BRIDGING THE GAP BETWEEN BUSINESS EXPERTS AND SOFTWARE AGENTS: BPMN TO AUML TRANSFORMATION

Andreea URZICĂ[1], Claudiu TĂNASE[2], Adina Magda FLOREA[3]

La momentul actual există o barieră între sfera de business şi cea a dezvoltatorilor de agenţi software. Pentru a crea o legatură între cele două lumi, trebuie găsite metode pentru simplificarea procesului de transfer al cunoştinţelor între rolurile implicate. Acest articol propune o traducere din limbajul familiar analiştilor de business pentru design-ul modelului de business într-unul după care se pot ghida dezvoltatorii de software pentru implementarea proceselor. Articolul conţine prezentarea unei metode originale pentru maparea proceselor BPMN (Business Process Modeling Notation) în specificaţii AUML (Agent UML), precum şi o analiză a transformărilor existente între diferite limbaje de modelare sau specificare.

Currently, there is a gap between the business area and multi-agent oriented software development. In order to create a bridge between the two worlds, we saw the necessity of simplifying the knowledge transfer between the roles involved. This paper proposes a translation between the language familiar to the business analysts when designing a business model and the one easily understood by the developers when implementing a process. It consists both of an original method for mapping BPMN (Business Process Modeling Notation) processes to AUML (Agent UML) agent specifications and an analysis on various transformations between different modeling and specification languages, as found in research papers and implemented mapping tools.

Keywords: business processes, agents, BPMN, AUML, interoperability

1. Introduction

The Service-Oriented Architecture Concept is very popular these days. The promises and benefits are well known and the technology mastery of service abstraction is already available. It is also well known that a successful Service-Oriented Architecture (SOA henceforward) depends on business vision, methodology, reuse initiative and an organizational structure. Business analysts

[1] PhD student, Faculty of Automatic and Computers, University POLITEHNICA of Bucharest, Romania, e-mail: andreea.urzica@cs.pub.ro
[2] Eng., Faculty of Automatic and Computers, University POLITEHNICA of Bucharest, Romania, e-mail, claudiutanase@gmail.com
3 Prof., Faculty of Automatic and Computers, University POLITEHNICA of Bucharest, Romania

are those able to use business process models as the link between business architecture and IT design, specifically SOA. They establish the optimal flow for business processes and also the capabilities of the roles involved, the resulted design being called a business model.

This paper focuses on the interoperability concept, a defining requirement in the Service-Oriented Architecture context, as the need for a unified way of representing and transferring information is present even during the development phase. The article does not provide a unified language, but facilitates the translation between existing ones. Along with similar translation efforts [1], we hope to achieve interoperability and automation in the development process. This can provide a link between the high-level concepts of the business and the software implementation by means of translation tools and automated processes that may significantly reduce development time and be used by a larger community (the nature of Business Process Management dictates that its representation must be accessible to non-technical experts). Last but not least, shortening the distance between the idea, concept or basic design of any solution (particularly a business process) and its implementation is part of the natural tendency towards a more user-friendly software development.

While the most widely used standard for modelling business process is BPMN (*Business Process Modeling Notation*), a promising approach for executing them is the use of software agents. Autonomous agents offer the coordination framework and advanced technological means that make them very suitable to be used in the development of complex applications, such as business processes.

The aim of this paper is twofold: to present an analysis of existing efforts to map different modeling specification and implementation languages related to Business Processes (BP) and agents, and to investigate the possibilities of automatic transformation between business process diagrams and agent interaction diagrams by proposing a mapping of BPMN [2] processes to AUML agent specifications [3]. We consider that this mapping is needed at the moment as there certainly is an important gap between the business experts and multi-agent oriented software development. Taking the necessary steps towards an automatic process would help simplifying the transfer of knowledge between the different roles involved in the cycle.

The paper is structured as follows. Section 2 argues the perspective of using software agents in executing business processes. In Section 3 we analyze the existing transformations between various representation and implementation languages of business processes and agents and we propose a new one. In order to define the context, Section 4 and 5 present the BPMN and AUML languages. The actual mapping between the two languages is presented in Section 6. For a better

understanding, in Section 7 we provide a simple example that illustrates the mapping. The last section draws on conclusions and presents future work.

2. Towards executing business processes using agents

Currently there is a process oriented view of Web Services, as the business modelling standards extend the Web Services standards. A business process is a collection of interconnected activities following a business logic in order to solve a problem for a client or for a market. The business logic is especially focused on how the tasks are placed in time and space, and not on the resulting product.

