Validating Requirements Using Gaia Roles Models

Nektarios Mitakidis[1], Pavlos Delias[2], Nikolaos Spanoudakis[3]

[1]School of Electronic and Computer Engineering (ECE)
[3]School of Production Engineering and Management (PEM),
Applied Mathematics and Computers Laboratory (AMCL)
Technical University of Crete (TUC), Chania, Greece
{nmitakidis, nispanoudakis}@isc.tuc.gr
[2]Business School,
Eastern Macedonia and Thrace Institute of Technology, Kavala, Greece
pdelias@teiemt.gr

Abstract. This paper presents a method that aims at assisting an engineer in transforming agent roles models to a process model. Thus, the software engineer can employ available tools to validate specific properties of the modeled system before its final implementation. The method includes a tool for aiding the engineer in the transformation process. This tool uses a recursive algorithm for automating the transformation process and guides the user to dynamically integrate two or more agent roles in a process model with multiple pools. The tool usage is demonstrated through a running example, based on a real world project. Simulations of the defined agent roles can be used to a) validate the system requirements and b) determine how it could scale. This way, engineers, analysts and managers can configure the processes' parameters and identify and resolve risks early in their project.

Keywords: model checking agents and multi-agent systems·business process models·agent simulation·Gaia methodology

1 Introduction

This paper aims to show how a Gaia Multi-Agent System (MAS) analysis (or architectural design) role model can be represented as a business process model. This allows employing available tools to validate specific properties of the modeled system before its final implementation. Moreover, a business partner has a greater potential to comprehend the system being modeled through intuitive process visualization.

Rana and Stout [1] highlighted the importance of combining performance engineering with agent oriented design methodologies in order to develop large agent based applications. To derive process performance measures, we need a quantitative process analysis technique. Process simulation appears to be a prominent technique that allows us to derive such measures (e.g., cycle time) given data about the activities (e.g., processing times) and data about the resources involved in the process. Through process simulation an engineer can forecast the process execution time, identify possible bottlenecks and perform tests regarding the response of the process to increasing demand. Process simulation is a versatile technique supported by a range of process modeling and analysis tools [2]. However, to run a process simulation, the engineer needs a process model.

In this paper we will see how liveness formulas, an important property of agent role models, introduced by the Gaia methodology [3], and later employed by ROADMAP [4], the Gaia2JADE process [5], Gaia4E [6] and ASEME [7], can be transformed to process models. Moreover, we will present a tool that allows these models to be integrated to produce a process model of a multi-agent system using the XML Process Definition Language (XPDL) [8] portable standard. Having transformed the MAS role model to a process model, we can use simulation to validate several properties of the modeled system, and also determine its ability to scale, as early as the analysis [3] or architectural design (introduced in the second version of Gaia [9]) phases. This is demonstrated through a case study based on real world system's requirements for smart-phone services.

Therefore, this work is expected to have a high impact on a) Agent Oriented Software Engineering (AOSE) practitioners using the Gaia methodology and its successors, who can immediately take advantage of this work to evaluate their models, b) AOSE researchers, and practitioners of other methodologies who can use this transformation combined with method engineering to compile new methodologies, and, c) those who use business process models for agent-based simulations [10, 11] or for communicating them to business people [12], who can now use an AOSE methodology to aid them in their modeling tasks.

In the following section we will briefly discuss the background of this work. Then, in section three, we will present the algorithm for the automatic transformation process and, in section four, the tool that allows integrating many individual agent processes to build a common process that will resemble how the different agents collaborate. In section five we will present the results of a number of simulations. In section 6 we present the software process fragment that an engineer can use to integrate this method to an existing software engineering process. Section seven discusses our findings and the tool's limitations, and, finally, section eight concludes and provides an insight to future work.

2 Background

2.1 The Gaia Liveness formulas and AOSE

The liveness property of an agent role was introduced by the Gaia methodology [3, 9]. Gaia is an attempt to define a general methodology for the analysis and design of MAS. MAS, according to Gaia, are viewed as being composed of a number of autonomous interactive agents forming an organized society in which each agent plays one or more specific roles. The latest version of Gaia defines a three phase process and at each phase the modeling of the MAS is further refined. These phases are the analysis phase, the architectural design phase, and, finally, the detailed design phase. In the analysis phase, Gaia defines the structure of the MAS using the *role* *model*. This model identifies the roles that agents have to play within the MAS and the interaction protocols between the different roles. The role model is further refined in the architectural design phase [9].

The objective of the Gaia analysis phase is the identification of the roles and the modeling of interactions between the roles found. Roles consist of four attributes: *responsibilities*, *permissions*, *activities* and *protocols*. Responsibilities are the key attribute related to a role since they determine the functionality. Responsibilities are of two types: *liveness properties* – the role has to add something good to the system, and *safety properties* – the role must prevent something bad from happening to the system. Liveness describes the tasks that an agent must fulfill given certain environmental conditions and safety ensures that an acceptable state of affairs is maintained during the execution cycle. In order to realize responsibilities, a role has a set of permissions. Permissions represent what the role is allowed to do and, in particular, which information resources it is allowed to access. The activities are tasks that an agent performs without interacting with other agents. Finally, protocols are specific patterns of interaction with other roles.

Gaia originally proposed some schemas that could be used for the representation of interactions between the various roles in a system. However, this approach was too abstract to support complex protocols [5]. ROADMAP [4] proposed that protocols and activities are social actions or tasks and ASEME [13] moved one step further by allowing protocols to define the involved roles processes as liveness formulas that would later be included in the liveness of the system role model (a model inspired by the Gaia roles model). This is one assumption of this work, i.e. that the protocols are a send message action, a receive message action or a combination of message send and receive actions and, possibly, other activities for each participating role.

Although the Gaia methodology does not explicitly deal with the requirements capture phase, it supposes that they exist in some kind of form before the analysis phase. ASEME supports the systematic gathering of requirements in free text form and associating them with the goals of specific actors in the System Actor-Goals Model [7]. Since ASEME has adopted a model-driven engineering approach these requirements influence the role model definition, which emerges at the end of the analysis phase.

In both cases, it makes sense to seek to validate or forecast specific properties of the system to be, based on its requirements. Until now, an analyst can only reach this goal by manually transforming the model. In this paper, we propose a systematic method for achieving the same goal. The advantages of such an approach are that it can be automated, is less error prone and faster. This is the actual research question of this work.

