Software Agent Architecture for Managing Inter-Organizational Collaborations

E. Tello-Leal*[1], O. Chiotti[2] and P.D. Villarreal[3]

[1] Facultad de Ingeniería y Ciencias
Universidad Autónoma de Tamaulipas
Victoria, Tamaulipas, México
*etello@uat.edu.mx
[2] INGAR – Instituto de Desarrollo y Diseño
CONICET-UTN, Consejo Nacional de Investigaciones Científicas y Técnicas
Santa Fe, Argentina
[3] Centro de Investigación y Desarrollo de Ingeniería en Sistemas de Información (CIDISI)
Facultad Regional Santa Fe, Universidad Tecnológica Nacional
Santa Fe, Argentina

ABSTRACT
The growing importance of cooperation among organizations, as a result of globalization, current market opportunities and technological advances, encourages organizations to dynamically establish inter-organizational collaborations. These collaborations are carried out by executing collaborative business processes among the organizations. In this work we propose an agent-based software architecture for managing inter-organizational collaborations. Two types of agents are provided: the Collaboration Administrator Agent and the Process Administrator Agent. The former allows organizations setting up collaborations. The latter allows organizations executing collaborative business processes. A Colored Petri Net model specifying the role, which an organization fulfills in a collaborative process, is used to carry out the behavior of the Process Administrator Agent that represents the organization. Planning and execution of the actions of the Process Administrator Agents are driven by a Colored Petri Net machine embedded to them. Thus, Process Administrator Agents do not require to have defined at design-time the protocols they can support. In addition, we propose a model-driven development method for generating Colored Petri Net models from a collaborative process model defined as interaction protocol. Finally, an implementation of the agent-based software architecture and methods based on model-driven development are presented.

Keywords: Software agent, inter-organizational collaboration, Model-Driven Development, collaborative business process, BPMN.

RESUMEN
La creciente importancia de la cooperación entre las organizaciones, como consecuencia de la globalización, las oportunidades actuales de mercado y los avances tecnológicos, alienta a las organizaciones a establecer en forma dinámica colaboraciones inter-organizacionales. Estas colaboraciones se llevan a cabo mediante la ejecución de procesos de negocio colaborativos entre las organizaciones. En este trabajo de investigación se propone una arquitectura basada en agentes de software para la gestión de colaboraciones inter-organizacionales. La arquitectura provee dos tipos de agentes: el Agente Administrador de Colaboraciones y el Agente Administrador de Proceso. El primer agente permite a las organizaciones a establecer colaboraciones. El segundo agente habilita a las organizaciones ejecutar procesos de negocio colaborativos. El rol que una organización desempeña en un proceso colaborativo es especificado mediante un modelo de redes de Petri coloreadas. Este modelo es usado para dirigir el comportamiento del Agente Administrador de Proceso, el cual representa a una organización. La ejecución de los planes y las acciones del Agente Administrador de Proceso son dirigidas mediante una máquina de redes de Petri coloreadas embebida en el agente. Entonces, los Agentes Administrador de Proceso no requieren tener definido en tiempo de diseño los protocolos que dan soporte a su comportamiento. Adicionalmente, se propone un método basado en el desarrollo dirigido por modelos para la generación en forma automática de modelos de redes de Petri coloreadas a partir de un modelo de procesos de negocio colaborativo definido como protocolo de interacción. Finalmente, la implementación de la arquitectura y los métodos basados en el desarrollo dirigido por modelos son presentados.

1. Introduction

The growing importance of cooperation among organizations, as a result of globalization, current market opportunities and technological advances, encourages organizations to dynamically establish inter-organizational collaborations. An inter-organizational collaboration entails a process-oriented integration between heterogeneous and autonomous organizations which must be achieved at a business level and at a technological level [1]. Inter-organizational collaborations are carried out through the execution of collaborative business processes. A collaborative business process (CBP) defines the global view of the behavior of the interactions among enterprises to achieve common business goals [1, 2]. The design and implementation of CBPs implies new challenges, such as participants' autonomy, decentralized management, peer-to-peer interactions, negotiation, and alignment between the business solution and the technological solution [1, 3]. In order to maintain the organizations' autonomy, it is required a decentralized management of the CBPs, which can be achieved through distributed and synchronized implementation of the integration business processes of each involved organizations. An integration process specifies the public and private behavior that supports the role an organization performs in a CBP. It contains, from the viewpoint of an organization, the public and private logics required to process or generate the information exchanged with its partners.

Therefore, the software applications must be developed in a manner they can interoperate effectively in this new distributed, heterogeneous, and sometimes, unreliable environment. Software agent technology is seen as a potentially robust and scalable approach to meet this challenge. Since the features of software agents such as autonomy, heterogeneity, decentralization, coordination and social interactions are also desirable for organizations involved in inter-organizational collaborations [1], the use of this technology can be considered as appropriate to be used within this domain [4, 5]. Software agents that execute CBPs can help to improve process integration, interoperability, reusability and adaptability [6, 7, 8].

In this work we propose an agent-based software architecture for managing inter-organizational collaborations. This architecture consists of two types of agents that represent organizations. Collaboration administrator agents provide the functionality to establish a collaboration agreement among organizations. Process administrator agents support the execution of collaborative business processes that organizations agreed to carry out in a collaboration agreement. The role an organization fulfills in a collaborative process is defined in a Colored Petri Net (CP-Net) model, which is used to drive the behavior of the process administrator agent representing the organization. The planning and execution of the actions of process administrator agents are driven by a CP-Net machine embedded in them. Thus, the behavior of the process administrator agents is derived in run-time. In addition, we propose a Model-Driven Development (MDD) [9] method for the automatic generation of CP-Net models from collaborative process models. CP-Net models of the process administrator agents are derived from collaborative process models, which are described as business interaction protocols defined with the UML Profile for Collaborative Business Processes based on Interaction Protocols (UP-ColBPIP) language solution [1, 10]. Hence, interaction protocols representing collaborative processes are executed by process administrator agents without the need for protocols defined at design-time in these agents. Finally, as a proof of concept, we also present an implementation of the agent-based architecture for inter-organizational collaborations.

