A Technique for Defining Agent-oriented Software Engineering Process Models with Tool Support

Iván García-Magariño a Alma Gómez-Rodríguez b
Juan C. González-Moreno b, 1

a Department of Software Engineering and Artificial Intelligence, Universidad Complutense de Madrid, Madrid, Spain
b Department of ..., Universidad de Vigo, Orense, Spain

Abstract

The processes of development in software engineering are fundamental when referring to costs and quality of products. Moreover, the variety of software engineering processes in Multi-agent System developments implies the necessity of formalizing software engineering processes. This work proposes a technique for defining process models for agent-oriented developments. The technique presented is provided with a supporting tool. The tool was generated from a software process engineering metamodel, by following a Model-driven approach resulting, in a low cost maintenance tool. This work formalizes two agent-oriented process models: the Unified Development Process and the SCRUM process, both of which are within the context of INGENIAS methodology. Finally, the presented technique and supporting tool are compared with other existent alternatives.

Key words: Software Process Engineering Metamodel, Multi-agent System, Model-driven Development

1 Introduction

The Agent Oriented paradigm has shown its suitability in modeling and developing huge, complex and distributed systems (Xiang and Lee, 2008; Posadas et al., 2008; Negenborn et al., 2007). Therefore, the Agent Oriented approach has become an important field for investigation in Systems Development and, 1 The author names are alphabetically ordered.
particularly, in Software Engineering. In this discipline, software quality assurance is based on methodology definition and usage. All these, jointly with the increasing interest in Multi-Agent Systems (MAS), have resulted in the definition of several Agent Oriented semiformal methodologies (Pavón and Gómez-Sanz, 2003; O’Malley and DeLoach, 2002; Cuesta et al., 2002; Bernon et al., 2005; Mas, 2004).

Nowadays, in the field of quality assurance, one of the more relevant lines of work is the study and improvement of processes for software development and maintenance. The foundations of this field lie on the direct relation between process quality and final product quality. This encourages the necessity for obtaining models for development processes; a software process model defines the structural and behavioral issues of a process, by using a Process Modeling Language (PML) (Breton and Bezivin, 2001). A fundamental challenge in software process modeling is to find a standard PML for process definition. For MAS based process modeling, the FIPA Methodology Technical Committee has suggested the use of Software Process Engineering Metamodel (SPEM), which is defined by Object Management Group (OMG). SPEM was introduced as a UML profile for software development process modeling. This standard has a stable version 2.0, which adapts the original proposal to the UML-2.1.1 standard (it can be found in http://www.omg.org).

The idea of distinguishing among Agent Oriented Software Engineering (AOSE) methodology processes is not new. Some works define a metamodel of the development process. For instance, Henderson-Sellers and Gonzalez-Perez (2005) compare four process metamodels and provide a new generic standard, for obtaining a global process of development, with independence of the chosen methodology. Cernuzzi et al. (2005) provide a survey of the AOSE methodologies and classifies the most relevant AOSE methodologies according to their process models. For this taxonomy, the paper relates directly each methodology with a type of development process. Following that classification, this paper provides references to AOSE methodology descriptions with their process model descriptions. A similar approach is introduced by Penserini et al. (2007), in which processes and methodologies are described jointly.

The proposal presented in next sections addresses this issue with a different perspective. It is based on considering that a methodology can be applied using different development processes depending on several factors (such as the kind of project, costs, development team, etc.) and by also adapting human resources to new phases or activities. In any case, aspects defined by the methodology like the entities of the model or the roles implied will not suffer any change. Taking this hypothesis into account, the development team uses a particular methodology that may dynamically change the development process to readapt the development to new costs or time restrictions.
Some authors (Cernuzzi et al., 2005; Cossentino et al., 2006) indicate that the modelling of processes for agent-based development is an increasing demand for many reasons. Some of the AOSE methodologies (Pavón and Gómez-Sanz, 2003; O’Malley and DeLoach, 2002; Cuesta et al., 2002; Bernon et al., 2005; Mas, 2004; Padgham and Winikoff, 2002; Chella et al., 2006) may follow different process models depending on the system to be constructed. In these cases, the anticipated definition of the process, by means of its modelling, is useful for the right agent-based development. Besides, the process models can be shared among Multi-agent System (MAS) designers; in this way, the process models defined by experienced designers will be provided for helping inexperienced developers. Moreover, defining the process models provides the basis for automating the process itself, in the same way as it is done in other engineering fields, and opens the possibility of customizing CASE tools for a particular process inside a methodology.

This work presents a technique for defining agent-oriented process models. In this technique, a software development process is considered, at a high level of abstraction, as a simple dependency graph with three basic components: the process participants (roles or workers), the consumed and generated products (work products) and the activities and tasks achieved during the process, which constitute particular instances (work definitions) of the works that must be done. Nevertheless, most software process developments are so complex that they require additional guidance or suggestions on what activities must be done and when such activities are to be completed. Despite this simplistic interpretation of development processes, it is worth noting that process development modeling, like software development, presents several orthogonal and complementary views. Each of these views provides a partial explanation of the whole process and allows a gradual definition.

Taking the aforementioned technique into account, this paper models some of the processes that can be followed for agent-based development. This definition is simplified by using an editor which allows designers to create the suitable diagrams and definitions in an automated manner. Until the moment, several editors, such as APES (2008); EPF (2008); Cossentino et al. (2006), have been proposed, but they fail in satisfying all the necessities of the agent-based development or when describing the technique for defining a process using the editor. For this reason, this paper presents both an open-source editor tool and a technique for defining process models. Although the main goal of the editor tool and the technique is to define agent-based development processes, since they are based on a general-purpose standard (SPEM (Group, January 2005)), they can be used in the definition of process models for any software development.

