Developer Pack [15]. In particular, we have used the JAXB (Java Architecture for XML Binding) library to build Java classes from a BPEL4WS specification (for more detail see [5]).

In this paper, we address the problem of verification of BPI based on interaction protocols. Indeed, we propose a novel and flexible representation of protocols that uses CPN in which, interaction building blocks explicitly denote joint conversation states and messages. So, interaction protocols specification can be translated to an equivalent CPN model and CPN tools can afterwards be used to analyze the process.

3 A CPN-Based Model for BPI Based on Interaction Protocol

BPI is defined as an interaction protocol involving different companies. It specifies the interaction between local business process and Web services and their coordination. For this purpose, we define the IP as follow:

Definition: An Interaction Protocol is a quadruplet: IP = <ID, R, M, f_M>, where:

- ID is the identify of the interaction protocol
- R = {r_1, r_2, …, r_n} (n>1) is a set of Roles (private business process or Web Services)
- M is a set of non-empty primitive (or/and) complex messages, where:
  - A Primitive Message (PM) corresponds to the simple message, it is defined as follow: PM = <Sender, Receiver, CA, Option>, where:
    - Sender, Receiver ∈ R
    - CA ∈ FIPA ACL (**C**ommunicative **A**ct such as: cfp, inform, …)
    - Option: contain additional information (Synchronous / Asynchronous message, constraints on message, …)
  - A Complex Message (CM) is built from simpler (primitive) ones by means of operators: CM = PM_1 op PM_2 … op PM_m.where:
    - m>1, op ∈ {OR, XOR, AND}, and
    - ∀ i ∈ [1, m[, PM_i.Sender = PM_{i+1}.Sender, PM_i.Sender ∈ R .
- f_M: a flow relation defined as : $f_M \subseteq$ (RxR), where (RxR) is a Cartesian product (r1,r2) ∈ (RxR), for r1,r2 ∈ R

Ideally, IP should be represented in a way that allows performance analysis, validation and verification, automated monitoring, debugging, etc. Various formalisms have been proposed for such purposes. However, Petri nets have been shown to offer significant advantages in representing IP, compared to other approaches [16]. Specifically, Petri nets are useful in validation and testing, automated debugging and monitoring and dynamic interpretation of IP.

Our main motivation in describing the semantics of IP applied to BPI by using CPN is that the existence of several variation points allows different semantic interpretations that might be required in different application domains. This is usually our case, and so, high-level Petri nets are used as formal specification. This provides the following advantages:

- CPN provide true concurrency semantics by means of the step concept, i.e. when at least two non-conflictive transitions may occur at the same time. It is the ideal situation for our application domain (several activities moving within the same space of states: the <flow> section in BPEL4WS).
- The combination of states, activities, decisions, primitives and complex message exchanges (namely fork-join constructions) means that the IP notations are very rich. CPN allow us to express, in the same formalism, both the kind of system we are dealing with and its execution.
- Formal semantic is better in order to carry out a complete and highly automated analysis for the system being designed.

3.1 Translation Rules from IP Elements to CPN

The objective of this section is to propose some general rules which may be applied to formally specify interaction protocols endowing them with a formal semantics. Such a semantics will enable the designer to validate his/her specifications. As shown in the translation rules in Table 1, we focus on the description of dynamic aspects of protocols using the CPN's elements (places, transitions, arcs, functions, variables and domains).