The paper is organized as follows. Section 2 describes the agents that compose agent-based architecture for inter-organizational collaborations. Section 3 describes the MDD-based method for generating CP-Net models. Section 4 describes an implementation of the proposed agent-based architecture. Section 5 shows an application example of the architecture and its implementation. Section 6 includes related work and, finally, Section 7 presents conclusions and future work.

2. Agent-based architecture for inter-organizational collaborations

In this section we present an agent-based architecture for inter-organizational collaboration.

Two main functionalities are offered by this architecture: support for managing dynamic collaboration agreements among organizations; capabilities for executing and monitoring CBPs that organizations agree to carry out. The proposed architecture is composed of two types of agents supporting the above functionalities (see Figure 1):

• Collaboration Administrator Agent (CAAgent), which represents an organization participating of an inter-organizational collaboration. It is responsible for setting up communications with other CAAgents in order to agree on the CBPs to be performed. This is also responsible for instantiating PAAgents that execute the CBPs.

• Process Administrator Agent (PAAgent), which executes the role an organization fulfills in a CBP. It is responsible for the jointly execution of a CBP along with the PAAgents of the other organizations involved in the CBP. At any time, an organization will have a PAAgent for each execution of a CBP in which it participates.

Relations between these agents are shown in Figure 2, which depicts an Agent-Object-Relationship (AOR) diagram [11]. This diagram defines the conceptual relations between the agents and the resources used by those agents represented as objects. Agents are connected among them through a communication relation to achieve their goals. The diagram depicts the basic architecture required to establish an inter-organizational collaboration between two organizations and how they can execute CBPs in order to achieve common business goals.

A CAAgent can communicate with another CAAgent for establishing collaboration to execute CBPs. The CAAgent that initiates the communication is responsible for distributing CBP models, which are received by the other CAAgent. This communication is carried out by these agents through the protocol defined in section 2.1.

The CBP models are defined with the UP-ColBPIP language. This language supports the definition of the behavior of CBPs through the modeling of business interaction protocols [1, 10]. In addition UP-ColBPIP extends the semantics of UML2 Interactions to model interaction protocols in UML2 Sequence Diagrams. An interaction protocol describes a high-level communication pattern through business messages composed by speech acts and business documents [1, 10]. Speech acts (request, agree, refuse, etc.) provide semantics to business messages, which allows defining complex negotiations and avoiding ambiguous understanding of the messages.

When the CAAgents have agreed on a collaboration and have shared the models of the CBPs to be executed, they generates the CP-Net models automatically, and instantiate the corresponding PAAgents. A PAAgent of an organization is instantiated by a CAAgent of the same organization when a CBP has to be executed. The CAAgent assigns a CP-Net model to the PAAgent containing the behavior of the role the organization fulfills in the CBP which agreed to execute. The CP-Net models of the PAAgents are generated from the UP-ColBPIP model that defines the CBP to be executed.

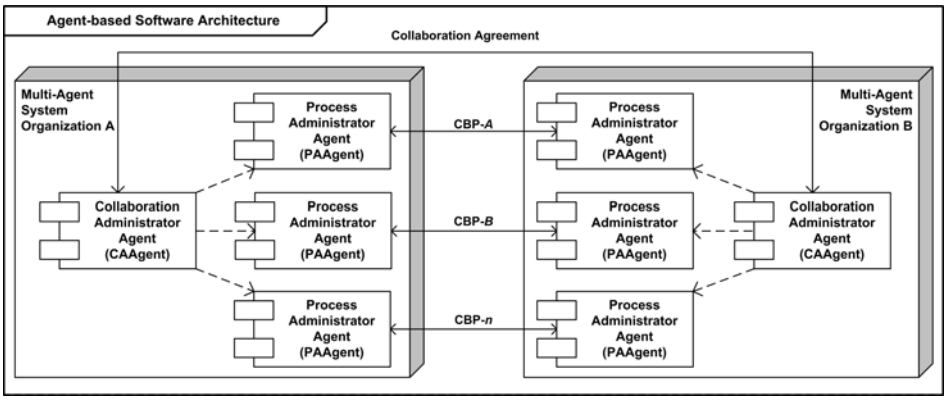

Figure1. Agent-based architecture for inter-organizational collaborations.

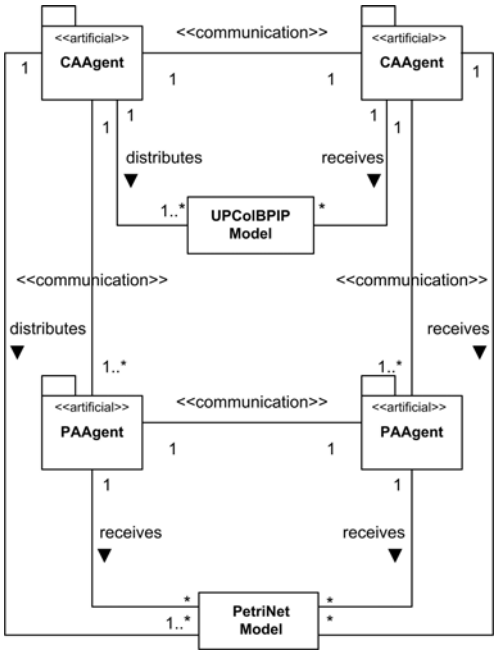

Figure 2. Conceptual AOR diagram for the agent-based software architecture.