Agents turn out to be very appropriate in executing business processes as they are able, on behalf of their users, to get involved in message exchange, auctions, negotiations, and to use services.

In 1997, Wooldridge gave the following definition of an agent: *An agent is an encapsulated computer system that is situated in some environment and that is capable of flexible, autonomous action in that environment in order to meet its design objectives* [4]. The definition has been reshaped many times; a more functional approach defines an agent as being *a software program that independently performs its task on behalf of its user.* [5]

Many problems involve the use of multiple agents. An agent-oriented view of the world implies focusing on the interaction between the agents. The agents try to achieve their individual objectives or to manage the dependencies that ensue from being situated in a common environment [6]. A community of agents that communicate together, making decisions in order to accomplish their goals is referred to as a *multi-agent system*. Also, agents can be *reactive* or *proactive*. They are reactive in that they respond to other agents. They are proactive in that they work to accomplish a goal. [5]

Technologically speaking, Agent Oriented Programming (AOP) may be seen as a specialization of the Object Oriented Programming (OOP) paradigm [7]. Unified Modelling Language (UML) is gaining more and more acceptance in representing technological artifacts in Object Oriented Programming, and it has became the de-facto standard for analysis and design in this domain. The agents are seen as being the next step beyond objects and that leads to the necessity in exploring the UML extensions in order represent the agent specific features.

3. Current efforts and mapping analysis

Previous efforts concerning the design of a multi-agent framework from the vantage point of the business user, and a mapping from the Business Process Modeling Notation, a graphical language used to represent business processes, to agent concepts have been developed by [1].

The authors of the cited paper have chosen to use agents following the BDI paradigm, as they consider this type of agents to be the best suited in order to capture the whole functionality expressed with BPMN. In [1] a set of rules is defined showing how business processes can be modeled in terms of BDI-type agents. The first step of the algorithm is to simplify the BPMN model by putting away all the information concerning positioning. The result is a graph-shaped structure with various types of edges and vertices that must comply with a series of properties in order to be a correct Business Process Diagram (BPD). The authors define a certain formalism to describe a BPD in terms of graphs and graph properties. In addition, they propose the usage of syntactical and semantically verification, combined with a normalization of BPDs.

By contrast, we consider that it would be more appropriate to use general purpose agents as the target of the transformation. We think that considering only a particular species of agents, namely BDI agents, may bring a series of limitations to the use cases of business processes that could be implemented. Additionally, the agents described in [1] are not quite typical agents, but rather a certain type adapted to the constraints imposed by the transformation algorithm.

By using general purpose agents, the transformation becomes applicable and useful to a larger number of use cases. The agents implemented in JADE for example, use a very general model, based on behaviors, that can be easily specialized to realize both reactive and BDI architectures [8]. Agent interactions are best represented by using a designated graphical language, named AgentUML. All a developer has to do in order to implement the agents described with AUML is to follow the interaction protocols clearly represented in the diagram. Moreover, such an approach allows the simple integration of external software into one of the agent tasks. The implementation of the JessBehaviour, for example, allows the use of JESS [9] as the agent-reasoning engine. [10]

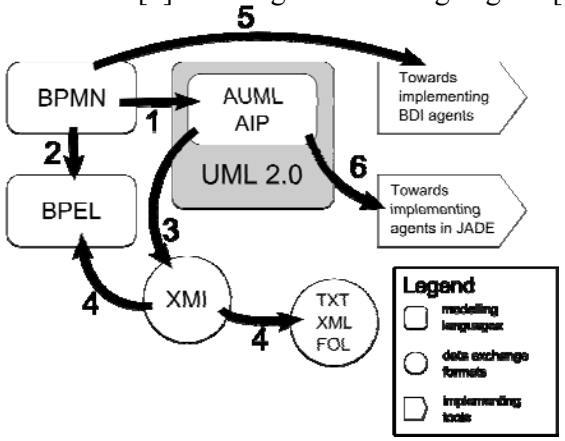


Fig. 1 – Inter-languages translation

Fig. 1 presents a synopsis of the various transformations between different modeling and specification languages, as found in research papers and implemented mapping tools. The number of transformations points out that these languages are strongly connected and have a similar scope of expression, namely the modeling of various facets for designing and implementing business ideas. The numbered links (arrows) in the Fig. 1 are shortly discussed in what follows.