The liveness model has a formula at the first line (*root formula*) where activities can be connected with Gaia operators. Abstract activities must be decomposed to *activities* again connected with Gaia operators in a following formula. The operators used in the liveness formulas are:

A+	(activity A is executed one or more times)
A^*	(activity A is executed zero or more times)
[A]	(activity A is optionally executed)

A.B	(activity B executes after activity A)
A|B	(activity A or B exclusively is executed)
A||B	(activities A and B are executed in parallel)
A~	(activity A is executed forever, the original Gaia operator was the greek character omega "ω", however for keyboard compatibility we chose to use the tilde)

Figure 1 shows a Gaia roles model for an indicative role named *ComplexProvider*. This role employs two protocols, one for servicing a complex service request and one for requesting a simple routing service (activities are underlined in the *Protocols and Activities* field). In its liveness formula it describes the order that these protocols and activities will be executed by this role using three liveness formulas.

The liveness property is defined as a string, adhering to a grammar. The latter is defined using the Extended Backus–Naur Form (EBNF), which is a metasyntax notation used to express context-free grammars [14]. In Listing 1 we define the liveness property grammar (*char* is any lower or upper case alphabetic character).

Role: ComplexProvider

Description: This role provides an added value service in routing requests. It receives a routing request containing needed information but also the user's preferences. Firstly it decides the route type to request (public transport, car and/or pedestrian), then it composes a simple routing request and after it gets the results it sorts them according to the user's preferences.

Protocols and Activities: ComplexService, ReceiveComplexServiceRequest, DecideRouteType, SimpleService, SortRoutes, SendComplexServiceResponse, SendSimpleServiceRequest, ReceiveSimpleServiceResponse.

Responsibilities - Liveness:

CP = ComplexService+

ComplexService = ReceiveComplexServiceRequest. DecideRouteType. SimpleService. SortRoutes. SendComplexServiceResponse

SimpleService = SendSimpleServiceRequest. ReceiveSimpleServiceResponse.

Fig. 1. Part of the Gaia role model for a role.

2.2 Metamodels and Model Transformations

Model transformation is an essential process in Model Driven Engineering (MDE). It is the process of transforming a model to another model [15]. To define a transformation an engineer needs the metamodels of the source and target models. A model is defined as an abstraction of a software system (or a part of it) and a metamodel is an abstraction defining the properties of the model. A metamodel is itself a model. For example, the metamodel of a text model can be the EBNF grammar.

A model's metamodel defines the elements that can be used by the engineer to create the (terminal) model, usually in a format defined by a metametamodel which is the language for defining metamodels. The Eclipse Modeling Framework (EMF, [16]) defines such a language, namely *ecore*, that is much like a UML Class definition. Ecore defines that a model is composed of instances of the *EClass* type, which can have attributes (instances of the *EAttribute* type) or reference other EClass instances (using the *EReference* type). *EAttributes* can be instances of terminal data types such as *string*, *integer*, *real*, etc). EMF allows to extend existing models via inheritance, using the *ESuperType* relationship for extending an existing *EClass*.

Thus, using EMF technology, in order to define the text to model transformation that is the liveness to XPDL transformation we need the XPDL metamodel.

Listing 1. The liveness property grammar

```
Liveness        → {Formula}
Formula         → LeftHandSide, "=", Expression
LeftHandSide    → string
Expression      → Term | ParallelExpr | OrExpr | SequentialExpr
ParallelExpr    → Term, "||", Term { "||", Term }
OrExpr          → Term, "|", Term { "|", Term }
SequentialExpr→ Term, ".", Term { ".", Term}
Term            → BasicTerm | "(", Expression, ")" |
               "[", Expression, "]" | Term, "*" | Term, "+" |
                Term, "~"
BasicTerm       → string
String          → char, {char | digit | "_"}
```

2.3 Business Process Modeling

Software Engineering (SE) and Business Process Management (BPM) are two disciplines with clear associations. A visible influence of SE to BPM concerns quality assessment, while SE aims its attention to BPM mainly to take advantage of its advanced monitoring and controlling functions [17] and its experiment design principles. For example, following the BPM paradigm, one can find solutions about how business people and software engineers are facilitated in communicating system requirements. Stakeholders are able to get involved in the system's design, and hence to assure the alignment of the produced software with the business objectives.

Simulation is employed to quantify the impact that a process design is likely to have on its performance, and to numerically indicate the best design alternatives. Regarding business process simulation, various tools exist [18], which facilitate the adoption of BPM as a practical way for designing systems. However, a critical factor in selecting which tool is more appropriate is the modeling language used.

Popular modeling languages in designing software systems, such as the object-oriented ones (e.g., UML), lack process views, an issue that has been early identified by [17]. On the other hand, process models do not usually map clearly to a programming environment. Both approaches have their relative advantages, so it is a hard decision to spare one. This is why there have been efforts to bridge object-oriented models and process models through model transformations [17, 19].

In this work we chose the XML Process Definition Language (XPDL version 2.1) as the target language. XPDL, a standard supported by the Workflow Management Coalition (WfMC, http://www.wfmc.org), has a good potential for process interchange and heterogeneous system integration since it is used today by more than 80 different products to exchange process definitions and keeps up to date with BPMN 2.0.

The XPDL metamodel that we used for our project is shown in Figure 2. The *Package* concept represents a set of processes and contains:

- *pools*, which represent major participant roles in a process, typically separating different organizations. A pool can contain:
    - *lanes*, which are used to organize and categorize activities within a pool according to function or role.
- *workflowProcesses*, which aggregate sets of activities and transitions
    - *activities* are represented by rounded rectangles and correspond to the execution of a task or to the functionality of a gateway, which can be:
        - *XOR* gateway (exclusively one of the outgoing transitions will be followed), which is represented by a diamond shape with the "X" character in the middle
        - *parallel* gateway (all the outgoing transitions lead to activities that will be executed in parallel), which is represented by a diamond shape with the "+" character in the middle
    - *events* are represented by circles and are specific kinds of activities that correspond to something that happens. Common events are the start of a process lane and its ending
    - *transitions*, are represented with a solid line and arrowhead and have source and target (at the arrowhead) activities and define the control flow in the workflow process
- *associations*, are represented with a dotted line and arrowhead and have source and target (at the arrowhead) activities and define the message flow between different pools. Therefore, they also have source and target pools.

3 The Transformation Algorithm

The transformation algorithm uses elements from the liveness formulas grammar (Listing 1) and the XPDL metamodel (Figure 2). It is a recursive algorithm that takes the liveness formula expression elements from left to right and applies the templates shown in Figure 3, gradually building the XPDL process. For all templates, the control flows from left to right, i.e. if a template follows another, then it is connected to its rightmost element. The algorithm is provided in pseudocode at the appendix.