Hence, PAAgents communicate between them to carry out the decentralized and jointly execution of a CBP. The interactions among them are governed by their CP-Net models and are carried out according to the UP-ColBPIP interaction protocol that describes the logic of the CBP being executed.

2.1 Collaboration Administrator Agent

The organizations use CAAgents for setting up collaboration agreements and generating new PAAgent instances. These can also be used to stop the PAAgent execution and to remove it. The CAAgent initiates a conversation by executing a predefined protocol called *Request for Collaboration*. Figure 3 shows the interaction protocol, which was defined with the UP-ColBPIP language.

In the behavior defined in Figure 3, the CAAgent performing the initiator role starts this protocol by sending a *request* for collaboration to the responder role.

The responder role is performed by the CAAgent of the other organization that receives the request. This means that a CAAgent can play any role depending on the way the organization engages in collaboration. The request sent by the *initiator* conveys a *CollaborationRequest* document containing the UP-ColBPIP model of the CBP that the *initiator* proposes to jointly execute.

The *responder* can send an *agree* message or a *refuse* message. In the last case the protocol ends. When the *initiator* receives an *agree* message, it generates a CP-Net model with the behavior of the role that *initiator* fulfills in the CBP. The CP-Net model is generated through the execution of the transformation engine, which is described in the Section 3.

Then the *initiator* creates an instance of a PAAgent and assigns to it the generated CP-Net model. Afterwards, the *initiator* sends an *inform* message indicating that the collaboration was successfully initiated. Alternatively if it cannot instantiate the PAAgent, sends a *failure* message.

Once received the notification of an initiated collaboration by the *initiator*, the *responder* also generates a CP-Net model for the role the *responder* fulfills in the CBP. Then the *responder* instantiates a PAAgent and assigns to it the generated CP-Net model. If it was successful, it sends an *inform* message indicating the collaboration was defined.

Otherwise, it sends a *failure* message indicating the collaboration was not defined. In both cases the protocol ends.

Thus, by executing this protocol CAAgents dynamically define an agreement to execute a CBP and instantiates the PAAgents that will execute the CBP. To create the CP-Net model required by the PAAgents, the CAAgents implement a method that enables automatically generating CP-Net models of the agents from an UP-ColBPIP model (this method is explained in Section 3).

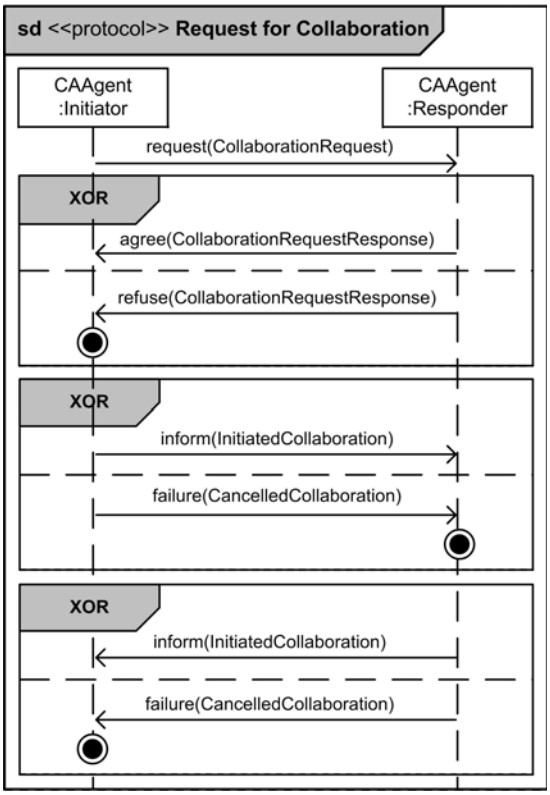

Figure 3. Request for collaboration interaction protocol.

2.2 Process Administrator Agent

The PAAgent executes a process model known as *integration process*, which contains both public and private activities needed to support the role [1, 9, 4]. Thus, through the enactment of integration processes, PAAgents of the organizations jointly execute in a decentralized manner the activities of the CBP agreement. The integration process is defined as a CP-Net model generated from the UP-ColBPIP model of the CBP. Petri Nets are particularly suited for modeling the behavior of concurrent systems in terms of control flow or flow of objects or information [12]. Some advantages when choosing Colored Petri Nets for executing integration processes are as follows [13]: Petri Nets have formal semantics, which enables an unambiguous execution and simulation of process models. Unlike some event-based process modeling notations, such as dataflow diagrams, Petri Nets can model both states and events. Besides, there are many analysis techniques for Petri Nets, which makes it possible to identify deadlocks, proper completion, absence of dead tasks, and safety issues.

A PAAgent has an embedded Colored Petri Net based process machine (see Figure 4) that interprets CP-Net models to performs the organization's role in a CBP and interact with another PAAgent to execute the CBP. The transitions defined in a CP-Net model represent the PAAgent's actions that support the message exchange defined in the interaction protocol of the CBP being executed. The order in which these actions should be planned and executed is defined by the process logic contained in the CP-Net model. Therefore, the behavior and actions of this agent are driven by the process machine according to a CP-Net model. In the execution of the CP-Net model, the process engine invokes the behavior action manager component (see Figure 4) for planning an action when a transition is enabled and for executing an action when the transition is triggered.

PAAgents have only generic actions defined at design-time. They do not require to have implemented the behavior of predefined interaction protocols. The transitions of CP-Net models represent the actions the agent has to execute. Such actions are used to carry out the behavior of a PAAgent according to the transition type to execute in the CP-Net model.

The types of transitions represent both public and private activities. The former are represented by the transitions send and receive, which support the sending and the receiving of a message, respectively. The latter are composed of the transitions *generate* and process. A generate transition represents a private action that generates the required information to be sent, previous to a send transition. A process transition represents a private action that process the information previously received. A *process* transition may be specialized in other types of transitions, such as invoke, verify or evaluate, which represent specific actions for processing a received message.