The CPN representation in Table 1 introduces the use of token colours to represent additional information about business processes interaction states and communicative acts of the corresponding interaction. The token colour sets are defined in the net declaration as follow: (the syntax follows standard CPN-notation [9])

```
Colour sets :
Communicative Act = with inform|cfp|propose|… ;
Role = string with "a".. "z" ; // Role = {r1, r2, …}, ri ∈ R
Content = string with "a".. "z" ;
Bool = with true|false;
MSG = record s,r: Role; CA: Communicative Act; C: Content
Variables:msg, msg1, msg2: MSG;      x: Bool;
```

The *MSG* colour set describes communicative acts interaction and is associated with the net message places. The *MSG's* coloured token is a record <s,r,ca,c>, where the *s* and *r* elements determine the sender and the receiver of the corresponding message. This elements have the colour set *ROLE*, which is used to identify business processes or/and Web services participating in the corresponding interaction. The *COMMUNICATIVE ACT* and the *CONTENT* colour sets represent respectively the FIPA-ACL communicative acts and the content of the corresponding message. We note that

places without colour set hold an indistinguishable token and therefore have the colour domain token = {●}.

We now show how various interaction protocols features described in our work can be represented using the CPN formalism.

R1: *A role* (the <partner> section in BPEL4WS) is considered equivalent to a type of resource, which is represented in a Petri net as a place. Hence, there will be one token in the place for each actor playing this role. Each one of these places is labelled with the corresponding role name.

R2: *The "life line" of role* is represented implicitly by a places and transitions sequence belonging to this role. The net is constituted therefore by one sub-net (Petri net process) for each role acting during the interaction and these nets are connected by places that correspond to the exchanged messages.

R3: *A message exchange* between two roles is represented by a synchronization place and arcs. The first ongoing arc connects the transition of "message sending" to the "synchronization place", while the second outgoing arc connects this place to the "receiving message transition".

R4: *A primitive message exchange:* As we have already said, a primitive message corresponds to the simple message. A <receive> and <reply> activities (asynchronous messages) are represented by a transition which has an in-place and out-place (see R3 in Table 1). An <invoke> activity (synchronous messages) is represented by a pair of transitions, one of them may fire a request token to the sub-net of the receiver role, and the other may wait for a token from this sub-net.

R5: *A complex message exchange:* A complex message is represented by a substitution transition. The control flow between messages exchange is captured by connecting the activity-related transitions with arcs, places, and transitions purely used for control flow purpose. More refined control flow can be expressed using arc inscriptions and transition guard expressions.

Table 1 (R5 – (1)) shows a more complex interaction, called XOR-decision. (the <if>/<pick> section in the BPEL4WS specification) so that only one communicative act can be sent. In this case, each type of message is associated to a transition with a function on its input arc. The function plays the role of a filter, i.e. it control the firing of the transition corresponding to the message type. Table 1 (R5 – (2)) shows another complex interaction, the OR-parallel interaction (the <switch> section), in which the sender can send zero, one or more communicative acts (inclusively) to the designated recipients simulating an inclusive-or.

The last type of complex message is the *AND*-parallel (the <follow> section) which models concurrency messages sending. This type of complex interaction is represented by means of parallel case or multi-threading in CPN.

R6: *Iteration:* An iteration in a part of IP specification is represented by an arrow and a guard expression or an end condition (the <while> section in BPEL4WS). In CPN, an iteration is specified in the same way except that the end condition is a guard expression associated with the transition that starts the iteration.