Regarding the theoretical properties of the algorithm we believe that it can be easily proved that it is correct using induction and the assumption that if we have a correct XPDL model and replace an XPDL activity with a correct XPDL fragment (or a *well-structured fragment*, as in [20]) the resulting model is correct. The templates are all correct XPDL diagrams (well structured fragments) if they have a *start* event on their left and a transition to an *end* event on their right, as every task is on a path from the start event to the end event. Then, for each of these valid models we can easily assert that if we take a random template and replace an activity of the model with it then, again, the model is correct. Then, we hypothesize that after *n* insertions the model is correct and we insert a new random template. Then we show again that the resulting model is correct.

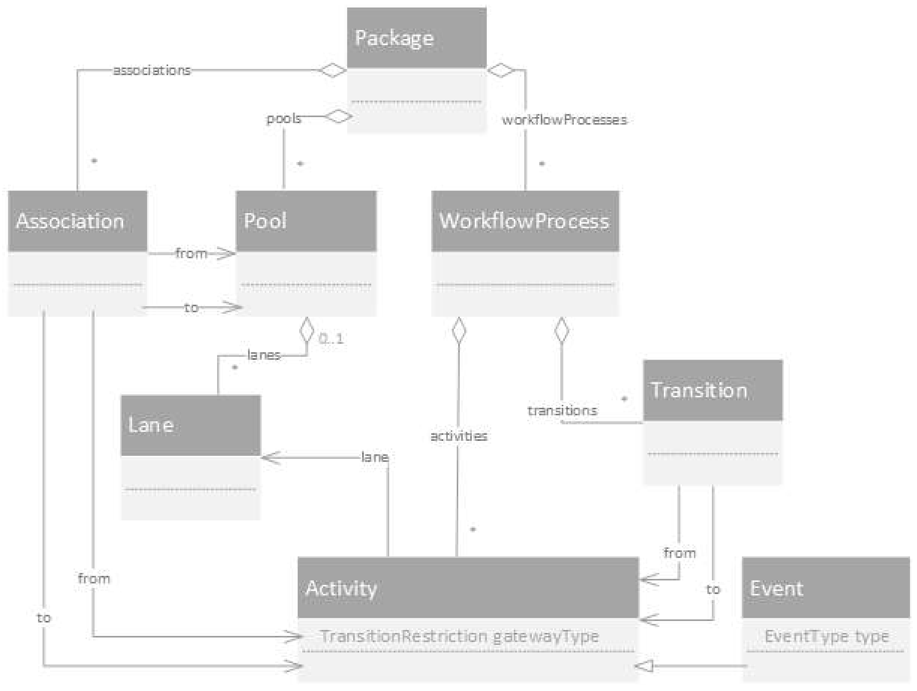


Fig. 2. The XPDL metamodel[1]

The reader should note the common templates for the ~ and + operators. Considering the semantics of the ~ operator the exclusive gateway should not be used (the activity should just loop back to itself). In this way, the resulting process model would not be easily ported to existing analysis techniques as it would not pass the Proper Completion test (each workflow ends with an end event) [21]. Given the fact that in a later stage the situation could be remedied by adjusting the gateway to always return the flow to the activity, and that in the second version of Gaia there is a case where the authors allow the indefinite operator to be followed by a sequential activity [9], we believe that our approach is the best compromise for this case.

As far as the algorithm's complexity is concerned, since we have a recursive function call inside a for loop, the complexity of our algorithm is $O(n^2)$, where *n* is the number of activities and protocols present in the liveness formulas. The algorithm

[1] We used the *org.enhydra* Java package defining the metamodel for XPDL 2.1, which is distributed under the GNU Free License by Together Teamsolutions Co., Ltd. It is downloadable from `http://tinyurl.com/org-enhydra`

would run forever should there be circular references to *LeftHandSide* from a formula's *Expression* (or from subsequent formulas), however, we have a pre-processing step guarding against this possibility and preventing the algorithm from executing.

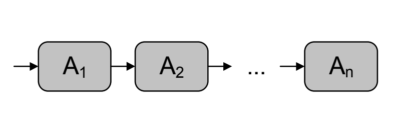
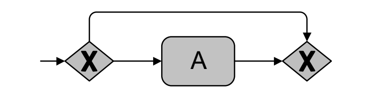
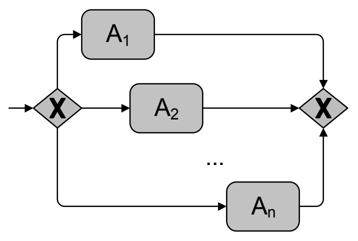
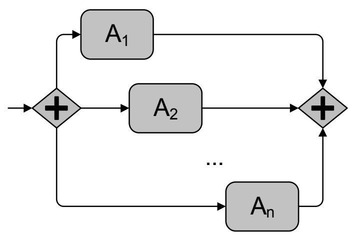
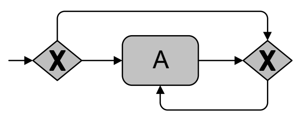
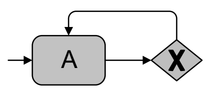
Op.	Template	Op.	Template
$A_1.A_2.\ldots.A_n$		[A]	
$A_1\|A_2\|\ldots\|A_n$		$A_1\|\|A_2\|\|\ldots\|\|A_n$	
A*		A~ A+	

Fig. 3. Templates of liveness formula (Gaia) operators (Op.) for XPDL model generation.

4 The Liveness2XPDL Tool

The tool allows defining one or more agent roles. For each role, the user can edit a liveness formula or import a role model. We researched for the Gaia methodology and its derivatives' metamodels to create the relevant import functionality. We found documented metamodels for the Gaia [22], ROADMAP [23] and the ASEME [7] methodologies. However, Gaia's metamodel abstractly defines the LivenessProperty class and ROADMAP's metamodel file is not available on-line. Thus, we created an importer for the ASEME System Roles Model (SRM) metamodel to demonstrate the capability of our approach in importing meta-models. Since our tool is open source, interested developers can create an importer for the metamodel they prefer or they can type their formulas in the text editor.

The tool allows integrating multiple roles in the same XPDL model. We create one *Pool* instance for each role in a common *Package* (the transformation algorithm executes as many times as the participating roles with the same Package instance) and then the user defines the associations for message sending and receiving activities.

Then, the tool creates the needed references of the associations to the pools and outputs the Package in XPDL format.

In this section we demonstrate the usage of the developed tool. We consider a real world system developed in the context of the ASK-IT Integrated Project[2] where a personal assistant agent on a lightweight device (e.g., a smart phone) requests services from a mediator agent (or broker). This broker has the capability to service simple requests but can also access a complex service provider agent who can offer high level services. The complex provider also needs simple services from the broker in order to compose a high level service. In our case, we consider a route calculation service that can be simple (I want to get from point A to B with a car using the quickest route) or complex (I want to get from point A to B with the best transport means according to my user's impairment needs and habits). In the second case the complex provider will reason on the type of simple request based on the user's profile, make a simple route calculation service request to the broker and then sort the results according to the user's habits before replying to the user through the broker.