When a send transition is fired by the process machine, this transition activates a send action of the PAAgent, which carries out the sending of a

message. When a receive transition is enabled, a blocking action of the agent is invoked, which waits for the reception of a message. The receive transition is fired when a message is received, which implies the activation of the receive action of the agent. If a generate transition is triggered, this activates a generate action, consisting in generating certain required information to be sent. The process action is activated by the process transition when this is triggered. The process action consists of an internal process of evaluation of the business document previously received, after of enactment a receive action. Therefore, the concrete actions of these agents are obtained at run-time according to the logic defined in CP-Net models. This enables PAAgents to execute any interaction protocol representing the CBP that organizations want to execute as part of a dynamic collaboration agreement.

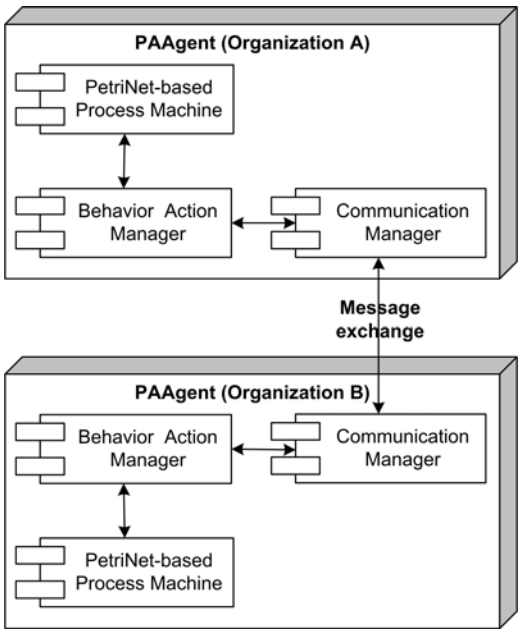

Figure 4. Process administrator agent components.

Sending and receiving a message to/from another agent is carried out by the *communication* manager component of the PAAgent. This component is *invoked* when the *behavior action manager executes a send or a receive action* (see Figure 4). In the execution of a CP-Net model, to coordinate the exchanged information between the agents the tokens maintain information about the process (e.g.: process ID) and the business documents which are sent, received, or generated internally. The transitions are also allowed to have a guard with a Boolean expression. When a guard exists, it must evaluate to true to enable the transition, otherwise the transition is disabled and cannot occur [14]. A guard puts an extra constraint on the enabling of a transition.

3. MDD-based method to generate Colored Petri Net models

In order to automate the generation of CP-Net models we propose a method based on the Model-Driven Development (MDD) [9, 15]. This method provides a transformation process to generate CP-Net models that PAAgents require to execute CBPs. The MDD-based method takes as inputs a UP-ColBPIP model that contains a CBP to be executed, and a role an organization will perform in the CBP (see Figure 5). With these inputs, the transformation process generates as output a CP-Net model for an organization, according to the target role the organization will fulfill in the CBP. This method is applied for generating the CP-Net models of all the organizations that want to execute a CBP. Thus, the behavior required for the agents to execute a CBP is generated in an automatic and agile way.

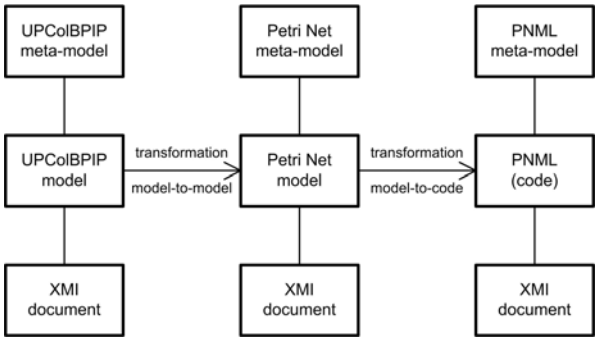

Figure 5. Structure of the UP-ColBPIP to Colored Petri Net transformation tool.

The transformation process consists in analyzing each element of an interaction protocol from the viewpoint of a target role, and generating the corresponding elements of the CP-Net that represent the protocol's element. The public logic of the control flow, i.e. the ordering of transitions

and places is derived according to transformation rules for control flow segments, which are similar to those defined in [16] for generating BPMN models. These rules define how a control flow segment of a protocol is transformed in transitions in the CP-Net, to express control flows such as exclusive gateway, loop, parallel gateway, and synchronization.

The transformation of a business message of a protocol requires generating a send or receive transition, according to the role for which the CP-Net model is generated. Also, this transformation requires generating a generate transition previous to a send transition or a process transition after a receive transition. This is carried out by the following transformation rule for a business message. This rule indicates the transitions and places to be generated for the sender of the message and the receiver of the message, as follows.

• Sender: two transitions are added. The first one is the send transition, which is added along with one input place and one output place. The send transition is labeled as: Send + Speech Act + Business Document name, being the speech act and the business document those defined in the message of the protocol. The second one is the generate transition that is added and connected to one input place and one output place, which is also the input place of the send transition. This indicates that previous to the sending of a message, *a generate* transition is performed for producing the business document to be sent. The generate transition is labeled as: Generate + Business Document name, being the business document that one defined in the message of a protocol.

• Receiver: two transitions are added. First one is the receive transition with one input place and one output place. The receive transition is labeled as Receive + Request + Business Document name, being the speech act and the business document those defined in the message of the protocol. The second one is the process transition, which has one output place and has as input place the output place of the receive transition. This indicates that after the execution of the receive transition, a process transition is executed. The process transition is labeled as: Process + Business Document name, being the business document that one defined in the message of a protocol.