The remaining of the paper is structured as follows. The next section describes a technique for defining process models, and Section 3 presents the
editor tool which supports the technique. Then, Sections 4 and 5 contain two process models, which are defined with the presented technique. These process models are the INGENIAS Unified Development Process (UDP), and a SCRUM process for INGENIAS methodology. Section 6 measures and compares the presented tool and other existent tools. Finally, Section 7 indicates the conclusions and future work.

2 Technique for Defining Process Models

Cernuzzi et al. (2005) research the need of process models for agent-based development has been established. After this work, many others have modelled different kinds of processes within a well known methodology. From authors’ point of view, processes and methodologies are in two orthogonal dimensions that can be merged or interchanged during the life cycle of a MAS development process. This could be done if, previously, development resources and deliverables are defined and modelled into a well known process or if, a new process is created to match the necessity of the development team for fulfilling the user requirements.

In the latest years some tools have been proposed to model and specify Software Development Processes. Most of these are based on the SPEM standard, like EPF, for instance. As part of the EPF documentation, two possible development processes are described: OpenUp and Scrum. Moreover, the authors propose an example expressed with the EPF Composer Overview, based on several activities defined for OpenUp into a Scrum process skeleton. On the other hand, in (Cernuzzi et al., 2005; Henderson-Sellers and Giorgini, 2005) this situation is approached trying to find a set of common activities for different Agent Oriented Software Engineering Methodologies (AOSEM). The objective, in this case, is to reuse those activities across real Multi-Agent Software Development Life Cycle Processes.

For authors, process activities can not be shared across different development processes in AOSEM because the intended meaning is different and those activities use different resources (i.e. human, software and hardware). Nevertheless, different processes for the same methodology which use the same resources can be used. This section addresses this goal. That is, to show how an agile project management technique such as Scrum can be used within a methodology like INGENIAS; whose development process nature is basically those of the Rational original Unified Development Process (UDP) and which has been recently mapped to the OpenUP process (García-Magariño et al., 2008).

This section presents a technique for defining process models. The editor tool
support, which is presented later in Section 3, has been used as a tool during specification. The framework is based on the development of three orthogonal views of the systems: the view of lifecycle process, the disciplines used in the process view and the guidance and suggestions view. The first view proposed describes the path which must be followed for obtaining as a final product, a MAS, jointly with the activities that must be accomplished along the path. The second view establishes the disciplines of the process, the roles and participants and the tasks to perform, which will consume and produce a set of products. Finally the guidance and suggestions view details products, works and roles which constitute model elements of the other views.

When trying to map a well established methodology/process into a new process, it is necessary to define the steps (the process) in order to obtain the desired result. Several steps that must be followed in the definition of a new development process model for AOSEM are defined, by means of adopting SPEM as the model specification. These steps are:

1. **Identify the process model with an existent process model** if possible. If not, define the new process from the beginning, using the next steps as a basis. Before defining a new process model, it is useful to take the existent process models into account. Some of the most known process models are matched (Cernuzzi et al., 2005) with some AOSE methodologies. The matching between methodology and process must be a good point of departure for starting the definition. Nevertheless, it is worth mentioning that a methodology may use several process models depending on the MAS.

2. **Define the lifecycle view.** Identify the phases, iterations and sequence of application. This step is essentially a temporal step in which other resources that differ from time are not considered.

3. **Define the disciplines.** Disciplines in SPEM determine process packages which take part in the process activities, related with a common subject. That is, disciplines represent a specialization of selected subactivities, where these new subactivities can not appear in other packages or disciplines. In this step, resources are the subject of the activities defined.

4. **Define the guidance and suggestion view.** The Guidelines provide information about certain model elements. Each Guidance is associated to a Guidance Kind. This step is focused in exemplifying and documenting the activities previously defined.

The cases of study selected for introducing this model guide are the Unified Development Process of INGENIAS (Gómez-Sanz, 2002; Gómez-Sanz and Fuentes, 2002) and the SCRUM-INGENIAS. The first process is defined as the integration of INGENIAS metamodels and the Unified Development Process (UDP) (Jacobson et al., 1999), in what refers to analysis and design disciplines. SCRUM-INGENIAS process integrates INGENIAS methodology with
2.1 Lifecycle

The specification is started by modelling the particular lifecycle wanted for the development. This definition allows the analyst to forget about participants and products to obtain and focus on the definition of the activities to accomplish in each step. The development path is divided in consecutive phases and each of them in a different number of iterations depending on the phase.

Using the editor described in the previous section, the representation of the iterative and incremental lifecycles is made in several steps. To begin with, the elements (phase, iteration and activities) which constitute the lifecycle are defined. Initially, a child of Specification of kind PLCProcess Lifecycle must be created. Afterwards, the analyst must define an entity of Lifecycle type, an entity for each phase of Phase type and one entity of Iteration type, which will include in its name the minimum and maximum number of allowed iterations for each phase. The following step is the definition of the existing temporal and membership relationships among these elements. Figure 1 presents a snapshot of the editor when doing the iteration definition for a phase of lifecycle for a possible INGENIAS process model. Figure 2, on the other hand, shows how the precedence relationships among lifecycle, phases and iterations are defined creating the children entities Dep Rel of type Precedes.

Once the previous activities are done, it is necessary to define for each iteration the phase it belongs to, and which activities must be performed.

After having specified the development process along with the activities to perform in each iteration, the next step is to define the disciplines implied in this activities formalization.

2.2 Disciplines

Disciplines in SPEM determine process packages which take part in the process activities, related with a common subject. That is, disciplines represent a specialization of selected subactivities, where these new subactivities can not appear in other packages or disciplines. The definition of these disciplines is
made by creating a child of PC Ent type (Process Component entities) type within Specification. Next, all the created disciplines are included within it by selecting new entities of Discipline type.