1. – The BPMN to BPEL (Business Process Execution Language [11]) mapping makes the subject of many papers, starting with a translation informally sketched by [12], a discuss on how conceptual mismatch between business analyst and technical analyst process models can be identified, by [13], a technique that proposes to overcome the limitations on the structure of the source model, especially with respect to cycles by [14] to an automatic translation from BPMN to BPEL [15]. Other papers presenting techniques for generating BPEL code from process models are [16], [17] which, in addition, proposes a solution for the round-trip engineering and a case-study by [18] which is targeting four evaluation criteria: completeness, correctness, readability and reversibility.

2. – The XMI (XML Metadata Interchange) representation of a UML diagram is obtained using UML 2.0 tools such as Altova UModel [19].

3. – A very interesting tool is auml2bpel [20] based on [21]. This tool takes as input a XMI file (resulted from a sequence diagram) and returns: a XML file, a .txt file containing the AUML textual notation for the provided sequence diagram, a .fol file containing ProLog rules that can be used in implementing agents in JADE (Java Agent DEvelopment Framework [22]), a .wsdl and a BPEL file – the primary purpose of the application.

4. – BPMN to Agents [1] choosing as target of their transformation the BDI agents.

5. – Providing a mapping between BPMN and AUML is the purpose of this paper.

6. – A paper that bridges the gap between design and implementation is [23].

4. Business Process Modelling Notation

BPMN consists of a set of graphical elements such as shapes, icons and line styles representing tasks, subprocesses or events, which permit the easy development of simple diagrams that will look familiar to most business analysts. Diagrams are essentially flowcharts. Process participants are defined by *pools*, which may be subdivided into *swimlanes* in order to separate different processes. The basic units of a process are tasks, subprocesses, and events. *Subprocesses* and *tasks* are represented by rectangles, events by circles. Various icons within those shapes indicate the particular type of event or subprocess. Solid lines called

sequence flows interconnect the tasks, subprocesses, and events within a pool. In addition, BPMN offers a diamond *Gateway* shape that, depending on its icon, can be used for branching, splits, conditional splits, merges, or synchronizing joins.

In addition to sequence flows within a pool, BPMN shows *message flow*s exchanged *between* pools. These indicate the signals the process engine uses to communicate with invoked services and partner processes. They are optional in the BPMN diagram, but are extremely useful for translating the diagram into real-world executable solutions.

A very important component introduced by the BPMN standard to be used when designing a diagram are the *events*. Events are actions triggered by various signals, such as receipt of a message, expiration of a timer, detection of an error, etc. Each event has a trigger and a resulting action. The various types of triggers and resulting actions are indicated by the placement of the event in the diagram along with its internal icon. As a general rule, a sequence flow coming into the event icon suggests that the process *issues* the event, while an outgoing sequence flow shows that the process is *triggered* by the event.

5. Agent Unified Modeling Language

It has been recognized that the use of software agents is unlikely to gain wide acceptance in industry unless it relates to de facto standards (object-oriented software development) and supports the development environment throughout the full system lifecycle. Therefore, an extension of the Unified Modeling language (UML) has been proposed. The proposal is a result of the cooperation between the Foundation of Intelligent Physical Agents (FIPA) and the Object Management Group (OMG). The AUML notation also reflects the recommendations included by the GAIA methodology [24] concerning the high-level summary of a protocol as an atomic unit. [8]

5.1 General description

Bauer et. al. [25] argue that UML is sometimes insufficient for modeling agents and agent-based systems. Basically, this is due to two reasons: *Firstly*, compared to objects, agents are active because they can take the initiative and have control over whether and how they process external requests. *Secondly*, agents do not only act in isolation but in cooperation or coordination with other agents. In a second paper, [8], a subset of an agent-based UML extension for the specification of *agent interaction protocols (AIP)* is described. The definition of an agent interaction protocol (AIP) describes a communication pattern and a semantics that is consistent with the communicative acts within a communication pattern. A communication pattern should have an allowed sequence of messages between agents having different roles and constraints on the content of the messages [8]. Messages play a central role in the overall design of a multi-agent

system and must satisfy standardized communicative (speech) acts which define the type and the content of the messages (e.g. the FIPA Agent Communication Language (ACL) [26], or KQML[27]) [5], [8]. AUML introduces a new type of diagrams called *Protocol Diagrams* as being a combination between sequence diagrams with the notation of state diagrams for the specification of interaction protocols.

5.2 Elements of protocol diagrams

The top-level element of an AIP is the *lifeline*, just like a BPMN diagram consists of swimlanes. The agent lifeline in protocol diagrams defines the time period during which an agent exists, represented by dotted vertical lines. The lifeline starts when the agent of a certain role is created and ends when it is destroyed. The lifeline is represented as a vertical axis, on which the AUML elements are anchored. It specifies the name of the agent to which it belongs to, the role the agent plays in the MAS and its class.