Figure 6 shows a transformation rule defined with the ATL transformation language, which from an UP-ColBPIP model allows generating a transition generate and a transition send in a Petri Net model, for each business document with a request or propose speech acts content in an UP-ColBPIP model. Transformation rule is the basic construct in ATL used to express the transformation logic [17]. The transformation rule presented in the Figure 6 is specified in a declarative style.

```
rule sendBusinessMessage2SendTransition {
	from
		businessMessage: MMupcolbpip!BusinessMessage (
			businessMessage.isSenderRequiredRole('Customer') and
			(businessMessage.isRequest or businessMessage.isPropose)
		)
	to
		transition1: MMPetriNet!Transition (
			location <- 'abstractTask_' + businessMessage.id,
			name <- 'Generate ' + businessMessage.information.name
		),
		Transition2: MMPetriNet!Transition (
			location <- 'sendTask_' + businessMessage.id,
			name <- 'Send ' + businessMessage.intention + ' ' + businessMessage.information.name
		)

}
```

Figure 6. Transformation rules to generate send and generate transitions (CP-Net) from business message (UP-ColBPIP) with the request or propose speech act.

3.1 Tools for the MDD-based method

A set of tools were developed in order to aid organizations to apply this methodology. They were built on the Eclipse open development platform [18] in order to take advantage of a well-known development environment and its extension mechanisms. The tools were implemented as Eclipse plug-ins (see Figure 7) and consist of: editor plug-ins for the UP-ColBPIP language that allow modeling an integration agreement and the collaboration processes; a plug-in for Petri Nets that allows editing CP-Nets models; a transformation engine for Petri Net that implements the rules of the method for generating CP-Net integration process models from UP-ColBPIP interaction protocols; and a transformation engine for Petri Net Markup Language (PNML) [19] that generates a CP-Net in an XML format from an integration process model.

The editor plug-ins are based on Graphical Modeling Framework and manipulates and store models based on the Eclipse Modeling Framework. The rules of the transformation engine for CP-NET and PNML were implemented by using the ATL transformation language [17].

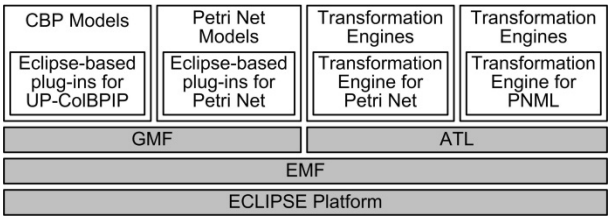

Figure 7. Eclipse-based tool for the proposed MDD-method.

4. Implementation of the agent-based architecture

In this section we present a prototype implementation of the proposed agent-based architecture for managing inter-organizational collaborations, which is shown in Figure 8. The software agents were built using the Java Agent DEvelopment Framework (JADE) [20], which is a physical multi-agent development framework which complies with FIPA specifications [20] and aims at simplifying the development and implementation of multi-agent systems. The proposed architecture based on software agents is composed of a message transport system (MTS) used for communicating with other agents or agent platforms, an agent management system (AMS) intended for managing the agent life cycles such as starting and stopping, and a directory facilitator (DF) used for recording the services provided by an agent. Therefore, the CAAgent and PAAgents are executed on the JADE platform and they use the Agent Communication Language (ACL) messages to communicate among them [21].

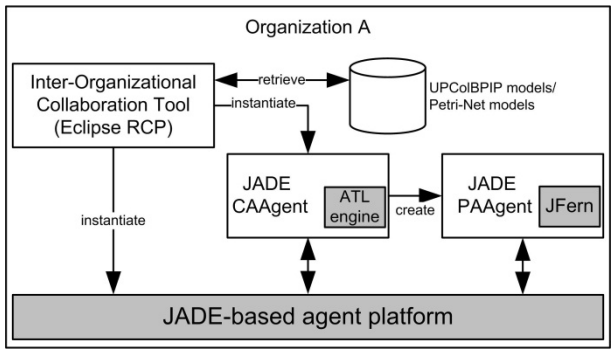

Figure 8. Implementation of the agent-based software architecture.

The *process machine* component of the PAAgent was implemented by using the Java-based Petri Net framework (JFern) [22]. JFern provides an object-oriented Petri Net simulator. It consists of a lightweight Petri Net kernel, providing methods to store and execute Petri Nets in real-time, as well as for simulation. JFern supports XML based persistent storage of Petri Nets and markings. The tokens contain XML formatted *business documents*, which are parts of the content of ACL messages that the agents exchange.

The agent-based architecture includes an administration tool based on Eclipse Rich Client Platform [18]. This tool has to be deployed in each organization and provides the support to:

• Instantiate the JADE platform.

• Instantiate the CAAgent of the organization.

• Search, select and retrieve UP-ColBPIP and CP-Net models stored in the local repository of the organization.

• Manage and monitor the PAAgents instantiated.

5. Example scenario

In this section we illustrate the functionality of the proposed agent-based architecture. We chose a case study from the Supply Chain Integrated Management domain. The collaboration is based on the scenario D of the CPFR business model [23] that involves two manufacturing enterprises. The CBP to be executed by them is the Collaborative Demand Forecast. We apply the above proposed method to generate the CP-Net models.

First, the CBP is defined with the UP-ColBPIP language (see Figure 9). The protocol begins with the customer, who requests a demand forecast. The supplier processes the request and may respond by accepting or rejecting it, as it is indicated by the Xor control flow segment. If it is accepted, the supplier undertakes to realize the required forecast; otherwise, the process finishes with a business failure. If the supplier accepts the request, the customer informs, in parallel, a sale forecast of its points of sales (POS) and its planned sales, as it is indicated by the And control flow segment. Finally, with this information, the supplier generates a demand forecast and sends it to the customer. Then, the process ends.