The participants definition is made using the different roles that each participant will play along the project and the products, as a basis. In the editor, the mechanism for including roles and products is to create, in PS Ent, entities with types Proc Rol and Work Prod, respectively.

Once the participants, products, disciplines and process are defined, each activity must be divided for each iteration depending on the task pertaining to each discipline. The activities must be performed by any of the specified participants consuming and/or producing any of the existent products. The process for creating this step is similar to the previous ones. However it includes tasks like Step within PS Ent and relations between activities and task like a PS Rel of RAc Step type. The relationships for making an activity by a participant will be defined using relations PS Rel of type RAss. Activities and consumed and/or produced products can not be related directly; so this relationship is established using the discipline where it is used and the role which manages it as intermediates.

2.3 Guidances

The Guidances provide information about certain model elements. Each Guidance is associated to a Guidance Kind. SPEM provides a basic repertoire of Guidance Kinds, but it allows one to define new kinds.

In the editor, the Guidance and GuidanceKind elements can be added to the BE Ent element (entities of the Basic Elements SPEM package). Each Guidance must be associated with the RGuidK relationship in the BE Rel element. Each Guidance must also be associated with a Model Element, using the relationship RAnnot in the BE Rel element. Guidances are linked with work products which are model elements. Each guidance has its own External Description, which contains a complete explanation of the guidance. The Ext Descrip element can be added to the BE Ent element to define an external description. The guidance can connect with its external description using the RPresentation element in the Core Rel element (relationships of the Core SPEM package). Each guidance (guideline or technique) is linked to an external description (element Ext Descrip). The guidance external description has an attribute named content with the guidance content (see Figure 6). It also contains an attribute indicating the language; in the example, English is selected.

In multi-agent systems, we specially recommend using two kinds of guidance:
Technique and Guideline. The technique provides an algorithm to create a work product. The guideline is a set of rules and recommendations about a work product organization.

3 Editor Tool Support for the Technique

As mentioned before, the technique for defining process models presented in this paper is supported by an editor tool, which will be described in this section. The editor is based on SPEM and it is built using the Domain-specific Model Languages (DSML) (Amyot et al., 2006) and the Model-driven Development (MDD) (Atkinson and Kuhne, 2003) approaches. The use of DSML and MDD approaches make the tool specially suitable for MAS modeling, because it allows savings in development costs and makes it easier the tool adaptation to the changes in methodologies and/or development process. Following Model-driven Development (MDD) approach, first a language model (metamodel) is defined and, next, a framework generates the editor from the metamodel. In this way, the process of obtaining the editor is reduced to the proper definition of the metamodel.

Currently, there are several metamodeling languages available. Some of them are MOF, EMOF, CMOF, SMOF, ECore, KM3, CORAL M3, Microsoft DSL language and GOPRR. In order to decide among so many languages, there are two important facts to consider. First, the SPEM specification (Group, January 2005) uses MOF (Group, January 2006) language. Second, the ECore language is used by the most reliable tools and frameworks. In particular, ECore is supported by Eclipse Modeling Framework (EMF) (Budinsky, 2003; et al, 2004) and EMF can generate an editor automatically from an ECore metamodel. Therefore, we defined the SPEM metamodel with ECore, from its definition in MOF, and the SPEM editor was automatically generated. In addition, this editor serializes their SPEM models using XMI documents, which are widely supported.

To summarize, the editor proposed in this paper facilitates the definition of software process models with a graphical user interface, and the process of the editor creation makes it possible a low-cost maintenance. Next subsection describes the process of the editor tool creation.

3.1 The Tool Creation Process

In Figure 3, the adopted phases for creating the presented tool are shown. These phases are: specification, design, implementation, testing and mainte-
The design phase is composed of three stages. The first stage is the selection of the metamodel language (metametamodel). The second stage is the extraction of the structural features of the model language, SPEM. The third stage is the definition of the metamodel. The implementation phase has a low cost since it is computerized. The testing and maintenance phases provide feedback. When feedback is provided, the iteration has to start from the metamodel definition. Further details of each phase are mentioned in this section.

The goal of this work is to obtain a tool for defining software process engineering models. The specification of the model language can be defined with a metamodel. In particular, the Software Process Engineering Metamodel (SPEM) (Group, January 2005) is selected for the specification. SPEM is defined by the Object Management Group (OMG) which is an international computer industry consortium. The OMG goal is the definition of standard specifications. Its memberships include hundreds of organizations. Thus, the SPEM specification provided by OMG is considered appropriate as the specification of our tool.

In our approach, the tool is automatically generated from the metamodel. Thus, the selection of the metamodel language (metametamodel) and the selection of the framework for generating the tool is relevant in the design phase. Amyot et al. (2006) provides an evaluation of the frameworks of developing tools for modeling languages. According to that work, the ECore model is selected as a metametamodel. EMF is the selected framework for the tool generation.

SPEM is defined with MOF by OMG. MOF is another metamodel language. The translation from MOF to ECore can be done in several ways. The extraction of the structural features of the model language defined by SPEM is necessary. The way of defining the metamodel with ECore has effects on the generated tool. Thus, the stage of metamodel definition must consider the implications of the metamodel in the generated tool. For this reason, the original metamodel must be adapted.

The implementation of the tool is automatically achieved. In fact, the tool is automatically generated by EMF.