Messages are represented as arrows between lifelines. A message must comply to a formal specification of agent speech acts (e.g. FIPA agent communication language or KQML), which defines the type of content of the message and the message's "intention", specified by the performative. The complete description of FIPA communicative acts can be found at [28]. We will present below some of the most frequently used speech acts:

- "Call for proposal" is a general-purpose action to initiate a negotiation process by making a call for proposals to perform the given action. In normal usage, the agent responding to a *cfp* should answer with a proposition (the "*Propose*" speech act). For accepting or rejecting a previously submitted proposal to perform an action during a negotiation "*Accept Proposal*" and "*Reject proposal*" will be used.
- The sender requests the receiver to perform some action by sending him a "*Request*". The receiver's response can be "*Agree*" or "*Refuse*". If the sender gives up on his intention he will use "*Cancel*".
- "*Inform*" and "*Confirm*" is used when the sender informs the receiver that a given proposition is true, with the latter referring to a specific receiver.
- The speech acts for describing exceptions are: "*Not understood*", used when sender perceived that the receiver performed some action but didn't understand what action it was, and "*Failure*", for a failed attempt at an action.

AUML allows agents to consider several alternatives during the interaction based on their mental states, their intentions and the current state of the interaction. This materializes in interaction operators, which provide modeling support for the communication alternatives. They are represented with a box, surrounding the involved lifelines and time frame, with an interaction operator given in the top-left corner. Boxes can recursively contain messages and other

boxes, and can be divided into a number of regions separated from each other by heavy horizontal dashed lines (which separate the available alternatives). Each region can include a condition depicted as text in square brackets. The types of AUML interaction operators are: weak sequencing, alternative, option, parallel and loop.

- *Alternative* operator shows that the agent has to choose at most one of many paths to follow (similar to a 'switch' structure in programming).
- The *option* operator only considers one path in the region.
- The *parallel* operator describes the parallel execution of different paths in any order, allowing concurrent message transmission.
- *The loop* operator repeats the demarcated exchange of messages as long as the condition is satisfied.

The *weak sequencing* operator as defined in UML 2.0 indicates that the ordering of the event occurrences is significant only within the same lifeline but not also across lifelines in the same box.

6. The BPMN to AUML Mapping

The results of our research show that there are two possible methods for transforming a business process model designed in BPMN into an interaction protocol represented by an AUML diagram. Both languages are visual modeling environments; the former is conceived for business people, the latter is familiar to software developers. A mapping between the two would connect the two worlds together. The agents executing business models offer a simulation that can facilitate the evaluation and improvement of the models.

One alternative would be transforming the BPMN models in BPEL (originally intended as a back end to BPMN) and then representing BPEL in AUML by reversing the existing AUML to BPEL mapping (see Fig. 1). Such an approach implies some shortcomings concerning mainly the difference between the abstraction levels of the two languages. The AUML to BPEL mapping, described in [21] is a more natural one, as AUML is a high level modeling language that can be mapped to an execution language as BPEL. The reverse transformation, although possible with some amount of human assistance, lacks of a real meaning because of the executable nature of the BPEL code.

Our proposal for obtaining an AUML protocol diagram from a business process model consists in a direct mapping of the BPMN components into AUML interaction sequences.

6.1 The need for a textual notation

As BPMN is a visual language, we need to establish a textual notation in order to handle and sketch some equivalence between its elements and those of

AUML. Although AUML was conceived to be used by humans, and not by the machines, there are papers proposing a syntax that would facilitate an automatic process. In his paper, M. Winikoff, [29], presents two primary reasons for defining a textual format for AUML: the simplicity and the usefulness in writing down AUML interaction protocols.

6.2 A textual notation for AUML

M. Winikoff, [29], defines a textual notation meant to capture every aspect of an AUML protocol in a sequence of commands (one per line). The authors of [29] propose an extension to this syntax, by adding new elements and constraints. Our transformation uses the Winikoff textual notation for AUML. A detailed description of the method can be found in [29].