Table 1. A Translation Rules From IP to CPN

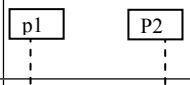
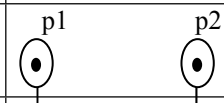
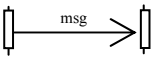
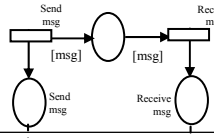
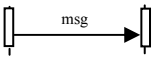
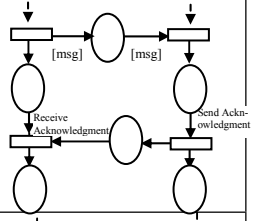
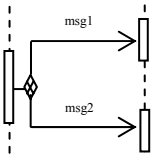
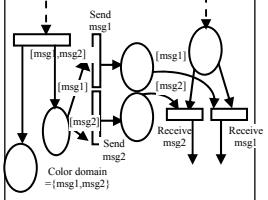
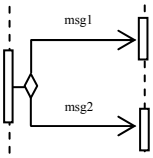
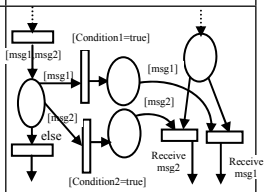
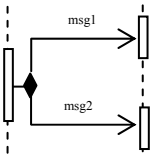
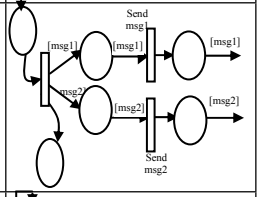
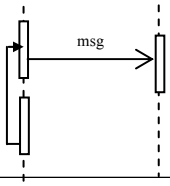
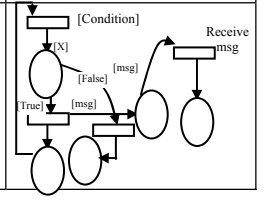
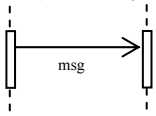
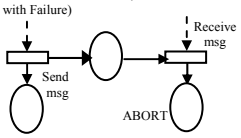
	AUML elements	**BPEL4WS elements**	**CPN elements**
R1: Roles/Web services		<process> <partners> <partner name="p1"/> <partner name="P2"/> </partners>	
R2: Role life line			
R3: message exchange (asynchronous messages)		<sequence> <receive name="msg" partner="p2" …… </receive>	
R4: primitive message exchange (synchronous messages)		<invoke name="p2" partner="P2" inputVariable="Request" outputVariable= "Result"> …… </invoke>	
R5 (1): complex message exchange (the XOR-Decision)		<if condition= "Bool-Exp"> <reply name="msg1"> …… </reply> <reply name="msg2"> …… </reply> </if>	
R5(2): complex message exchange (the OR-Decision)		<switch standard-attributes> <case condition1> <reply name="msg1"> …… </reply> </case> <case condition2> <reply name="msg2"> …… </reply> </case> <otherwise> …… </otherwise> </switch>	
R5(3): complex message exchange (the AND-Decisiosn)		<flow> <reply name="msg1"> …… </reply> <reply name="msg2"> …… </reply> </flow>	
R6: Iteration		<while condition= "Bool-Exp"> <receive name="msg" partner="p2" …… </while>	

Table 1. (*continued*)

R7: Case of termination	msg.CA∈{failure, cancel,, Refuse, not-understood } 		ABORT is a final state (end of Interaction with Failure) 

R7: *Case of termination:* In the specification of the FIPA-ContractNetProtocol besides the AUML diagram other requirements are described in the text [13]: The sending of not-understood messages and the so called FIPA-Cancel-Metaprotocol: Every received message is responded to by a not-understood, if the comprehension of the message failed. In this case, the protocol is cancelled for the corresponding participant. In a CPN, this is realized by adding a transition to the final state ABORT (except the initial state). This transition corresponds to the reception of acts: Failure, Cancel, Refuse or not-understood, which can terminate the IP with failure.

3.2 An Algorithm for Transforming an IP to Its CPN Representation

Previous investigations have explored various machine-readable Petri net representations. However, interaction protocols are typically specified in human-readable form (e.g., in AUML [13]). The question of how to automatically translate an interaction protocol specification into a machine-readable form has been previously ignored [16]. We present an automated procedure for transforming an IP to its CPN representation.

The algorithm is presented in figure 2. It inputs an IP as defined in section 3, and it outputs a corresponding CPN representation. The CPN is constructed by iterating: The algorithm essentially creates the IP-net by exploring the interaction protocol. Lines 1 and 2 initiate different variables used in this algorithm and respectively the CPN output. The roles places, denoted by the variable RP, hold the initiating places for the Petri net. These places correspond to the roles of the IP (line 3, 4 and 5). Each one of these places is labelled with the corresponding role name.

We enter the main loop in line 7 and set *curr* to the first message in the IP. Lines 8-16 create the CPN components of the current iteration. First, in line 8, message places, associated with *curr* role place, are created using *CreateMessagePlace*.