The agent roles models for the personal assistant and the broker are presented in Figure 4 (just the role name and liveness property). The complex provider is the same with the one presented in Figure 1.

Role: PersonalAssistant

Liveness:

PA = SendServiceRequest. ReceiveServiceResponse

Role: Broker

Liveness:

Broker = (ServicePAs || ServiceCP)+

ServicePAs = ReceiveServiceRequest. ProcessRequest. (InvokeDataManagement |
    SendComplexServiceRequest. ReceiveComplexServiceResponse). SendServiceResponse

ServiceCP = ReceiveSimpleServiceRequest.
    InvokeDataManagement. SendSimpleServiceResponse

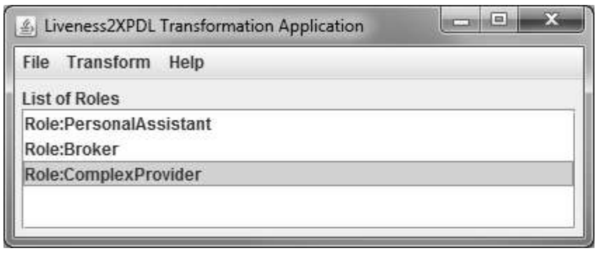
Fig. 4. The Personal Assistant and Broker role models.

The user starts the Liveness2XPDL tool and imports through the *File* menu the three role models, as presented in Figure 5. Then, the user can select one role and the *Single role transformation* option from the *Transform* menu, or more than one (holding down the control key) and the *Multiple role transformation* option from the *Transform* menu. In Figure 6 the reader can see the single role file for the Complex Provider role.

In the case of multiple roles transformation, the tool then prompts the user to select where to save and how to name the output XPDL file. If there are activities that send

[2] ASK-IT has been co-funded by the European Union under the 6th Framework Programme (no IST-2003-511298)

or receive messages the graphical interface presented in Figure 7 helps the user to create message flows.


Liveness2XPDL Transformation Application

File Transform Help

List of Roles

Role:PersonalAssistant
Role:Broker
Role:ComplexProvider

Fig. 5. The main screen of the Liveness2XPDL tool.

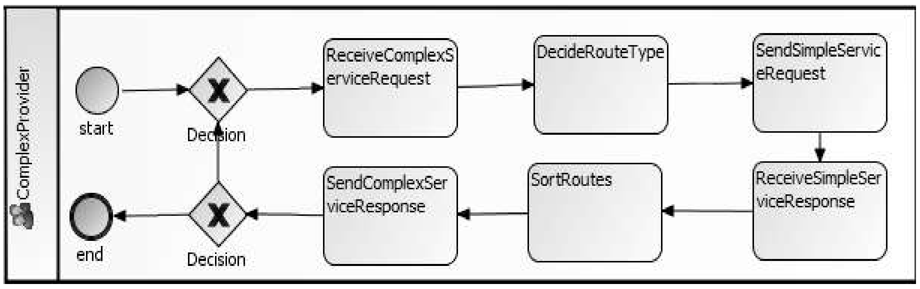
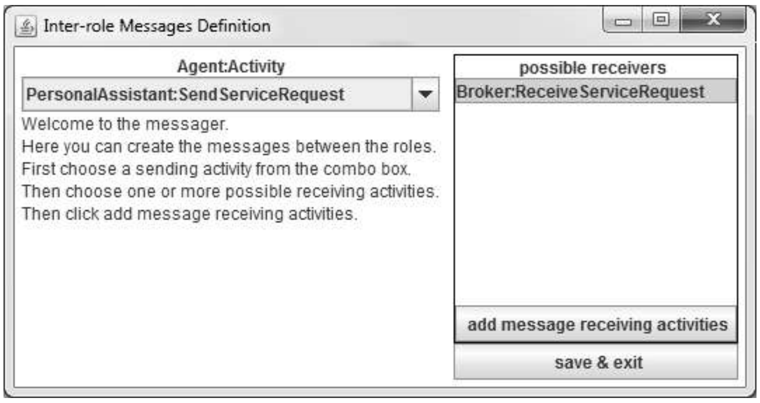


Fig. 6. The Complex Provider displayed in Together Workflow Editor[3].


Inter-role Messages Definition

Agent:Activity

PersonalAssistant:SendServiceRequest

Welcome to the messager.
Here you can create the messages between the roles.
First choose a sending activity from the combo box.
Then choose one or more possible receiving activities.
Then click add message receiving activities.

possible receivers

Broker:ReceiveServiceRequest

add message receiving activities

save & exit

Fig. 7. The Inter-role Messages Definition screen.

[3] A graphical Workflow Editor implementing XPDL specification V2.1 using the BPMN graphical notation, `http://www.together.at/prod/workflow/twe`

Finally, in Figure 8 the reader can see the combined roles process model for all the roles used in our project. The modeler has used the graphical tool depicted in Figure 7 to define the message flows between the agents. A message flow represents the flow of information between two separate roles (pools). The screenshot in Figure 8 has been taken from the Signavio tool[4]. To import the model into the Signavio tool we first used a free online XPDL to BPMN conversion service[5].