6.3 A textual notation for BPMN

This paper provides a new and original textual notation for BPMN. The textual notation for a BPMN model consists of a sequence of component names (one per line) preceded by the name of the swimlane where it belongs and followed by the list of components towards which it points to. The ":" symbol separates the name of the swimlane and the name of the BPMN element. In order to keep the right order of the elements, the successors of each element (i.e. all the constructs towards he points to) are enumerated between brackets. If the current component is a decision element, the first "parameter" will be the construct corresponding to the satisfied condition branch. If the current component is a message event, then the first chosen successor will be the destination of the message. Please consult Section 7 for examples. A BPMN element is identified by its type (*ExclusiveGateway, MessageEvent, etc.*) along with a unique number inside its swimlane. Here is the BNF description of the syntax:

```
<bpmn> :== <line>*
<line> :== <element> <EOL>
<element> :== <decision_element> | <message_element> |
          <flow_element> | <end_element>
<flow_element> :== <swimlane>":"<flow_element_type>
          element_number> "(" <element> ")"
          <!-- parameter value: the next activity in the flow -->
<decision_element> :== <swimlane> ":" <decision_element_type>
          <element_number> "(" <element> "," <element> ")"
<!-- first parameter: flow if condition is met, second parameter:
          flow if condition is false -->
<message_element> :== <swimlane> ":" <message_element_type>
          <element_number> "(" <element> "," <element> ")"
          <!-- first parameter: message reciever, second
          parameter: next element in flow-->
<end_element> :== <swimlane> ":" <end_element_type>
          <element_number>
```

```
<flow_element_type> :== "StartEvent" | "ServiceTask" |
          "MessageIntermediateEvent" | "TimerIntermediateEvent"
          <!-- the MessageIntermediateEvent is a "flow" element if
          it is the receiving element of the message -->
<decision_element_type> :== "DataBasedExclusiveGateway"
<message_element_type> :== "MessageIntermediateEvent"
          <!-- the MessageIntermediateEvent is a "message" element
          if it is the sending element of the message -->
<end_element_type> :== "EndEvent"
          <swimlane> :== <capital_letter>*
          <element_number> :== <digit>*
```

6.4 The mapping between the BPMN and the AUML textual notations

The following section presents the translation rules, written in the order of precedence (the first having the highest priority), in a CLIPS-style syntax [30]. The rules are not written in a rigorous language, but are meant to demonstrate the translation mechanism.

Swimlanes. Each swimlane in a BPMN model represents the activity flow assigned to an entity. Thus a swimlane can be seen to be the equivalent of a lifeline in an AUML diagram protocol. Generally, the name given to a swimlane is the same with the entity executing the process. The agent specified in AUML may have the same name. Using the textual notation, each swimlane, starting with the uppermost, is declared as an agent, on a new line. The syntax for agent definition, according to [29] is: "agent *shortname longname*".

```
(
(swimlane ?swimlane_sn1 ?swimlane_ln1 <EOL>)
(swimlane ?swimlane_sn2~?swimlane_sn1 ?swimlane_ln2 <EOL>)
=> (write ("agent" ?swimlane_sn1 ?swimlane_ln1 <EOL>))
 (write ("agent" ?swimlane_sn2 ?swimlane_ln2 <EOL>))
)
```

Task. A task is executed internally by the entity. This is the reason why it should not be visible in an interaction protocol.

Events. We have selected only the BPMN events that are meaningful in the AUML context.

The *EndEvent* is the terminal for the translation:

```
(
 (?swimlane_sn ":EndEvent" ?nr <EOL>)
=> (write ("stop" ?swimlane_sn)).
)
```

MessageEvent. The start, stop or intermediate events indicating the receipt or sending of a message are the most relevant in the context of interaction diagrams. The first parameter of the *SendMessageEvent* textual represented is the corresponding *ReceiveMessageEvent*. The AUML textual notation for this construct is: "message <the name of the sender's swimlane> <the name of the destination's swimlane>".

```
(
 (?src_sn ":MessageEvent" ?nr "(" ?dest_sn ":" ?element ","
          ?next_element ")" <EOL>)
=> (write ("message" ?src_sn ?dest_sn <EOL>))
)
```

TimerEvent. This event is not meaningful in the context of agent interaction, as it denotes a single agent's individual state (same as the *Task* above).

Gateways. The transformation acts on the *Parallel Gateway* and *Data-Based Exclusive Gateway*.