These places correspond to communicative acts. Then, in line 9, we create intermediate places that correspond to interaction state changes as a result of these messages associated with *curr* place. Then, in *CreateTransitions* and *CreateArcs*, these places are connected through transitions and arcs, using the CPN building blocks previously described (section 3). Finally, we add token elements colour to the CPN structure, implementing attributes using the *FixColor* function (line 16).

To complete the iteration, the CPN output, is updated according to the current iteration in lines 17-19. The loop iterates as long as *M* contains messages that have not been handled. Finally, the resulting CPN is returned (line 21).

Algorithm CreateIP-net (**input** : IP=<ID, R, M, f_M>, **output** : CPN)

1: RP←Ø // Roles places
MP← Ø // Messages places
IM← Ø // Intermediate places
TR← Ø // list of transitions
AR← Ø // list of arcs
2: CPN ← new CPN
3: **For** every r ∈ R **do**
4. RP←*createRolePlace()* // there would be one token in every RP place
5: CPN.places ← RP
6: **While** M ≠ Ø **do**
7: *curr* ← M.*dequeue()*
8: MP← *CreateMessagePlace(curr)*
9: IM ← *CreateIntermediatePlace(curr,MP)*
10: TR← *CreateTransitions(curr,MP,IM)*

11: **If** curr.CA ∉ {Failure, Cancel, Refuse, not-understood}
12: AR← *CreateArcs(curr,MP,IM,TR)*
13: **Else** // MP is a terminating place
14: AR← *CreateArcs(curr, IM,TR)*
15: **End If**

16: *FixColor(MP,TR,AR,curr.CA)*

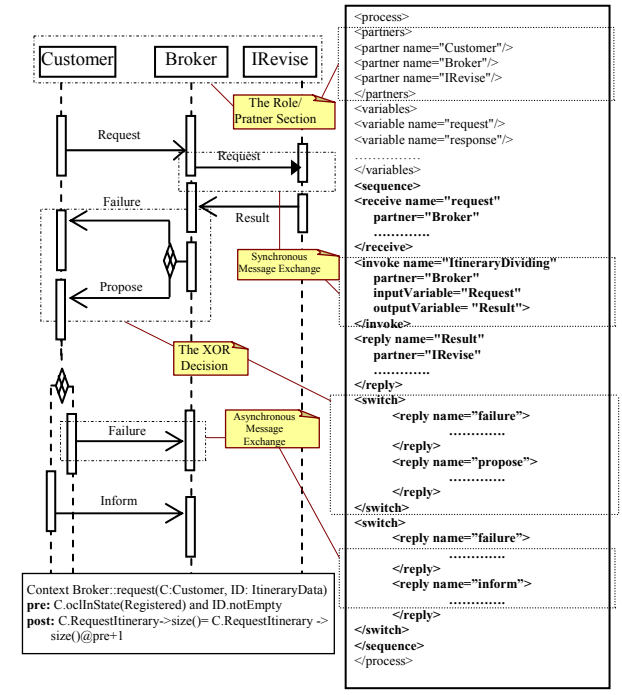
17: CPN.places ← CPN.places ∪ MP ∪ IM
18: CPN.transitions ← CPN.transitions ∪ TR
19: CPN.arcs ← CPN.arcs ∪ AR
20: **End while**
21: **Return** CPN

Fig. 2. IP to CPN Conversion Procedure

4 A Case Study: The Agent-Based Transportation e-Market System

To illustrate this algorithm, we use it to construct a CPN of a part of our example presented in [7] (shown as IP in figure 3). This example illustrates the interaction among three parts: Customer, Broker and IRevise, where the two first parts are Interfaces of different business systems, and the last part is an automatic service. In this protocol, the process starts when the Customer role sends a message with business information: `request(ItineraryData)`. Once the Broker receives these messages, the Web service IRevise is invoked for reviewing the customer itinerary and divide this itinerary into sub-itineraries.