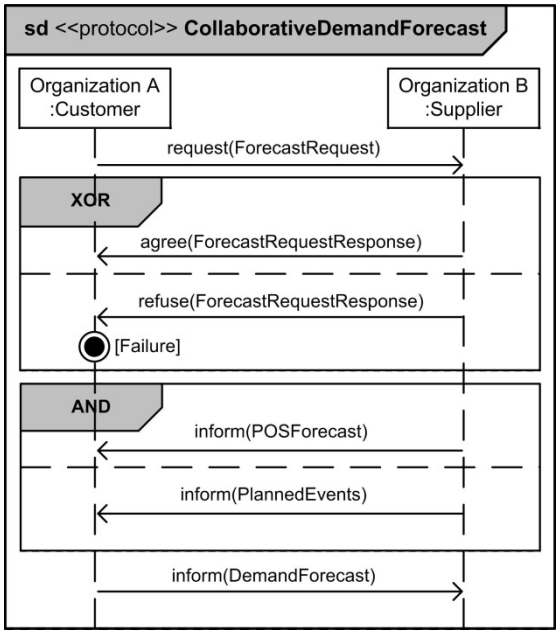

Figure 9. Collaborative demand forecast interaction protocol.

Second, the automatic process of transformation of the UP-ColBPIP model to CP-Net model is carried out. The transformations of the UP-ColBPIP process models are performed by each organization involved in the collaboration relationship. Afterwards to this, the privates activities are added, which are required to support the collaborative processes of each organization. Thus, the CP-Net models contain the behavior of each organization role. These models will serve to determine the behavior of PAAgents. For each individual role there is a separate CP-Net model. The collection of individual CP-Nets associated with all the relevant roles represents the entire interaction protocol. The PAAgents involved in this collaborative process are the customer PAAgent and the supplier PAAgent. Figure 10 shows an excerpt of code of the PNML document generated for the customer role in a Collaborative Demand Forecast process.

Figure 11 shows the Petri Net representing the supplier role that the PAAgents have to execute the Collaborative Demand Forecast collaborative process. We use a graphical editor to read and display the Petri Net model stored in the PNML file. When PNML file is interpreted by the process machine component, the execution of CP-Net model is carried out. In this case, we describes the actions performs by the PAAgents according with the transitions of the CP-Net model.

The actions executed by the customer PAAgent are: First, a generate action is performed to generate the information to be sent. Then, a send action is placed in the queue of the customer PAA*gent* for sending the business document which contains the information previously generated. Next, the PAAgent receives a message, and the *process machine* component placed in the queue of the customer PAAgent a *generate action*, which invokes a process to generates the information of the business documents required. Afterwards, two send actions are placed in the queue for sending of the two different business documents.

Finally, the process machine placed a blocking *receive action*, in waiting for a message. After receiving the message, CP-Net model execution is complete and a method to terminate agents must be called.

```
<transition id="id11">
    <annotation id="a35" type="name"><text>Send request ForecastRequest</text></annotation>
    <annotation id="a60" type="type"><text>sendnoreceive</text></annotation>
</transition>
<transition id="id12">
    <annotation id="a36" type="name"><text>Receive agree ForecastRequestResponse</text></annotation>
    <annotation id="a60" type="type"><text>receivenosend</text></annotation>
</transition>
<transition id="id13">
    <annotation id="a37" type="name"><text>Receive refuse ForecastRequestResponse</text></annotation>
    <annotation id="a60" type="type"><text>receivenosend</text></annotation>
</transition>
```

Figure 10. Excerpt of code of the PNML document generated by transformation engine.

The actions executed by the *supplier PAAgents* are: First, the *process machine* placed a blocking *receive action* (*receive request forecast request*), in waiting for a message, when the message is received, a *process action* (*evaluate forecast request*) is placed in the PAAgent to evaluate the message previously received. Then, a *send action* (*send refuse* or *agree forecast request response*) is placed in the queue of the *supplier PAAgent*, which contains the result of the previous action. Next, a *receive action* (*receive inform POS* and *inform planned events*) is activated in the agent in waiting for a message, next with a *process action* (*process inform POS* and *inform planned events*). Afterwards, the PAAgent can executes a *generate action* (*generate demand forecast*) to invoke an internal process to generate the target business document. Thus, a *send action* (*send inform demand forecast*) is enabled in the *supplier PAAgent* for sending of the message. After sending the message, the *supplier PAAgent* ends.

6. Related work

There are several approaches that exploit the benefits of software agents in the execution of collaborative business processes.

In [24], an approach to the design and implementation of collaborative processes based on an architecture for software agents is described. Private processes are modeled at the level of business, as well as on a technical level, an additional level of abstraction is introduced between private processes and collaborative process, which is called view process. This view process represents an interface for interaction with other organizations, describing the interactions of one or more private processes from the perspective of an organization. At the implementation level collaborative processes are extended with information on the specific platform. The agent model is constructed from a process model, generated at the technical level and based on the Service-Oriented Architecture (SOA).

In [7], a method based on the MDD for the generation of models of software agents capable of running collaborative processes is proposed, which allows generating an agent model specification using capabilities defined as modules that encapsulate functionality. These are defined by using the Web Services Description Language (WSDL) to specify invocations to internal systems of the organization. Such capabilities are defined in the plans of the agent. The agent model contains only interaction messages generated from the collaborative process, and the agent plans are specified with calls to other services, delegating the execution of tasks.

A methodology to model business processes and to generate specifications for implementation in an agent platform is presented in [25]. The methodology uses a MDD-based approach that allows the transformations by JIAC-V Framework [26]. The development process starts with the analysis using use case diagrams. Then, for each use case diagram, a BPMN-based process model is created. Starting from process model and the use case diagrams, the role of each participant, behavior and capabilities are derived. Finally, models of organization (roles and agents) of each agent, and the behavior of the agents (plans, rules, and services) are integrated, allowing the generation of the agent code.