Parallel Gateway. This element corresponds to the parallel AUML construct. The transformation takes place as follows: the parallel flows are recursively transformed (using a stack operator) up until the flows are rejoined (both flows end up in one single joining element), and the AUML syntax elements for the parallel construct are inserted between them:

```
(
 (?src_sn ":ParallelGateway" ?nr "(" ?elem1 "," ?elem2 ")" <EOL>)
 ($?elem1 <EOL> ?joinGateway)
 ($?elem2 <EOL> ?joinGateway)
=> (write ("box parallel"))
           (push $?elem1)
           (write ("next"))
           (push $?elem2)
           (write ("end parallel"))
)
```

Data-Based Exclusive Gateway. This gateway routes the sequence flow according to a logical expression, towards exactly one following element. When two or more arrows enter the gateway, at least one has to be active in order to open the gate (the gateway does acts as a barrier). The equivalent element in AUML is an *alternative* region. An *alternative* region is represented by an “alt” labeled box and includes the interactions that depend on the evaluation of the logical expression. For each branch there is one region limited by a dotted line. The AUML textual description of this element is a zone starting with “box alternative”, ending with “end alternative”, the options being separated by a “next” line. If the Gateway's elements are both situated after the Gateway itself in the BPMN, the “alternative” box in AUML will follow the pattern similar to the Parallel Gateway presented above. If one of the Gateway outputs leads to an element situated left (before) of the gateway, we have a *loop*, which iterates until the condition is met. However, distinguishing between the two is non-trivial. After all the other transformation steps have been taken, the remaining gateways form a Finite State Machine, with the transition characters corresponding to the translated AUML portions, and the terminal states represented by the EndEvent. We can then analyze the corresponding regular expression, using the algorithm described in [31] and substitute the *regexp* operators as such:

```
A* => "box loop" A "end loop" ; A+B => "box alternative" A "next"
          B "end alternative"
```

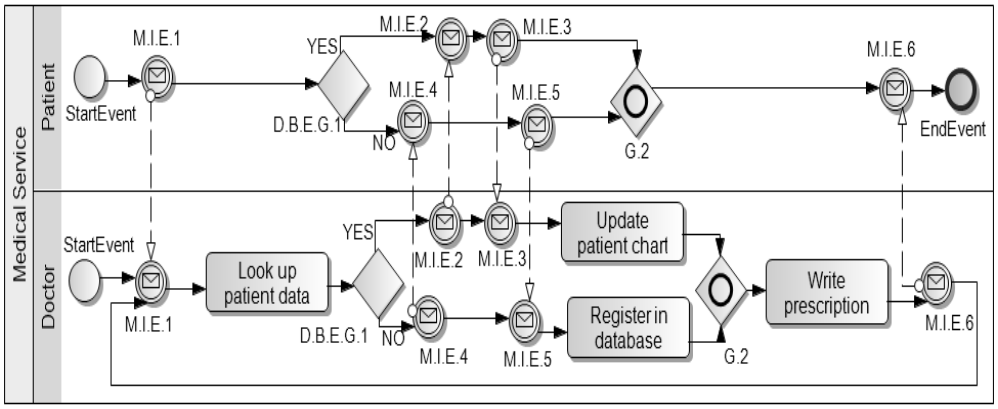
7. Example

In order to properly illustrate the usage of generic agents for business process execution in BPMN and their following implementation in AUML, we have chosen a simple example.

The scenario used is the well-known interaction during a medical visit, between the patient, requesting a prescription, and the doctor. From each patient, upon receiving a prescription request, the doctor verifies whether the patient is registered in his database. In order to append the database, if the patient is new, the doctor must ask him, in addition, some personal details while, if the patient is already registered, the doctor just asks about the symptoms and updates the patient's chart. In the end, the doctor releases the prescription and calls for another patient.

7.2 The BPMN representation

In order to model the business process, we have used the ActiveVOS Designer application [32], and we obtained the result depicted by Fig. 2. For the reader's convenience, we have abbreviated the names for some of the BPMN flow objects according to the following rule: each *DataBasedExclusive Gateway* is represented by "D.B.E.G" followed by its corresponding number, the name for



each *MessageIntermediate Event* is abbreviated to "M.I.E" followed by a number and for the *Inclusive Gateway* we have used "G" as a short name.