However, sometimes the results of the automatic generation are not the expected ones. So the testing of the generated tool shows that some changes may be necessary. In this case, feedback is provided to the design phase, some changes are done in the definition of the metamodel and the tool is generated again.
When the tool is considered stable enough, the process reaches the maintenance phase. The tool is provided to the user. In particular, the tool was used for defining the INGENIAS (Pavón and Gómez-Sanz, 2003) processes. INGENIAS is a methodology for developing multi-agent systems. In the definition of INGENIAS processes, some problems were found in the tool. These problems are remedied in the maintenance phase. In the maintenance phase, the user indicates problems and also suggests improvements. This user feedback mandates returning back to the design phase where some changes are done in the metamodel. The process starts again from the metamodel definition.

Next subsections describe the most relevant details of each phase of the tool editor creation process.

3.2 SPEM Structural Features.

For defining the metamodel, the structural features inherent to SPEM have to be considered. The definition of the metamodel has effects on the tool and model serializations. The SPEM structural features are mentioned below.

**Feature 1** Entities with attributes. *In SPEM, most of entities have attributes. WorkDefinition and Process entities are examples of entities with attributes.*

**Feature 2** Relationships with attributes. *For example, the relationship Precedes includes the enumerate attribute called PrecedesKind.*

**Feature 3** Relationships with attributes in their ends. *The Association relationship has several attributes, like isNavigable or multiplicity.*

**Feature 4** N-ary relationships. *ParentWork, for example, is a many-to-many relationship which connects a set of WorkDefinition entities to a set of ParentWork entities as their children.*

**Feature 5** Several kinds of relationships. *For example, some of the relationships in SPEM are Precedes and Association.*

**Feature 6** Entities hierarchy in the metamodel. *This hierarchy uses the attributes inheritance making the definition of new entities more robust and easy. The definition of the same attributes is not repeated; instead, the attributes are inherited. For example, the ModelElement entity is extended with entities such Namespace, GeneralizableElement or Parameter. In addition, the Classifier extends from the GeneralizableElement entity.*

**Feature 7** Relationship Hierarchy in the metamodel. *This hierarchy also uses the attributes inheritance. It has the same advantages as the entities hierar-
chy. For example, SPEM contains an abstract relationship called Relationship. This abstract relationship is extended with several relationships such as Generalization, Association and Dependency.

The aforementioned structural features are considered in the definition of the metamodel presented in the next section.

3.3 Definition of the Metamodel for the Editor Tool

This section describes the software process metamodel definition, from which the tool was generated. This metamodel representation is based on the classification of the model elements into entities and relations. This classification of the model elements is suitable for Connection-based model languages (Costagliola et al., 2002), and SPEM model language is one of these. In addition, the ECore language (Budinsky, 2003) is selected as metamodeling language because of its technical support, such as the Eclipse Modeling Framework (EMF) (Budinsky, 2003). Originally, SPEM (Group, January 2005) was defined with MOF, and its translation to ECore language requires to make certain decisions, which make it possible to obtain an editor with automatic generation (see the description of the metamodel definition in Appendix A).

Due to the particularities of the ECore language and the structural features of SPEM, the entities and relationships of SPEM are represented with certain ECore elements. For instance, each relationship is modeled with an element for the body of the relationship and another element for each relationship-end. In addition, certain meta-modeling elements were defined for structuring the SPEM elements according to the package classification of SPEM. Further details of the definition of this metamodel are included in Appendix A.

4 Case of Study: INGENIAS with UDP process

In order to test the utility of the technique and the editor introduced in this paper, they have been applied in the definition of several AOSE processes. This section and the next one describe how the technique and the editor have been used in two particular processes description.

In this section, the Unified Development Process of INGENIAS (Gómez-Sanz, 2002; Gómez-Sanz and Fuentes, 2002) is addressed. The process is defined as the integration of INGENIAS metamodels and the Unified Development Process (UDP) (Jacobson et al., 1999). UDP distributes the tasks of analysis and design in three consecutive phases: Inception, Elaboration and Construction,
Fig. 4. Activities corresponding to Inception Phase

with several iterations (where iteration means a complete cycle of development, which includes the performance of some analysis, design, implementation and proofs tasks). The sequence of iterations leads to the procurement of the final system.

4.1 Lifecycle

In INGENIAS, the specification is started by modeling the particular lifecycle selected for the development. The development path is divided in consecutive phases and each of them in a different number of iterations depending on the phase. For defining restrictions about the admissible number of iterations, the methodology suggests to add in the iteration name the range for the number of iterations, which are the maximum and minimum number of allowed iterations (for instance [1..3]). That will work if the number is always the same. In other instances, it is mandatory to particularly specify each of the different iterations.

Using the editor described in the previous section, the representation of the iterative and incremental lifecycle is made in several steps. In first place, the elements (phase, iteration and activities) which constitute the lifecycle are defined. Initially, a child of Specification of kind PCLProcess Lifecycle must be created. Afterwards, the analyst must define an entity of Lifecycle type, four more entities, corresponding to each phase, of Phase type and one entity of Iteration type, which will include in its name, as previously explained, the minimum and maximum number of iterations allowed in each phase. The following step is the definition of the existing temporal and membership relationships among these elements. Figure 1 presents a snapshot of the editor when doing the iteration definition for a phase of lifecycle.

For instance, Inception phase in the described process implies the accomplishment of the activities shown in Figure 4.

Once the previous activities are done, it is necessary to define the phases and activities for each iteration. The process of representing the sequence of these primary activities is done by introducing the activities as entities and, after, establishing the precedence creating the children entities Dep Rel of type Precedes. Finally, the member relationship between activities and the iteration when they are performed and between iterations and phases must be defined. This is done by creating a new child entity of PS (Process) type and, within it, another one with type PS Rel (Process Relationship). Next, relationships of type RParent W must be added, indicating the activity and
Fig. 5. Guidance Entities. This figure contains the entities related to the Guidance. These entities are the Guidance, Guidance Kinds, External Descriptions and the Work Products. The relationships are not shown.