A proposal for the design of interaction protocols based on a language independent modeling platform for the domain of multi-agent systems is described in [27]. First, in an interaction protocol view, used to define the behavior of the agent. The interaction view allows to define the interactions of the actors involved. Then, the system designer can perform a refinement to the description of behavior through a behavioral view, adding instances and additional information of the private process. Finally, from behavior generated a code transformation is performed, following the structure of an agent based on JACK platform.

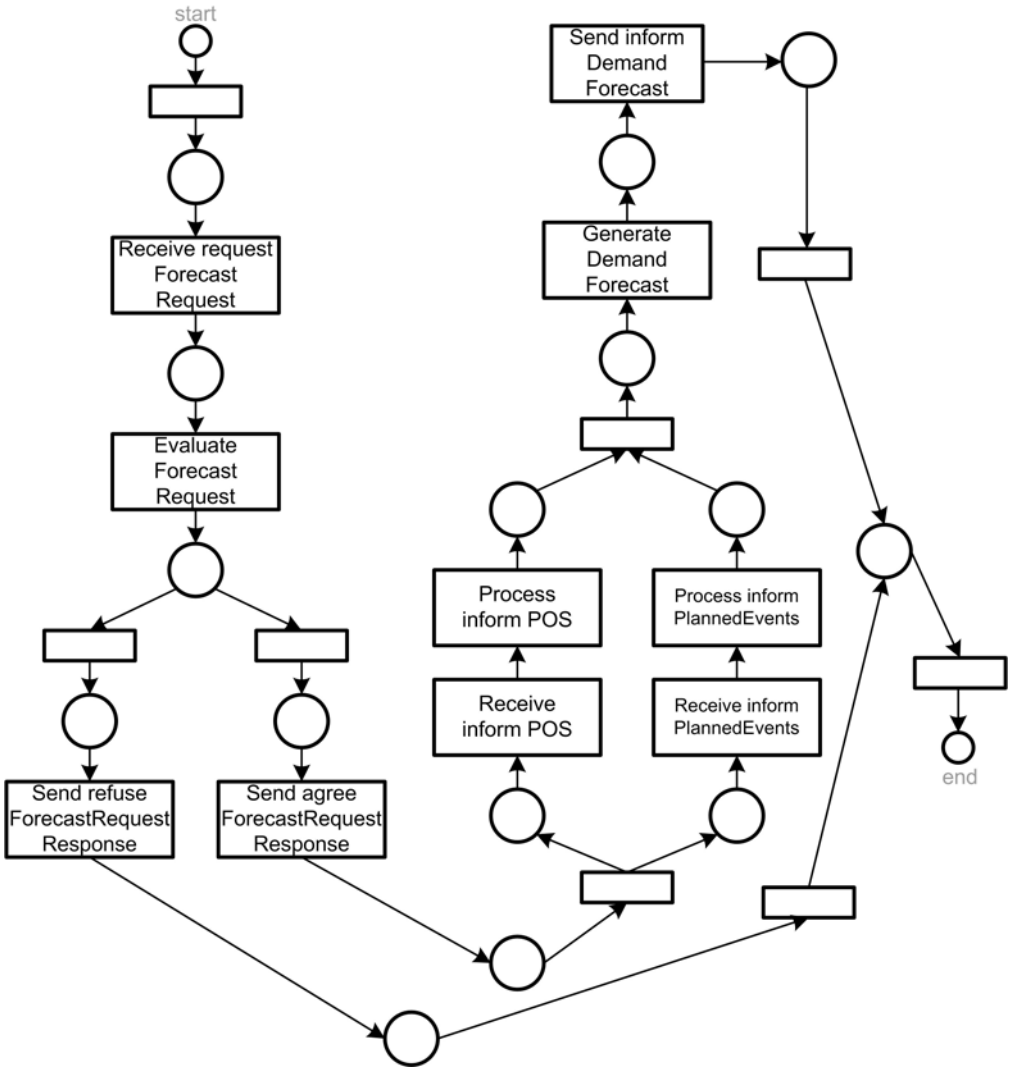

Figure 11. The CP-Net model of the example scenario (supplier role).

7. Conclusion

In this work we have proposed an agent-based software architecture for managing inter-organizational collaborations, which allows organizations setting up collaborations in a dynamic way and carrying out a decentralized execution of collaborative business processes (CBPs). Inter-organizational collaborations can be supported by a system implementing this agent-based software architecture composed by Collaboration Administrator Agents (CAAgents) and Collaboration Agents (PAAgents).

The CAAgents of the organizations execute a request for collaboration protocol to agree and establish an inter-organizational collaboration for the enactment of a CBP. These agents exchange the model of the process to be executed in the collaboration. CBP models are defined with the UP-ColBPIP language that enables the definition of the logic of CBPs as interaction protocols. CAAgents are also responsible for the instantiation of the PAAgents that will execute the CBPs agreed. Thus, through these agents organizations can achieve dynamic agreements for executing CBPs.

The Colored Petri Net models are used to specify the behavior a PAAgent needs to perform the role an organization fulfills in a CBP. These models are automatically generated from UP-ColBPIP models. Hence, the decentralized execution of a CBP is achieved by the enactment of CP-Net models carried out by each involved PAAgent. Due to the behavior of PAAgents is driven by Petri Net-based process machine, their action plans for executing an interaction protocol representing a CBP can be generated in run-time. The agent's actions to be executed are obtained at run-time according to the logic defined in CP-Net models. This makes more flexible the architecture of agents for setting up new collaborations and for adapting to changes in CBPs. Changes on CBP models are immediately reflected in CP-Net models, without having to modify the implementation of the software agents. Besides, this approach is different from the traditional development of software agents where their action plans and interaction protocols are beforehand implemented and defined in design-time.