We note that all the private processes are not defined by the interaction protocol because they are private aspects of the Broker. After dividing the itinerary, the Broker decides whether to send a message `propose(ItineraryPlan)` to the Customer or refuse the customer request because it cannot be satisfied. This is defined with a logical connector XOR, which represents that only one of the two alternative messages can be



a- the AUML Interaction Protocol b- the BPEL4WS of (a)

Fig. 3. An Interaction Protocol as AUML/BPEL4WS

sent. In this case, the Customer has two interaction threads that represent the incoming messages. When the Customer receives a message `propose(ItineraryPlan)`, he can accept this itinerary plan or can declare a failure during the negotiation because consensus has not been achieved.

We now use the algorithm introduced above (fig. 2) to create a CPN for this IP. The algorithm begins with the creation of three Roles Places (RP) initially marked (one place for every role/partner in the IP: lines 3 and 4). Line 5 permits to update the CPN with the *RP* variable. In the first iteration of the main loop (line 7), the *curr* variable is set to the first message in the IP (*curr* ← <*"Customer", "Broker", "request", "S"*>). The algorithm creates net places, which are associated with the *curr* variable, i.e. a request message place (line 8) and two places in the Customer and respectively the Broker sub-nets (the `CreateIntermediatePlace()` function at line 9).

These three places (see the resulting CPN in Figure 4) are connected using the asynchronous message building block shown in Table 1. The MP is not a terminating place (the Customer is waiting for a response from the Broker) and is thus connected through transitions and arcs with the `CreateTransitions()` and `CreateArcs()` functions (lines 10, 11, 12). Next, the colour sets of the corresponding places are determined (colour domains of the transitions are generally defined according to the domains of the results of functions evaluation of input arcs).

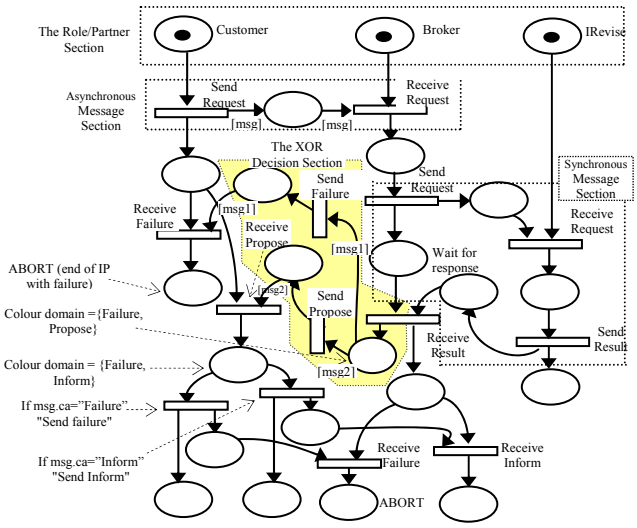


Fig. 4. The Resulting CPN of the IP presented in Fig. 3

In the second iteration, *curr* is set to *<" Broker", "Customer", "failure", "A">⊕ <" Broker", "Customer", "propose", "A">*. In this case, the Broker can send either a failure or a propose messages, and thus appropriate message places are created using the XOR-decision building block shown in Table 1. Then, two places, corresponding to the results of the messages are created. These places are connected using the XOR-decision described in Table 1. This building block involves the creation of the guard conditions on the transitions controlling the firing of the transition corresponding to the message type (which is represented as a colour in the Petri net).

In this iteration, we note that the MP place corresponding to the message "failure" is a terminating place, so no outgoing transitions or arcs are creating from this place. The loop iterates as long as M contains messages that have not been handled. Finally, the resulting CPN is returned (Figure 4).