Fig. 6. Guidance External Descriptions. The attributes (content, language, medium and name) of the selected External Description are shown in the Properties tab. In this case, the reader can see the beginning of the external description of the Model-Driven technique.

iteration to which they belong.

After having specified the development process along with the activities to perform in each iteration, the next step is to define the disciplines implied in this activities formalization.

4.2 Disciplines

As previously outlined, disciplines in SPEM determine process packages which take part in the process activities, related with a common subject. In the study case used in this section, the disciplines which are included in the Development Process are Requirements Specification, Analysis, Design and Implementation. The definition of these disciplines is made creating a child of PC Ent (Process Component Entities) type within Specification and the four disciplines are included by selecting new entities of Discipline type.

In the next step, the methodology suggests that, before detailing the subactivities (tasks) which constitute each discipline, the analyst indicates the participants. This definition is made using as a basis the different roles that each participant will play along the project and the products used. In the selected case study, the implied roles are: Analyst, Architect, Developer, Tester, Project Manager and FreeMan and the products used are the following ones: Use Case Diagram, Environment Model, Organization Model, Interaction Model, Agent Model, Object and Task Model, Glossary, NonFunctional Requirement and Stakeholder Vision.

4.3 Guidance

The Guidance provides information about certain model elements. In multi-agent systems, we specially recommend to use these two kinds of guidance: Technique and Guideline. The technique provides an algorithm to create a work product. The guideline is a set of rules and recommendations about a work product organization.
The most relevant INGENIAS Guidance are described below, and the reader can see their definition on Figure 5.

- *General Technique*. It indicates how to develop a complete INGENIAS specification, so that the multi-agent system, generated from the specification, runs itself. This technique gives the algorithm for developing a General work product. The general product is supposed to be any kind of product.

- *Model-Driven Technique*. This guideline recommends the user to follow the Model-Driven Development (Atkinson and Kuhne, 2003) principles. Pavón et al. (2006) explicitly explains how to develop an INGENIAS MAS driven by the model. Basically, the user edits the specification using the IDK editor. This specification is the model on which the development is based. The code generator obtains the Java code for Jade platform from the specification. In the specification, the user can add some pieces of code. This concrete piece of code remains on the specification so that, the next time the system is generated, the code is not overwritten. In addition, working only on the specification (model) fastens the software process. This technique is recommended for all the work products, represented by the General work product.

- *Remote Communication Guideline*. It provides rules and recommendations to organize a work product focused on remote communication. The INGENIAS 2.6 distribution provides an example, named IAF-cinema, that follows this guideline. This guideline is applied only to the work product named RemoteCommunication.

- *Referee Pattern Guideline*. It provides a referee-pattern INGENIAS MAS organization. An agent plays the role of a referee. This agent is in charge of controlling the communication among the other agents and stopping the multi-agent system when necessary. The delphi multi-agent system (García-Magariño et al., 2007a) follows this guideline.

5 Case of Study: INGENIAS with SCRUM process

This section shows how to specify and define a process model which adapts the Scrum Development Process for being used with INGENIAS Methodology (Gómez-Sanz, 2002; Gómez-Sanz and Fuentes, 2002).

5.1 SCRUM

A Scrum is a mechanism in the sport of rugby for getting an out-of-play ball back into play. The term was adopted in 1987 by Ikujiro Nonaka and Hirotaka Takeuchi to describe hyper-productive development. Jeff Sutherland developed
the Scrum process in 1993 while working at the Easel Corporation and Ken Schwaber formalized the process in the first published paper on Scrum at OOPSLA 1995 (Sutherland, 1995).

As pointed in (Schwaber and Beedle, 2001), Scrum is an empirical Agile project management framework which is used to iteratively deliver to the customer software increments of high value. It is a simple framework used to organize teams and get work done more productively with higher quality. Scrum relies on self-organizing, empowered teams to deliver the product increments. It also relies on a customer and the Product Owner, who provide the development team with a list of desired features using business value as the mechanism for prioritization.

For the aforementioned reasons, it must be clear that SCRUM is a model for management of the process of software development. It is not a methodology, but a framework which can fit with several methodologies. The SCRUM process is particularly suitable for Knowledge Engineering Developments based on the use of Multi-Agent Systems.

After applying the previous steps to IAF and Scrum, the results can be synthesized in the following points:

(1) **Process Model:** There are several Multi-Agents Systems that could be quickly constructed with INGENIAS by reusing previous developments. Recently, the INGENIAS Agent Framework (IAF) for JADE has been proposed and documented as a successful approach in this context. In Scrum, the production of a software version (release) is usually done in a couple of months. Potential release features are collected and prioritized in the product backlog. The product owner is responsible for updating the backlog.

(2) **Lifecycle:** In Scrum, each release is produced within a number of iterations from 2 to 4 weeks called Sprints (see Figure 7). The Sprint goal is defined by the product owner, taking both priorities and team capabilities into consideration. At the end of each Sprint, the team produces a product increment which is potentially releasable. The evaluation of the product release drives to a backlog update before the next sprint starts. All the work is done in two basic phases: the *Preparation Phase* and the *Sprint Phases* (see Figure 8).

(3) **Disciplines:** The team defines required *tasks* (Daily Scrum, Initiate Product Backlog, Manage problems, Plan Release, Sprint Planning, Sprint Retrospective, Sprint Review, Update product backlog) to develop selected features for the sprint. Within a sprint, progress checkpoints are performed during Daily Scrums. This enables the ScrumMaster to figure
work progress in regards to sprint goals and to suggest adjustments to ensure Sprint success.

(4) **Guidance and suggestion view:** The guidance for how to implement the process with INGENIAS could be found in the IAF documentation (Gómez-Sanz, 2008).