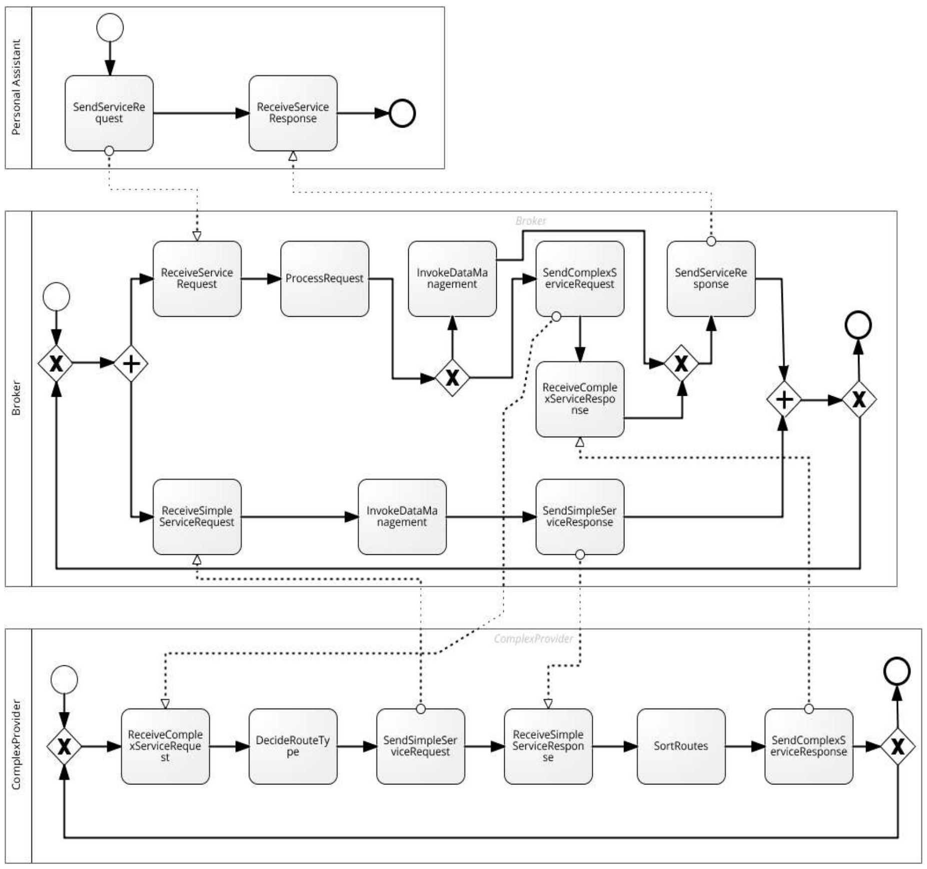


Fig. 8. The three agent roles displayed together in Signavio BPM Academic Initiative.

[4] The BPM Academic Initiative of Signavio offers a web-based process modeling platform to students, lecturers and researchers, `http://www.signavio.com/bpm-academic-initiative`

[5] E.g. the "Convert XPDL to BPMN" service provided freely on-line by Trisotech, `http://www.businessprocessincubator.com`

5 Simulating The Roles Interactions

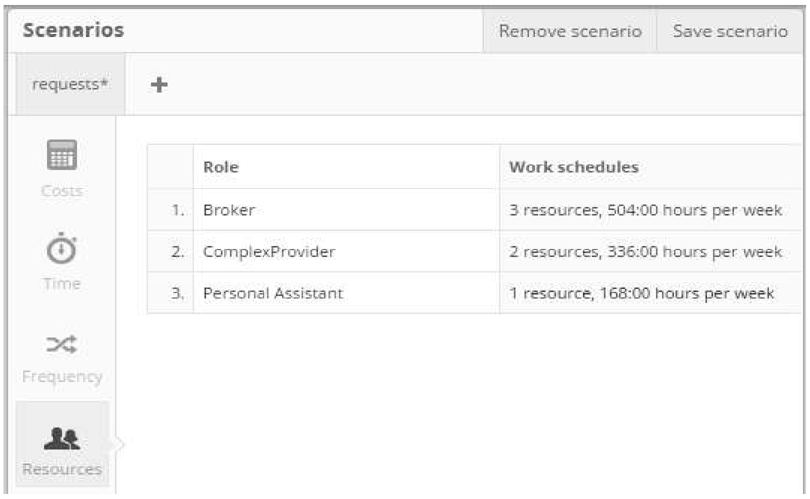
In this section, we demonstrate how simulation can aid the system modeler as well as the project manager to make important decisions, mainly concerning non-functional requirements.

Initially, there were two reasons for simulating the ASK-IT system. The first was that the ASK-IT service providers needed to know if the system can satisfy non-functional user requirements, one of which was the delivery of the service within ten seconds. The frequency of service requests was calculated to be one request per 30 seconds. The second was to find out how the system would scale when service demand increased. The latter would be used for preparing the project's exploitation plan.

The Signavio tool allows simulating a process model involving several roles. For each simulation scenario, it allows to define:

- available resources for each role (how many instances of this role are available)
- the frequency in which a role can appear and start executing
- the percentage of times that a XOR gateway selects one or the other execution path
- activity duration (distribution type, mean and standard deviation values)
- number of simulations for each scenario

For our simulations we used several executions of function prototypes to define the activities durations. Moreover, we added the network latency in the message receiving activities. All the distributions that we used are Gaussian (Normal). Then, we defined different scenarios by varying the frequency of PAs appearing in the network and asking for services, the number of brokers serving the requests and the number of complex providers (in Figure 9 you can see a screenshot from the Signavio tool for defining a scenario).



	Role	Work schedules
1.	Broker	3 resources, 504:00 hours per week
2.	ComplexProvider	2 resources, 336:00 hours per week
3.	Personal Assistant	1 resource, 168:00 hours per week

Fig. 9. Defining the scenario in the Signavio tool.

Our experiments are presented in Figure 10. We have validated the system to respond within 10 seconds in the worst case when we have an incoming request every 30 seconds with one broker and one complex provider. Moreover, we can see what the expected quality of service will be, while the requests' frequency rises. As far as system scaling is concerned we see that by adding more broker instances, the system performance has a better gain than by adding complex providers. Finally, we can claim that with three broker instances the system can offer the required quality of service (respond within ten seconds) even if we have a request every two seconds.