5 Validation and Property Verification

CPN allow us to validate and evaluate the usability of a system by performing automatic and/or guided executions. These simulation techniques can also carry out performance analysis by calculating transaction throughputs, etc. Moreover, by applying other analysis techniques it is possible to verify static and dynamic properties in order to provide the complement to the simulation. Some of these properties are that:

- There are no activities in the system that cannot be realized (dead transitions). If initially dead transitions exist, then the system was bad designed.
- The IP specification exhibits the liveness property (e.g., the output CPN guarantees the existence of an initial state such that for any accessible state, at least one operation is executed).

– It is always possible to return to a previous state (home properties). For instance, to compare the results of applying different decisions from the same state. (the case of XOR and OR decision)
– The system may stop before completion (deadlock). Thus, a work might never be finished, or it might be necessary to allocate more resources to perform it.
– Certain tokens are never destroyed (conservation). Hence, resources are maintained in the system.

6 Enabling Integration Process with Multi-Agent Systems

As we already have said, the BPEL4WS process specification is considered as a language for specifying the interaction protocol of multi-agents system. In this section we briefly describe how the MAS use the verified and validated BPEL4WS specification to establish the BPI. Our suggestion consists in the addition of a specific agent between the MAS application and its IP parts conceived as Web services (see figure 5). The main advantage of this approach is the integration completeness property inherent from our BPEL4WS specification. Integration completeness means that the IP is itself published and accessed as a Web service that can participate in other application integration. Since applications integration is often viewed as a hierarchy of different local systems and services, the integration completeness property permits the agent-based integration to be included via BPEL4WS into other applications integration definitions.

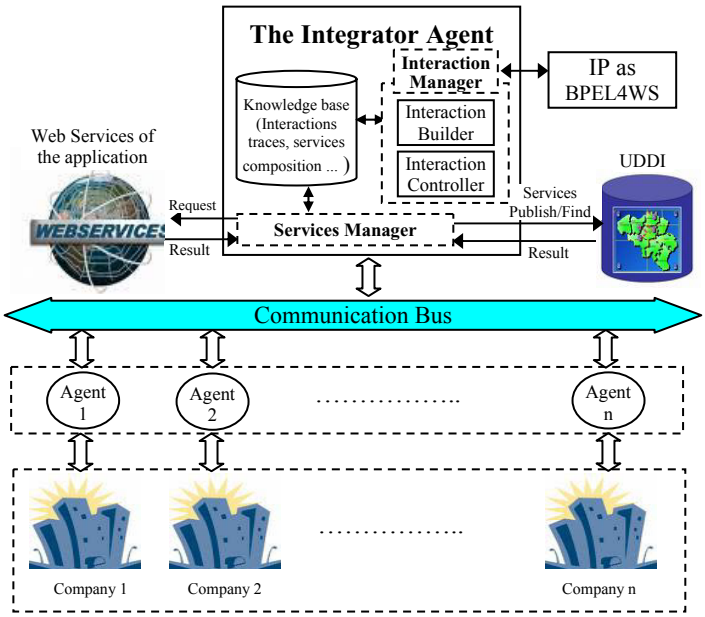


Fig. 5. Global Structure of our Architecture

As shown in Figure 5, the BPEL4WS specification is exploited thereafter with an intermediate agent called «Integrator Agent ». This integration must keep as much as possible the autonomy of architecture core based on agents. Indeed, The agents are coordinated with the Integrator agent and the exchange of messages to enact the BPI. In this architecture, the following communication pathways exist:

- agent to agent communication occurs via FIPA ACL and is facilitate by a FIPA compliant Agent Management System.
- agent to Web service communication is accomplished via SOAP messages.
- agent to BPEL4WS dataspace communication uses appropriate protocols/interfaces provided by the dataspace. The dataspace is used to store BPEL4WS process variables, which maintain the state of the IP.

The main roles of the Integrator agent are the creation, monitoring, and control of IP life cycle. It's architecture features two modules: an interaction manager and a service manager. The interaction manager contains operational knowledge (e.g., Interactions states). It also provides operations for monitoring interactions (i.e., creating and deleting instances). The service manager provides methods for receiving service requests, tracing service executions, and communicating with service requesters in accordance with IP definition (e.g., sending a notification informing the requester that deadline for cancelling an operation is passed).