In addition, we have proposed a model-driven development method for automatically generating CP-Net models of the PAAgents from CBP models. This method enables that CAAgents instantiate PAAgents and assign to them the Petri Net model that define their behavior for executing a CBP.

Finally, an implementation of the proposed architecture was presented, which is based on the JADE agent platform. A Petri Net simulator was used to implement the process machine of the PAAgent. Also, a distributed inter-organizational collaboration management tool based on the Eclipse platform was developed that enables organizations to manage their agents and their own repository of CBP models.

Future work is about the addition of mechanisms and tools to discover CBP models either in public or organizations' repositories. In the current implementation of the PAAgent, private activities that generate or process the exchanged messages are simulated. A support for a real integration and execution will be developed by integrating Web service technology with software agents.

References

[1] P.D. Villarreal et al., "Modeling and Specification of Collaborative Business Processes with a MDA Approach and a UML Profile," in P. Rittgen (eds), Enterprise Modeling and Computing with UML, Hershey, PA: Idea Group Inc., 2007, pp. 13-45.

[2] S. Roser and B. Bauer, "A Categorization of Collaborative Business Process Modeling Techniques," in 7th IEEE International Conference on E-Commerce Technology Workshops, 2005, pp. 43-54.

[3] M. Weske, "Business Process Management. Concepts, Languages, Architectures", Berlin, Germany: Springer, 2007.

[4] E. Tello-Leal et al., "An Agent-Based B2B Collaboration Platform for Executing Collaborative Business Processes," in C. Wojciech & E. Estevez (eds), Software Services for e-World, Berlin, Germany: Springer, 2010, pp. 40-50.

[5] L. Bearzotti et al., "The event management problem in a container terminal," Journal of Applied Research and Technology, vol. 11, no. 1, pp. 95-102, 2013.

[6] C.V. Trappey et al., "The design of a JADE-based autonomous workflow management system for collaborative SoC design," Expert Systems with Applications, vol. 36, no. 2, pp. 2659-2669, 2009.

[7] I. Zinnikus et al., "A Model-driven, Agent-based Approach for the Integration of Services into a Collaborative Business Process," in 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008), 2008, pp. 241-248.

[8] L. Guo et al., "A Novel Approach for Enacting the Distributed Business Workflows Using BPEL4WS on the Multi-Agent Platform," in IEEE International Conference on e-Business Engineering (ICEBE 2005), 2005, pp. 657-664.

[9] B. Selic, "The Pragmatics of Model-Driven Development," Journal IEEE Software, vol. 20, no. 5, pp.19-25, 2003.

[10] P.D. Villarreal et al., "A Modeling Approach for Collaborative Business Processes Based on the UP-ColBPIP Language," in W. Aalst et al., (eds), Business Process Management Workshops, Berlin, Germany: Springer, 2010, pp. 318-329.

[11] G. Wagner, "The Agent-Object-Relationship Meta-Model: Towards a Unified View of State and Behavior," Information Systems, vol. 28, no. 5, pp. 475-504, 2003.

[12] R. M. Dijkman et al., "Semantics and analysis of business process models in BPMN," Information and Software Technology, vol. 50, no. 12, pp. 1281-1294, 2008.

[13] W.M.P. van der Aalst, "Three Good Reasons for Using a Petri Net-based Workflow Management System," in International Working Conference on Information and Process Integration in Enterprises (IPIC-96), 1996, pp. 179-201.

[14] K. Jensen and L. M. Kristensen, "Coloured Petri Nets, Modelling and Validation of Concurrent Systems", Berlin, Germany: Springer, 2009, pp. 34.

[15] OMG., MDA Guide V1.0.1, 03-06-01.pdf. (online), Available from: http://www.omg.org/cgi-bin/doc?omg/03-06-01.

[16] I.M. Lazarte et al., "An MDA-based Method for Designing Integration Process Models in B2B Collaborations," in 13th International Conference on Enterprise Information Systems (ICEIS 2011), SciTePress, 2011, pp. 55-65.

[17] F. Jouault et al., "ATL: a model transformation tool," Science of Computer Programming, vol. 72, no. 1–2, pp. 31-39, 2008.

[18] Eclipse, Eclipse Platform. (online), Available from: http://www. eclipse.org

[19] M. Weber and E. Kindler, "The Petri Net Markup Language", in H. Ehrig et al., (eds), Petri Net Technology for Communication-Based Systems, Berlin, Germany: Springer, 2003, pp. 124-144.

[20] F. Bellifemine et al., "Developing Multi-Agent Systems with JADE", England: Wiley, 2007.

[21] FIPA, FIPA Agent Communication specifications deal with Agent Communication Language (ACL), (online), Available from: http://www.fipa.org/repository/aclspecs.html, 2002.

[22] M. Nowostawski, JFern - Java-based Petri Net framework, 2003.

[23] VICS, Collaborative planning, forecasting, and replenishment - Voluntary guidelines, V 2.0., (online), Available from: http://www.vics.org/committees/cpfr/voluntary v2/

[24] T. Kahl et al., "Architecture for the Design and Agent-Based Implementation of Cross-Organizational Business Processes," in R.J. Goncalves et al., (eds), Enterprise Interoperability II, London: Springer, 2007, pp. 207-218.

[25] T. Küster et al., " Integrating Process Modelling into Multi-Agent System Engineering," Multiagent and Grid Systems, vol. 8, no. 1, pp. 105-124, 2012.

[26] B. Hirsch et al., "Merging Agents and Services - the JIAC Agent Platform," in A. El Fallah et al., (eds), Multi-Agent Programming, USA: Springer, 2009, pp. 159-185.

[27] C. Hahn et al., "Automatic Generation of Executable Behavior: A Protocol-Driven Approach," in M.P. Gleizes and J.J. Gomez-Sanz, (eds), Agent-Oriented Software Engineering X, Berlin, Germany: Springer, 2011, pp. 110-124