Fig. 2 The BPMN representation

7.3 BPMN textual description

Based on the activity flow and applying the methodology presented in Section 6.3, we have obtained the following BPMN textual representation.

```
Patient:StartEvent (Patient:MessageIntermediateEvent1)
Patient:MessageIntermediateEvent1
          (Doctor:MessageIntermediateEvent1,
          Patient:DataBasedExclusiveGateway1)
Patient:DataBasedExclusiveGateway1
          (Patient:MessageIntermediateEvent2
          (Patient:MessageIntermediateEvent4)
Patient:MessageIntermediateEvent2
          (Patient:MessageIntermediateEvent3)
Patient:MessageIntermediateEvent3
          (Doctor:MessageIntermediateEvent3,
          Patient:InclusiveGateway2)
Patient:MessageIntermediateEvent4
          (Patient:MessageIntermediateEvent5)
Patient:MessageIntermediateEvent5
          (Doctor:MessageIntermediateEvent5,
          Patient:InclusiveGateway2)
Patient:MessageIntermediateEvent6 (Patient:EndEvent)

Doctor:StartEvent (Doctor: MessageIntermediateEvent1)
Doctor:MessageIntermediateEvent1 (Doctor:ServiceTask1)
Doctor:ServiceTask1 (Doctor:DataBasedExclusiveGateway1)
Doctor:DataBasedExclusiveGateway1
          (Doctor:MessageIntermediateEvent2,
          Doctor:MessageIntermediateEvent4)
Doctor:MessageIntermediateEvent2
          (Patient:MessageIntermediateEvent2,
          Doctor:MessageIntermediateEvent3)
Doctor:MessageIntermediateEvent3 (Doctor:ServiceTask2)
Doctor:ServiceTask2 (Doctor:InclusiveGateway2)
Doctor:MessageIntermediateEvent4
          (Patient:MessageIntermediateEvent4,
          Doctor:MessageIntermediateEvent5)
Doctor:MessageIntermediateEvent5 (Doctor:ServiceTask3)
Doctor:ServiceTask3 (Doctor:InclusiveGateway2)
Doctor:InclusiveGateway2 (Doctor:ServiceTask3)
Doctor:ServiceTask3 (Doctor:MessageIntermediateEvent6)
Doctor:MessageIntermediateEvent6(Patient:MessageIntermediateEvent6
          Doctor:MessageIntermediateEvent1)
```

7.3 AUML textual representation

We have transformed the textual representation of the BPMN model into the textual description of the AUML interaction diagram (presented in Section 6.2) using the mapping described in Section 6.4.

```
start ms MedicalService
agent p Patient
agent d Doctor
box loop
      message p d
      box alternative
            message d p
            message p d
      next
            message d p
            message p d
      end alternative
      message d p
end loop
finish
```

7.4 AUML representation

The diagram shown in Fig. 3 represents the AUML interaction protocol corresponding to the AUML textual description presented in Section 7.3.

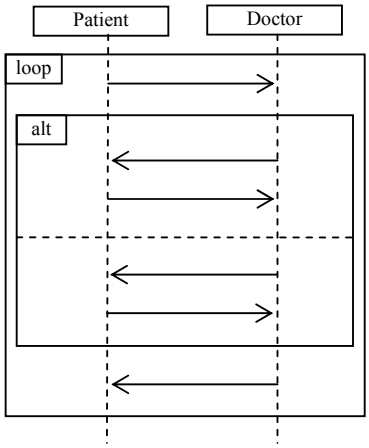


Fig. 3 AUML representation

An example illustrating a more complex loop handling in agent interaction can be found in [33], the model presented by this paper representing a refinement of the model described by the referred work.

8. Conclusions

This paper presents some alternatives that would increase the interest of using software agents in executing business process. In order to achieve a greater impact from using software agents, the emphasis must fall on the interoperability

concept. This paper provides the first steps towards an automatic process of mapping a BPMN model to an AUML diagram. The flexibility and dynamic behavior of software agents in conjunction with the wide acceptance of the BPMN and AUML standard notations strongly justify the need for such an approach.

After providing an analysis of the existing efforts, the paper proposes an original translation from BPMN to AUML. In order to handle the constructs of the two graphical languages, we introduce a new textual notation for BPMN and use the existing textual notation for AUML.

Having defined the basic rules of the transformation, we plan to extend our approach to an automatic translator based on the specifications in Section 6.4. Other points of investigation aim at taking advantage of interconnecting the shared concepts between AUML and BPMN, concepts shared as well by other representational language, and create a more general conversion system that would allow various translations between languages belonging to a wider set.

Acknowledgments

This research was supported by project PNII - Parteneriate Nr. 12118/2008 and Grant POSDRU ID 7713.