7 Related Work

BPI and automation is an active research domain. The community is still debating the issues of enterprises collaboration at the business process level.

In [17], P. Buhler et al. summarize the relationship between agents and Web services with the aphorism "Adaptive Workflow Engines = Web Services + Agents": namely, Web services provide the computational resources and agents provide the coordination framework. They propose the use of the BPEL4WS language as a specification language for expressing the initial social order of the multi-agent system. P. Buhler et al. does not provide any design issues to ensure the correctness of their interaction protocols.

In [21], authors propose translating rules for the conversation of an interaction protocol given in AUML to CPN. Unfortunately, no procedures were provided that guide the conversion of an interaction protocol given in AUML to Petri net representations.

The Symphony project [18] has developed an algorithm for analyzing a composite service specification for data and control dependences and partitioning it into a set of smaller components. These components are then distributed to different locations and, when deployed, cooperatively deliver the same semantics as the original workflow. Symphony does not provide any support for failures arising from workflow mismatches since it assumes that the distributed processes will be derived from a single complete BPEL process.

Several other approaches aim to solve the integration problem by emphasizing interaction protocols. The state transition diagram (STD) has been extensively used for IP specification due to its clarity. The weakness is that it does not reflect the

asynchronous character of the underlying communication [19]. Furthermore, it is not easy to represent integration of protocols. The Dooley Graph [20] is an alternative formalism for visualizing agent inter-relationships within a conversation. Object-oriented methods like UML [22] offer a way to reduce the gap between users and analyst when considering message transfers, yet they only address the dynamic behavior of individual objects and are informal.

Compared with the related work, our approach allows us to provide a clear separation of inter-enterprise collaboration management and local business process management, to make full use of existing workflow system components, to support both public processes and private business processes. Another advantage of our approach is the integration completeness property inherent from our BPEL4WS specification. It means that the IP is itself published and accessed as a Web service that can participate in other application integration. Since applications integration is often viewed as a hierarchy of different local systems and services, the integration completeness property allows agent-based integration to be incorporated via BPEL4WS into other applications integration definitions.

8 Conclusion and Future Work

In this paper, we presented a generic approach for BPI based on interaction protocols. The proposed translation rules from AUML/BPEL4WS notations to Coloured Petri nets enable the use of many verification techniques during the design phase to detect errors as early as possible.

Such translation allows to easily model complex e-business applications. We also proposed an automated procedure for converting an interaction protocol specification to a corresponding coloured Petri nets and illustrated its use through a case study.

The verified and validated interaction protocols specification is exploited afterwards with an intermediate agent called Integrator Agent to enact the integration process and to manage it efficiently in all steps of composition and monitoring.

Our primary future work direction is the exploitation of the BPEL4WS specified BPI by the Integrator agent to facilitate the creation, monitoring, and control of interaction life cycle at run-time. We will also introduce the notion of intelligence; we will try to specify all the cooperative agents of our architecture as intelligent and autonomous Web components.

References

1. Papazoglou, M.P., Kratz, B.: Web Services Technology in Support of Business Transactions. Int. journal of Service Oriented Computing 1(1), 51–63 (2007)
2. Jung, J.Y., Kang, S.H.: Business Process Choreography for B2B Collaboration. IEEE Internet Computing, 37–45 (2004)
3. Aissi, S., Malu, P., Srinivasan, K.: E-business process modeling: the next big step. IEEE Computer, 55–62 (2002)
4. Luck, M., McBurney, P., Shehory, O., Willmott, S.: The AgentLink Community: Agent Technology: Computing as Interaction - A Roadmap for Agent-Based Computing. AgentLink III (2005)