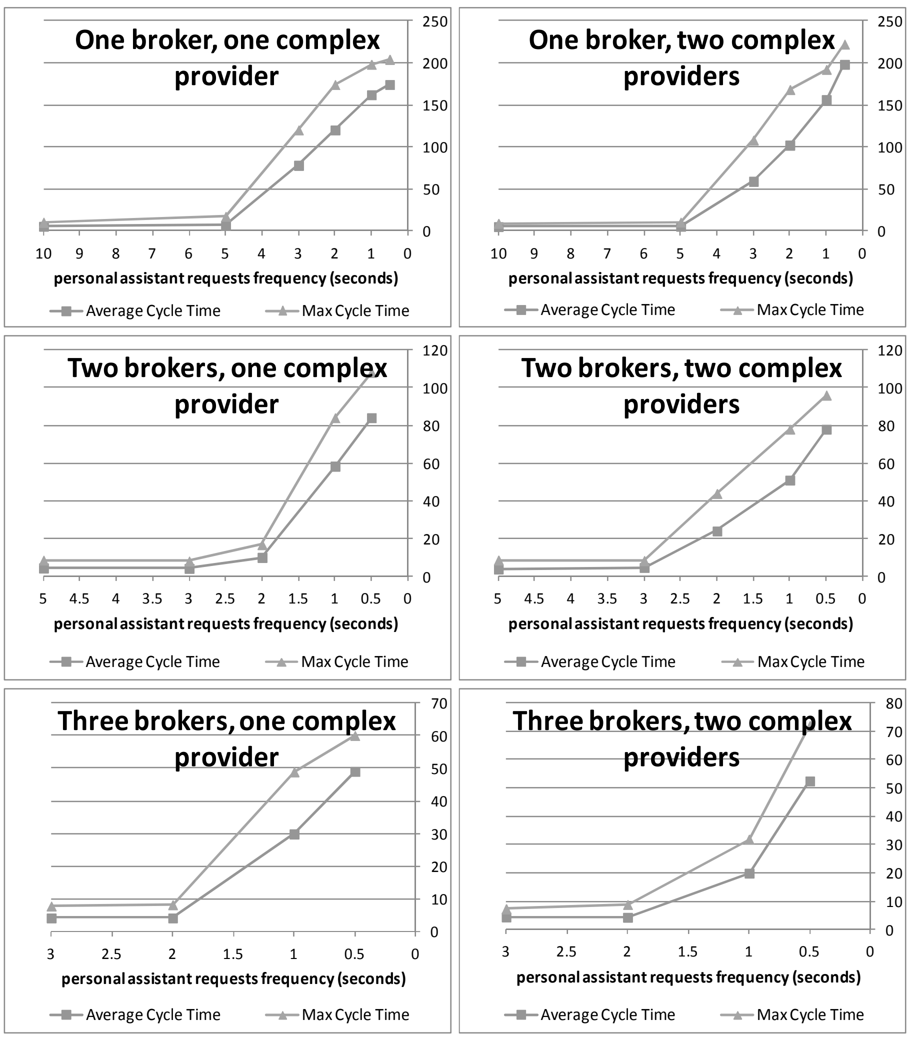


Fig. 10. Average and maximum response times in seconds (vertical axis). The horizontal axis represents the time interval between two requests (in a normal distribution).

6 The Method Fragment for Validating the Analysis Model

Method fragments [24] are reusable methodological parts that can be used by engineers in order to produce a new design process for a specific situation. This allows a development team to come up with a hybrid methodology that will support the needs of specific programming and modeling competencies.

The method fragment that corresponds to the process of validating an analysis model is presented in this section. It is defined as a software development process using the extended SPEM 2.0 language for representing agent oriented methodologies [25]. A *Software Process* is defined as a series of *Phases* that produce *Work Products*. In each phase simple or complex *tasks* take place. Tasks are achieved by *Human Roles*. Work products can be either graphical or textual models. Graphical models can be *Structural* (focusing in showing the static aspects of the system – such as class diagrams) or *Behavioral* (focusing in describing the dynamic aspects of the system – what happens as time passes). Textual models can be completely *Free* text or follow some specifications or grammar (a *Structured* work product).

Each process package defines a process that contains tasks connected through dashed arrows like in flowcharts. The black dot shows where the process starts and the black dot in a circle where it ends. A task has input and output work products. An arrow from a task to a work product means that the product is created (or updated) by the task. An arrow from a work product to a task means that the product is an input to the task.

This method fragment (shown in Figure 11) can be integrated with the Gaia methodology or one of its descendants by the software engineer. It is activated at the end of the analysis phase (or architectural design phase for Gaia 2.0) to validate the system model. Its inputs are the various activities execution times (average and standard deviation) and the (Gaia) role models (the liveness property).

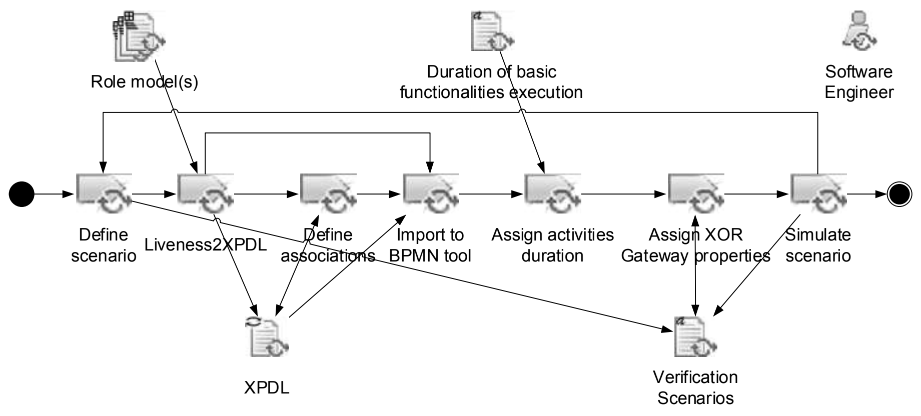


Fig. 11. The analysis model validation method fragment in SPEM 2.0.

The engineer must first define the scenario for validation (first task). The scenario is written in free text (the "Verification Scenarios" work product). Then the relevant roles are selected and transformed to process models in the Liveness2XPDL task using the developed tool. Optionally, using the same tool, in the next task, namely "Define associations", the engineer connects the message sending and receiving activities of the roles. The XPDL work product is automatically produced and in the next task it is imported to the desired tool that will be used for simulation (in our case study Signavio). Then, the engineer assigns the activities duration and XOR gateway properties using the same tool. Finally, the engineer simulates the scenario. The process package finishes by updating the "Verification Scenarios" with the results of the simulation or is restarted to simulate a new scenario.

7 Discussion

It is not the first time that the AOSE community studies and uses business process models. There are a number of works, e.g., one for improving a process model representing the behavior of agents [11], another for proposing a method for transforming BPMN models to agent-oriented models in the Prometheus methodology [26], and another that provides a mapping of BPMN diagrams to a normalized form checking for certain structural properties, which normalized form can itself be transformed to a petri-net that allows for further semantic analysis [27].

All these works can be aligned with ours using method engineering and provide a number of new paths or possibilities for a system modeler that has come up with the Gaia analysis models. Thus, an AOSE practitioner can transform the process model outputted from our work to a system specification using the Prometheus methodology notation [26] and continue using that methodology. Another might be interested in checking certain structural properties of the process model [27].

Some preliminary results of this work have appeared in EUMAS 2010 (with informal proceedings) [28]. In that work, we provided transformation templates targeting the BPMN v1 metamodel. This work extends that one by targeting the XPDL metamodel, which offers a wide range of possibilities when available tools are concerned. Moreover, this work caters for integrating multiple roles in a single process model.

Although we have achieved our goals, the Liveness2XPDL tool has specific limitations. Firstly, when the user decides to create multiple associations that define message flows from an activity that will be received by different activities in other pools the method cannot automatically tell whether one of the possible paths will be followed, or all of them. The inter-agent messages definition interface allows defining such associations; however, it is not clear how these can be exploited with simulation.