R E F E R E N C E S

[1] *H. Endert, T. Kuster, B. Hirsch and S. Albayrak.* Mapping BPMN to Agents: An Analysis. Agent, Web Services, and Ontologies Integrated Methodologies, 2007
[2] http://www.bpmn.org , accessed Feb 2009
[3] http://www.auml.org/ , accessed Feb 2009
[4] *M. Wooldridge*. Agent-based software engineering. IEEE Proc. Software Engineering, **vol. 144**, no. 1, 1997, pp. 26-37
[5] *J. Bisschop and M. Roelofs*. "AIMMS" in The Multi Agent and Web Services User's Guide AIMMS 3.8 July 15, 2008, Paragon Decision Technology B.V.
[6] *N.R. Jennings*. On agent-based software engineering. Artificial Intelligence, **vol. 117**, 2000, pp. 277-296
[7] *Y. Shoham,* Agent-oriented programming. Artificial Intelligence, **vol. 60**, 1993, pp. 51-92
[8] *B. Bauer, J.P. Muller, J. Odell.* Agent UML: A Formalism for Specifying Multiagent Interaction. Agent-Oriented Software Engineering, Springer-Verlag **, vol. 1957**, 2001, pp. 91-103
[9] http://www.jessrules.com/links/ , accessed Feb 2009
[10] *F. Bellifemine, A. Poggi and G. Rimassa.* JADE - A FIPA-compliant agent framework. Proceedings of PAAM, 1999
[11] http://en.wikipedia.org/wiki/Business_Process_Execution_Language , accessed Feb 2009
[12] *S. White.* Using BPMN to Model a BPEL Process. BPTrends, **vol. 3**, no. 3, 1999, pp. 1-18
[13] *J. Recker, J. Mendling*. On the Translation between BPMN and BPEL: Conceptual Mismatch between Process Modeling Languages. Proceedings 18th International Conference on Advanced Information Systems Engineering , 2006, pp. 521–532
[14] *C. Ouyang, W.M.P. van der Aalst, M. Dumas, A.H.M. ter Hofstede*. Translating BPMN to BPEL. BPM Center Report BPM-06-02*,* 2006

[15] *C. Ouyang, W.M.P. van der Aalst, M. Dumas, A.H.M. ter Hofstede*. From Business Process Models to Process-oriented Software Systems: The BPMN to BPEL Way, 2006
[16] *C. Ouyang, M. Dumas, S. Breutel, A. Hofstede*. Translating Standard Process Models to BPEL. Lecture Notes in Computer Science, Springer, **vol. 4001**, 2006, pp. 417
[17] *Y. Gao*. BPMN-BPEL transformation and round trip engineering, URL: http://www. eclarus. com/pdf/BPMN BPEL Mapping. pdf, March 2006
[18] *M. Dumas*. Case Study: BPMN to BPEL Model Transformation. 4th International Workshop on Graph-Based Tools: The Contest, 2008
[19] www.altova.com, accessed Feb 2009
[20] http://www.disi.unige.it/person/MascardiV/Software/AUML2WS-BPEL.html, accessed Jan 2009
[21] *Casella, V. Mascardi*, From AUML to WS-BPEL, Technical report, Computer Science Department, University of Genova, Italy, 2001
[22] http://jade.tilab.com., accessed Feb 2009
[23] *T. Doi, N. Yoshioka, Y. Tahara, S. Honiden*, Bridging the Gap Between AUML and Implementation Using IOM/T, 2005
[24] *M. Wooldridge, N.R. Jennings, D. Kinny*, The Gaia Methodology for Agent-Oriented Analysis and Design, Springer, Autonomous Agents and Multi-Agent Systems, **vol. 3**, no. 3, 2000, pp. 285-312
[25] *J. Odell, H.V.D. Parunak, B. Bauer*, Extending UML for Agents, Ann Arbor, **vol. 1001**, 1999, pp. 48–103
[26] http://www.fipa.org/repository/aclspecs.html , accessed Feb 2009
[27] http://www.cs.umbc.edu/research/kqml , accessed Feb 2009
[28] http://www.fipa.org/specs/fipa00037/SC00037J.html#_Toc26729689 , accessed Feb 2009
[29] *M. Winikoff*, Towards making Agent UML practical: A textual notation and a tool. Proc. of the 1st International Workshop on Integration of Software Engineering and Agent Technology (ISEAT 2005), 2005
[30] *** CLIPS Reference Manual, Basic Programming Guide. 2006 http://clipsrules.sourceforge.net/documentation/v624/bpg.htm, accessed Feb 2009
[31] *J.E. Hopcroft, R. Motwani, J.D. Ullman*. Introduction to Automata Theory, Languages, and Computation 2nd, Addison-Wesley, 2001, pp 96
[32] www.activevos.com, accessed Feb 2009
[33] *A. Urzica, C. Tanase*, Mapping BPMN to AUML: Towards an automatic process, Proceedings of the 17th International Conference of Control Systems and Computer Science, MASTS 2009 Workshop, 2009, pp. 539—547.