All the aforementioned technique steps are further explained in the following subsections.

5.2 **Lifecycle**

Although, Scrum does not describe engineering activities required for product development, INGENIAS-Scrum process must do it in order to adjust to IAF recommendations. IAF allows combining the classic approach of coding applications with modern techniques of automatic code generation. In this latest case, the generated MAS works over the JADE platform and additional tools available for this framework can also be applied.

According to the IAF documentation, applying MDD implies the necessity of a correct IAF specification. This requires the following aspects to be determined in a sound manner:

- **Interactions.** Interactions describe how agents coordinate. Coordination is defined in terms of information transfer units.
- **Tasks.** Tasks are the basic units of agent’s behavior. They modify the agent mental state and perform actions over applications.
- **Agents.** Agents are the main building block of the system. An agent is defined completely when its perception (main tasks) and its coordination means are described.
- **Deployment.** Deployment expresses how agents are instantiated and initialized in the final system.
- **Tests.** Tests describe the testing units the MAS should pass.
- **Goal.** The agents pursue certain goals. Goals guide the agent behavior.
- **Mental state.** The tasks performed by agents depend on their mental state and the pursued goals.

IAF requires the use of the INGENIAS Development Kit (IDK). IDK contains a graphical editor for working with the model specification. According to IDK, the scrum definition for the Preparation Phase (see Fig. 9) comprises the tasks: *Initiate Product Backlog, Plan Release* and *Preparation Tasks*.

The **INGENIAS Product Backlog** contains the product requirements which
are established with the IDK. This process can be done by adapting a known model from a previous project (i.e. IDK-IAF distribution comes with a complete cinema project which can be used for other distributed e-commerce developments in which the user needs to access the information and/or description of the product jointly with their location, check the products with their particular preferences and then decide to buy or not to buy the product) or defining a completely new product backlog with the editor.

After this initial model is completed, in the Preparation Tasks, the Scrum Master and the Product Owner establish the Plan Release in which the needed Sprints are defined.

To complete the initial product backlog and the preparation tasks, it must be taken into account that an agent in IAF performs a simple deliberation cycle:

- **Identify new tasks to schedule and tasks to remove from the schedule.** A task is scheduled if it satisfies a pursued goal and the input-task facts are available at the mental state of the corresponding agent.
- **Execute one scheduled task.**

The cycle previously described, omits the classic perception step. This step is not needed because the arrival of new information at any moment does not cause any kind of internal conflict for the agent. The incorporation of new pieces of information does not imply a change in scheduled tasks, although it may mean the incorporation of new tasks. These changes are done with the IDK, mainly modifying the Agent, Interaction and Task and Goal Models mainly.

From the Scrum perspective and taking into account that IAF bases on the automatic generation of code approach, the project team must be completely involved in the consecution of the release within the planned sprints. So, the INGENIAS specification must be established with the IDK as the core of the development. From this core, the different scripts and sources will be automatically produced.

Nevertheless, in the first stage, the generated code for the tasks may be incomplete. Thus, the programmer should add, if necessary, programming code in the tasks. This code must be incorporated in the specification by means of the IDK or the CodeUploader \(^2\) application. In this manner, the next code generation does not overwrite the added code. Furthermore, for some MAS, external components may be necessary, and the IDK provides a mechanism to integrate these components into the MAS. Also, a developer may find it

necessary to modify already generated code. In order to modify the generated code, in each sprint (see Fig. 10 and Fig. 11) the following tasks must be performed: Plan Sprint, Update Product Backlog, Daily Works, Manage Problems, Conduct Scrum Daily Meeting, Review Sprint, and Conduct Retrospective.

5.3 Disciplines

As previously pointed, in the INGENIAS-Scrum approach the disciplines are the required tasks in each sprint. In this subsection, the intended meaning of each task is explained in accordance with the IAF. Firstly, according to the guidelines proposed in our previous research (García-Magariño et al., 2008), the following roles and products involved in the development must be introduced:

- **Product Owner**. This role must be played by an active customer as in eXtreme Programming (XP).
- **Scrum Master**. The coach and main decision-maker in the development team.
- **Scrum Team**. This is a collective role that must be played by all of the team members.
- **Stakeholder**. Anyone that does not directly participate in the project but can influence the product being developed; thus, an interested party.

The products or artifacts involved in a Scrum development process are: Product backlog, Sprint backlog and Product increment. The product backlog contains the product requirements and has the purpose of listing all the functionalities to implement from a customer perspective. The sprint backlog is the list of things to do from the development teams point of view. It could be understood as fine-grained planning on detailed tasks. At last, the product increment is a partial product obtained at the end of each sprint, which can be deployed in the production environment or simply made available to users.

From the INGENIAS-Scrum perspective those artifacts are referred to the INGENIAS model and JADE code produced in each release. An INGENIAS model documented with the IDK could accomplish a lower or higher level of detail. Also, in the description of each model the Scrum Master could fix the work to be done in the next release, where release could be identified with the package entity of the INGENIAS model.

The tasks that must be performed during the development life cycle are:
- **Initiate Product Backlog.**- Explained in the previous subsection.
- **Preparation tasks.**- Explained in the previous subsection.
- **Plan Release.**- Explained in the previous subsection.
- **Sprint Planning.**- It involves the following steps: *Define the sprint’s goal, Select the items, Identify tasks from items, Estimate tasks, Assigning tasks, Getting team commitment.*
- **Update product backlog.**- The update should take sprint scope changes into account for planning the next sprints. It must be performed in three steps: *Collecting changes, Reprioritizing items and Reestimating items.*
- **Daily Scrum.**- The team performs backlog tasks to reach sprint goal in a free way, each scrum team selects the tasks to be performed or modified.
- **Sprint Retrospective.**- It allows the feedback and adaptation of the process.
- **Sprint Review.**- It shows what has been done and works.
- **Manage problems.**- This means to take into consideration the events that happen at every moment in a project and try to solve them. It is relevant with respect to last steps of the guideline presented in the next section.
- **Release work.**- This latest Sprint prepares product release.