An important note to the transformation approach concerns the templates' definitions. Undoubtedly, there is not a single way to express a concept with XPDL (or the BPMN notation). For example, the *A*~ formula can be represented either with the template illustrated in Figure 3, or by adding the loop symbol in the rectangle. Although some good styles and practices are in use today, in practice there are no rules that guarantee an optimal design. The appropriateness of the model must every time get validated by the end user. In our case, the templates were defined considering the BPMN simulation tools features. For example, for the *A*~ formula, we chose that particular definition because the loop symbol would introduce sub-processes to the model, and available simulation tools have limited support for such a feature.

Moreover, in XPDL it is acceptable to create more than one transition from an activity to other activities. This option reduces the complexity of the model as it is not mandatory to use XOR gateways. However, a large number of process management tools do not accept this option and most of the times they suggest that a gateway should be placed to avoid errors. This is why we used the XOR gateway in our templates.

Finally, after the process model is produced, the user still has to provide some additional elements concerning the send/receive activities' configuration. We are currently working towards automating this step based on the following guidelines (which are now manually configured):

- All activities that stand for sending or receiving messages are labeled as message type activities.
- When a receive activity immediately follows a start event, then the start event and the activity are merged into a start event triggered by a message.
- When a receive activity immediately precedes an end event, then the two are merged into an end event triggered by a message.
- When a message is intended to be sent to one or more out of many recipients and this decision has to be evaluated during runtime, then before the "send message" activity a data-based exclusive gateway is added.

8 Conclusion

In this paper we showed how a development team that employs the Gaia methodology, or its derivatives, i.e. ROADMAP [4], the Gaia2JADE process [5], Gaia4E [6] and ASEME [7] can transform the output of the analysis phase model (Role Model) to a process model. Actually, the role's liveness property is used for the transformation.

Process models are useful paradigms as they, on one hand, allow the usage of a wide range of tools (free or proprietary) for simulation, thus providing the means to explore non-functional properties of the system under construction, even before its implementation. Therefore, project managers and engineers can evaluate the use of methods and technologies in their project, but also information about the deployment and scaling of their application. On the other hand, process models are commonly used by business stakeholders, who can now understand and appreciate a MAS analysis model. Finally, such models can be used to define agent and humans interactions based on the associations of the process model.

Herein, we presented the transformation algorithm, demonstrated the developed tool and showed how it can be used to validate a system analysis for a real world application, which was created in the context of ASK-IT project. The open Java sources and executable java jar file for the Liveness2XPDL tool can be browsed by the interested reader at github[6].

The approach that we followed has some limitations, but also opens interesting paths for future work. A very promising path lies in developing a code generation tool based on the process model and targeting the WADE[7] toolkit of the popular JADE platform. Another path is that of accommodating the definition of human-agent interactions in the modern field of Human-Agent Collectives [29], based on process models.

Appendix: The recursive transformation algorithm.

The pseudocode of the tranformation algorithm is presented below. The different model elements are represented as classes and their properties as class properties, accessible using the dot operator, i.e. *<classname>.<property>*. For representing a list we use a *List* class that supports the operations *add* (to add an element to the list) and *size* (to return the number of its elements). The program takes as input an XPDL Package instance and the String liveness property of an SRM Role instance.

```
Program transform(Liveness liveness, Package package)
 WorkflowProcess workflowProcess = new WorkflowProcess
 package.workflowProcesses.add(workflowProcess)
 Event startEvent = new Event
 startEvent.type = start
 workflowProcess.add(startEvent)
 Activity lastActivity = createProcess(liveness.formula1.expression,
workflowProcess, startEvent)
 Event endEvent = new Event
 endEvent.type = end
 workflowProcess.add(endEvent)
 Transition transition = new Transition
 transition.from = lastActivity
 transition.to = endEvent
 workflowProcess.add(transition)
End Program

Function Activity createProcess(Expression expression, WorkflowProcess
workflowProcess, Activity activity)
 List terms = new List
 For Each termi In expression
  terms.add(termi)
 End For
 If terms.size() > 1 Then
  If expression Is SequentialExpr Then
```

6 `https://github.com/ASEMEtransformation/Liveness2XPDL`

7 WADE is a software platform based on JADE providing support for the execution of tasks defined using the workflow metaphor, `http://jade.tilab.com/wadeproject`

```
   For Each $term_i$ In expression
    Activity newActivity = createProcess($term_i$, workflowprocess,
activity)
    activity = newActivity
   End for
  Else If expression Is OrExpr
   Activity xorEntryGateway = new Activity
   xorEntryGateway.gatewayType = XOR
   workflowProcess.add(xorEntryGateway)
   Transition transition = new Transition
   transition.from = activity
   transition.to = xorEntryGateway
   workflowProcess.add(transition)
   Activity xorExitGateway = new Activity
   xorExitGateway.gatewayType = XOR
   workflowProcess.add(xorExitGateway)
   For Each $term_i$ In expression
    Activity newActivity = createProcess($term_i$, workflowprocess,
xorEntryGateway)
    transition = new Transition
    transition.from = newActivity
    transition.to = xorExitGateway
    workflowProcess.add(transition)
   End for
   activity = xorExitGateway
  Else If expression is ParallelExpr
   //similar with orExpr, parallel gateway type instead of XOR
  End If
  For Each $term_i$ In expression
   If $term_i$ Is BasicTerm
    boolean foundLeftHandSideEqualsBasicTerm = false
    For Each $formula_i$ In liveness
    If $formula_i$.leftHandside = $term_i$ Then
     Activity newActivity = createProcess($formula_i$.expression,
workflowprocess, activity)
     activity = newActivity
     foundLeftHandSideEqualsBasicTerm = true
    End If
    If foundLeftHandSideEqualsBasicTerm = false
     Activity newActivity = new Activity
     workflowProcess.add(newActivity)
        Transition transition = new Transition
        transition.from = activity
        transition.to = newActivity
        workflowProcess.add(transition)
        activity = newActivity
     End If
    Else If ($term_i$ is of type '(' term ')' ) Then
```

```
     Activity newActivity = createProcess(term, workflowprocess,
activity)
      activity = newActivity
    Else If (term_i is of type '[' term ']')Then
     //definition of the [A] template
    Else If (term_i is of type '*') Then
     //definition of the A* template
    Else If (term_i is of type '~') Then
     //definition of the A~ template
    Else If (term_i is of type '+') Then
     //definition of the A+ template
    End If
   End If
  End For
 return activity
End Function
```