5. Benmerzoug, D., Boufaida, M., Kordon, F.: A Specification and Validation Approach for Business Process Integration Based on Web Services and Agents. In: Int. Workshop on Modeling, Simulation, Verification and Validation of Enterprises Information Systems (MSVVEIS 2007), pp. 163–168. INSTICC press (2007)
6. Benmerzoug, D., Boufaida, Z., Boufaida, M.: From the Analysis of Cooperation Within Organizational Environments to the Design of Cooperative Information Systems: An Agent-Based Approach. In: Meersman, R., et al. (eds.) OTM Workshops 2004. LNCS, pp. 496–506. Springer, Heidelberg (2004)
7. Benmerzoug, D., Boufaida, M., Boufaida, Z.: Developing Cooperative Information Agent-Based Systems with the AMCIS Methodology. In: IEEE International Conference on Advances in Intelligent Systems: Theories and Application, Luxembourg (2004)
8. Cost, R., Chen, Y., Finin, T., Labrou, Y., Peng, Y.: Using Colored Petri nets for Conversation Modeling. In: Dignum, F., Greaves, M. (eds.) Issues in Agent Communication. LNCS (LNAI), vol. 1916, pp. 178–192. Springer, Heidelberg (2000)
9. Girault, C., Valk, R.: Petri Nets for Systems Engineering - A Guide to Modeling, Verification, and Applications. Springer, Heidelberg (2003)
10. Koehler, J., Tirenni, G., Kumaran, S.: From Business Process Model to Consistent Implementation: A Case for Formal Verification Methods. In: Pro. of the Sixth International Enterprise Distributed Object Computing Conference, IEEE Computer Society, Los Alamitos (2002)
11. Peregrine B2B Integration Platform, http://www.peregrine.com
12. Thatte, S.: XLANG: Web Services for Business Process Design, Microsoft Corp., cf (2001), http://www.gotdotnet.com/team/xml_wsspecs/
13. Huget, M., Odell, J.: Representing agent interaction protocols with agent UML. In: 3rd International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 1244–1245. IEEE Computer Society, Los Alamitos (2004)
14. Business Process Execution Language for Web Services Version 1.1 (2003), http://www-106.ibm.com/developerworks/
15. Sun Microsystems. Java Web Services Development Pack 1.1 (2006), http://java.sun.com/webservices/webservicespack.html/
16. Gutnik, G., Kaminka, G.A.: A Scalable Petri Net Representation of Interaction Protocols for Overhearing. In: Kudenko, D., Kazakov, D., Alonso, E. (eds.) AAMAS 2004. LNCS (LNAI), vol. 3394, pp. 1246–1247. Springer, Heidelberg (2005)
17. Buhler, P.A., Vidal, J.M.: Towards adaptive workflow enactment using multiagent systems. Int. Jour. On Information Technology and Management, 61–87 (2005)
18. Chafle, G., Chandra, S., Mann, V., Nanda, M.: Decentralized Orchestration of Composite Web Services. In: Proc. of the Alternate Track on Web Services at the 13th International World Wide Web Conference (WWW 2004), pp. 134–143 (2004)
19. Martial, F.: Coordinating Plans of Autonomous Agents. LNCS (LNAI), vol. 610. Springer, Heidelberg (1992)
20. Parunak, H.V.D.: Visualizing Agent Conversations: Using Enhanced Dooley Graphs for Agent Design and Analysis. In: Proceedings of the International Conference on Multi-Agent Systems (1996)
21. Mazouzi, H., Fallah-Seghrouchni, A.E., Haddad, S.: Open Protocol Design for Complex Interactions in Multi-Agent Systems. In: Proceedings of AAMAS 2002, pp. 517–526 (2002)
22. Booch, G., Rumbaugh, J., Jacobson, I.: The unified modeling language for object-oriented development. Document set version 1.0, Rational Software Corporation, Santa Clara (1997)
23. OMG; Object Constraint Language Specification, http://www.omg.org/cgi-bin/doc?formal/03-03-13