All these tasks could be matched with different guidelines appearing in IAF documentation (Gómez-Sanz, 2008). The next subsection will present a brief description of the full life cycle.

### 5.4 Guidances

Developing a system with code generation facilities requires some discipline. In IAF documentation (Gómez-Sanz, 2008) several guidelines for development are proposed. For instance, the following one refers to the iteration process which obtains a correct JADE code, with the following steps:

1. Creating the project in the folder Project.
2. Creating the specification and storing it in the Project/spec folder of the project.
3. Modifying the project properties so that it points at the right places of the hard disk.
4. Filling in the specification with the MAS definition solving your problem.
5. Generating code from the specification. This step will imply modifying the specification to provide missing information the code generator requires.
6. If necessary, adding additional code in the Project/src folder of the project.
7. If necessary, modifying some generated files in the Project/permsrc folder. These modifications should not delete or modify automatically generated functions.
8. Compiling and executing the code.
6 Comparison to Other Related Works

The presented technique and tool are qualitatively and quantitatively compared with existent techniques and tools. In fact, the authors of this work did not found explicit descriptions of how to use the editors for defining process models. That means that only the tools and the language used for creating the editor metamodel can be used for the comparison.

6.1 Qualitative Comparison of the tools

Although there is an increasing interest in process modelling and standardization, encouraged by FIPA committee (delle Ricerche, 2008), there are still few tools that allow one to define development processes. Among these, this section refers to APES (APES, 2008), EPF (EPF, 2008) and Metameth (Cossentino et al., 2006) as the most relevant tools in the context previously exposed.

APES, whose current version is APES2, is software for process modelling. APES2 is, according to its authors, a process modelling software which follows the SPEM specification written by the OMG. The tool has several advantages: it is in conformity with a standard, enables the user to validate a process component, enables the user to easily modify a model and it is free (expensive and proprietary solutions can be avoided). In fact, APES has been split into four tools: a modelling tool (APES2), a presentation tool (POG), a publication tool (IEPP) and an execution tool (PEACH). So, the comparison will focus on APES2 which addresses the same goal of the editor tool presented.

Initially, the INGENIAS process was modeled by using existent tools, in particular APES2. In Figure 12, an attempt to model several activities of the INGENIAS process is shown. The APES2 tool becomes invalid for this definition because it does not implement the full specification of SPEM. Moreover, it is not possible, for instance, to distinguish between packages and disciplines, which are a fundamental point for the adopted guideline. In Figure 13, the definition of packages in APES2 can be seen; there are no disciplines option.

APES2 software and technique uses the following entities: Activity, Work, Work Product, Work Product State and Process Role. In addition, APES2
Fig. 13. APES tool has no option for Discipline.

takes the following diagrams into account: Activity diagrams, Context diagrams, Work definition diagrams, Flow diagrams and Responsibility diagrams. As it can be easily noted, the tool does not cover completely SPEM standard. It lacks coverage for important features and entities, like discipline or guidance (these entities are covered by the tool presented in the paper). In addition, it has some limitations in respect to verification or consistency checking.

On the other hand, the work of the Process Framework Project (EPF) has two main goals: to provide an extensible framework for software process engineering and to provide exemplary and extensible process content for a range of software development. The first goal has been achieved by the creation of the Eclipse Process Framework (EPF), which is a tool for defining a customizable software process engineering framework.

EPF is a tool with coverage of the fundamental entities and diagrams for process definition. The definition of this tool is based on XML schemas, and in this sense is similar to the tool proposed in this paper but it differs in the structure of files generated. The editor tool proposed in this paper produces a XML document based in XMI standard for SPEM, while EPF does not follow XMI. In addition, EPF generates several XML documents organized in a directory tree, so the inherent structure is reflected in the tree of documents. However the editor tool proposed in this paper provides one structured standard XMI file (so the structure is enclosed in the hierarchical XML file).

An important characteristic of EPF is that it provides some integration with other software, for instance, it creates project schedules for Microsoft Project from project definition. However, a more powerful integration is needed. For this reason, the final goal of the tool proposed is to use the XMI document generated as the basis for feeding a CASE tool which will provide validation and guidance for software development. In this way the CASE tool will be adapted to the process of development defined for each particular system and will provide guidances to the user explaining what to do next, what the proposed scheduling for development will be, etc.

Recently a new tool for defining processes has been proposed. It is called Metameth (Cossentino et al., 2006) and has been constructed with a similar philosophy than the tool proposed in this paper; that is, following SPEM and using other tools for defining the process metamodel. The results are very interesting and the tool provides important functionalities for process definition. In this case, the tool seems to be useful and powerful, but there are few indications for how to use it for defining a new process. So the results of definition will rely on engineers experience.
6.2 Metrics for the Quantitative Comparison

Although most of process techniques and tools theoretically use SPEM for defining the metamodel, usually they only use a subset of SPEM. Considering that each tool use a particular subset of SPEM, this section compares the metamodels of the MLs that are used by the existent techniques and tools. In addition, the usability of the tools is also considered. Firstly, the metrics used in the quantitative comparison are described in the next subsections and, then, the comparison results are presented in Section 6.3.

6.2.1 Availability Metric

The goal of the availability metric (Garcia-Magarino et al., 2008) is to measure the appropriateness of a Modeling Language (ML) for modeling a particular problem domain. For the experimentations of this work, Multi-agent Systems (MAS) are used. The higher the value is, the better use the problem domain is making of the ML. A low value indicates the agent ML does not have concepts that the developer considers necessary.