References

1. Rana, O.F., Stout, K.: What is scalability in multi-agent systems? In: International Conference on Autonomous Agents. pp. 56–63. ACM, Barcelona, Spain (2000).
2. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management. Springer Berlin Heidelberg, Berlin, Heidelberg (2013).
3. Wooldridge, M., Jennings, N.R., Kinny, D.: The Gaia Methodology for Agent-Oriented Analysis and Design. Auton. Agent. Multi. Agent. Syst. 3, 285–312 (2000).
4. Juan, T., Pearce, A., Sterling, L.: ROADMAP: extending the gaia methodology for complex open systems. In: Proceedings of the first international joint conference on Autonomous agents and multiagent systems part 1 - AAMAS '02. pp. 3–10. ACM Press, New York, New York, USA (2002).
5. Moraitis, P., Spanoudakis, N.: The GAIA2JADE Process for Multi-Agent Systems Development. Appl. Artif. Intell. 20, 251–273 (2006).
6. Cernuzzi, L., Zambonelli, F.: Gaia4E: A Tool Supporting the Design of MAS using Gaia. In: Proceedings of the 11th International Conference on Enterprise Information Systems (ICEIS 2009), Volume SAIC, Milan, Italy, May 6-10. pp. 82–88 (2009).
7. Spanoudakis, N., Moraitis, P.: Using ASEME Methodology for Model-Driven Agent Systems Development. In: Weyns, D. and Gleizes, M.-P. (eds.) Agent-Oriented Software Engineering XI. pp. 106–127. Springer-Verlag, Berlin, Heidelberg (2011).
8. Workflow Management Coalition: Workflow Standard Process Definition Interface - XML Process Definition Language, WFMC-TC-1025. (2008).
9. Zambonelli, F., Jennings, N.R., Wooldridge, M.: Developing multiagent systems: The Gaia methodology. ACM Trans. Softw. Eng. Methodol. 12, 317–370 (2003).
10. Pascalau, E., Giurca, A., Wagner, G.: Validating Auction Business Processes using Agent-based Simulations. In: Proceedings of 2nd International Conference on Business Process and Services Computing (BPSC2009), March 23-24, Leipzig, Germany (2009).
11. Szimanski, F., Ralha, C.G., Wagner, G., Ferreira, D.R.: Improving Business Process Models with Agent-Based Simulation and Process Mining. In: Nurcan, S., Proper, H.A., Soffer, P., Krogstie, J., Schmidt, R., Halpin, T., and Bider, I. (eds.) Enterprise, Business-

Process and Information Systems Modeling. pp. 124–138. Springer Berlin Heidelberg, Berlin, Heidelberg (2013).

12. Onggo, B.S.S.: BPMN pattern for agent-based simulation model representation. In: Proceedings Title: Proceedings of the 2012 Winter Simulation Conference (WSC). pp. 1–10. IEEE (2012).
13. Spanoudakis, N., Moraitis, P.: An Agent Modeling Language Implementing Protocols through Capabilities. In: 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology. pp. 578–582. IEEE (2008).
14. Wirth, N.: Extended Backus-Naur Form (EBNF), ISO/IEC 14977:1996(E). (1996).
15. Sendall, S., Kozaczynski, W.: Model transformation: the heart and soul of model-driven software development. IEEE Softw. 20, 42–45 (2003).
16. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: Eclipse Modeling Framework (2nd Edition). Addison-Wesley Professional (2008).
17. Redding, G., Dumas, M., Hofstede, A.H.M. ter, Iordachescu, A.: Generating Business Process Models from Object Behavior Models. Inf. Syst. Manag. 25, 319–331 (2008).
18. Jahangirian, M., Eldabi, T., Naseer, A., Stergioulas, L.K., Young, T.: Simulation in manufacturing and business: A review. Eur. J. Oper. Res. 203, 1–13 (2010).
19. Cibrán, M.A.: Translating BPMN Models into UML Activities. In: Ardagna, D., Mecella, M., and Yang, J. (eds.) BPM 2008 International Workshops, Milano, Italy, September 1-4, 2008. Revised Papers. pp. 236–247. Springer Berlin Heidelberg, Berlin, Heidelberg (2009).
20. González-Ferrer, A., Fernández-Olivares, J., Castillo, L.: From business process models to hierarchical task network planning domains. Knowl. Eng. Rev. 28, 175–193 (2013).
21. Van der Aalst, W.M.P.: The application of petri nets to workflow management. J. Circuits, Syst. Comput. 8, 21–66 (1998).
22. Bernon, C., Cossentino, M., Gleizes, M.-P., Turci, P., Zambonelli, F.: A Study of Some Multi-agent Meta-models. In: Odell, J., Giorgini, P., and Müller, J.P. (eds.) Agent-Oriented Software Engineering V. pp. 62–77. Springer Berlin Heidelberg, Berlin, Heidelberg (2005).
23. Juan, T., Sterling, L.: The ROADMAP Meta-model for Intelligent Adaptive Multi-agent Systems in Open Environments. In: Giorgini, P., Müller, J.P., and Odell, J. (eds.) Agent-Oriented Software Engineering IV. pp. 53–68. Springer Berlin Heidelberg, Berlin, Heidelberg (2004).
24. Cossentino, M., Gaglio, S., Garro, A., Seidita, V.: Method fragments for agent design methodologies: from standardisation to research. Int. J. Agent-Oriented Softw. Eng. 1, 91 (2007).
25. Seidita, V., Cossentino, M., Gaglio, S.: Using and Extending the SPEM Specifications to Represent Agent Oriented Methodologies. In: Luck, M. and Gomez-Sanz, J.J. (eds.) Agent-Oriented Software Engineering IX. pp. 46–59. Springer (2009).
26. Dam, H.K., Ghose, A.: Agent-Based Development for Business Processes. In: Desai, N., Liu, A., and Winikoff, M. (eds.) Principles and Practice of Multi-Agent Systems. pp. 387–393. Springer Berlin Heidelberg, Berlin, Heidelberg (2012).
27. Endert, H., Hirsch, B., Küster, T., Albayrak, S.: Towards a Mapping from BPMN to Agents. In: Huang, J., Kowalczyk, R., Maamar, Z., Martin, D., Müller, I., Stoutenburg, S., and Sycara, K.P. (eds.) Service-Oriented Computing: Agents, Semantics, and Engineering. Springer Berlin Heidelberg, Berlin, Heidelberg (2007).
28. Delias, P., Spanoudakis, N.: Simulating Multi-agent System Designs Using Business Process Modeling. In: 8th European Workshop on Multi-Agent Systems (EUMAS 2010). , Paris, France (2010).
29. Jennings, N.R., Moreau, L., Nicholson, D., Ramchurn, S.D., Roberts, S., Rodden, T., Rogers, A.: Human-Agent Collectives. Commun. ACM. 57, 80–88 (2014).