The idea behind this metric is that, for a particular problem domain, some concepts are necessary and others are not. The availability metric measures the percentage of these necessary concepts that are contained in the ML. Therefore, the availability metric is defined with the ratio indicated in Equation 1. In this equation, $nc$ indicates the number of necessary concepts (metamodel elements) in the modeling of the particular problem domain; and $ncmm$ indicates the number of these necessary concepts that are actually contained in the metamodel. The considered concepts can be either entities, relationships or any kind of metamodel element.

$$availability = \frac{ncmm}{nc}$$ (1)

The set of necessary concepts for a particular problem domain must be adapted according to the ML, i.e., the user must adhere to what the metamodel contains. So, the user must decide which metamodel elements are necessary for solving the particular problem. This is the number of necessary metamodel elements ($ncmm$). Once this set is selected, the user must detect if any concept is missing for solving the problem. This is the number of missing concepts ($mc$). Then, it turns out the number of necessary components ($nc$) is the sum of $ncmm$ and the number of missing concepts ($mc$). Thus, Equation 1 can be converted into Equation 2 which can also calculate the availability measurement. The best result is obtained when there are no missing concepts.
\[ mc = 0 \] In this case, the availability measurement is the unity (100%).

\[ availability = \frac{ncmm}{ncmm + mc} \] (2)

This metric depends on the expertise of the user about the utility of the concepts of the ML for a particular problem domain. In order to reduce these variations, the presented framework strongly recommends the two following precautions. Firstly, for comparing certain MLs, the same person must evaluate the different MLs. Secondly, a large number of users must evaluate the collection of MLs to statistically reduce the mentioned variations. This statistical assumption is already applied to known successful empirical cost estimation methods such as COCOMO (Boehm et al., 1995).

### 6.2.2 Specificity Metric

The goal of the specificity metric (Garcia-Magarino et al., 2008) is to measure the percentage of the modeling concepts that are actually used for modeling a particular problem domain. If the value of this metric is low, it means there are many ML concepts not used for modeling the problem domain. Then, the scope of the ML is probably more general than necessary. On the contrary, if the value of the specificity metric is high, the scope of the tool is specific for the problem domain.

\[ specificity = \frac{ncmm}{cmm} \] (3)

The specificity metric is defined with the Equation 3. This equation introduces a new term \( cmm \) which represents the number of all the concepts in the metamodel. The specificity metric measures the ratio of the metamodel concepts necessary for a problem domain, divided by the whole number of metamodel elements. In the specificity metric, the best result is obtained when all the concepts of the metamodel are necessary (\( ncmn = cmm \)) for the corresponding problem domain. In this case, the specificity value is the unity (100%).

As a final remark, the multi-perspective MLs (Kingston and Macintosh, 2000) generally use more modeling concepts than other MLs. The multi-perspective MLs get low results for this metric only if some of the perspectives are not necessary. On the contrary, the multi-perspective MLs get high results if all the perspectives are necessary. In particular, the use of multi-perspective MLs is necessary for the process modeling. Actually, SPEM considers several perspectives of process modelling.
6.2.3 Usability Measurement

For measuring the usability of the presented tool, this work considers some of the attributes Nielsen (1990) considers for hypertext usability: easy to learn, efficient to use, and pleasant to user.

For the presented experiments, the learning time is measured, as metric of easy to learn attribute. The learning time is the time that users need to learn using the tool. This time depends on the user qualities, such as the intelligence, the expertise. Thus, for comparing several tools with this measurement, it is recommended to use the same group of users for measuring the comparison of learning time. Moreover, when comparing tools similar to each other, an user can increase his/her expertise by learning one tool, so the learning process of the second tool shorter because the expertise acquired. For this reason, is recommended that the order of learning the tools in the group of testers is balanced.

For measuring the attribute of efficiency of use, the time of defining a particular process model is measured. The same process model is used for measuring several tool editor, thus the comparison of this measurement is relevant. The less time is needed to define a particular process model, the more efficient of use is the editor tool.

Finally, the attribute of pleasant to user needs to be measured by user opinions. Thereupon, for this measurement, several users are requested to give a number out of five which represents the pleasure of use of each process editor tool.

6.3 Results of the Quantitative Comparison

This work compares the three following technique/editors: the editor presented in this paper, APES2 and EPF. This comparison uses the metrics indicated in the previous sections. Since the measurement of these metrics can slightly vary according to the human-being tester, a group of testers measure all these metrics. These group of testers was taken from two Spanish universities: Complutense University of Madrid and Vigo University. All the results presented in this paper are the average of the results obtained from each tester of the group.

For this comparison, the modeling language (described with metamodels) of the three technique/editors are compared. For this purpose, the three corresponding metamodels are measured with the metrics described before. The metrics need a battery of process domains to measure metamodels. These process domains are taken from the AOSE field. The processes for the experiment
are: the INGENIAS methodology with UDP process (in Section 4), the IN-
GENIAS methodology with SCRUM process (in Section 5), and the process
recommended for the Agile PASSI (Chella et al., 2006) methodology.

The results obtained in this experiment are shown in Table 1. Further details
of this comparison experiment are provided for the researcher community in
http://grasia.fdi.ucm.es, in the software section. Since it is really impor-
tant that an editor supports the elements needed by users, the availability is
the most relevant metric in this comparison. The availability measurement of
the presented editor is 100%, which indicates that all the elements the users
needed for the mentioned process domains are supported by the presented edi-
tor. Secondly, almost all the necessary elements are supported by EPF (86.7%
obtained for the availability measurement). For instance, the concept of steps
within the tasks (activities) is an example of missing concept in EPF. Finally,
only a few of the necessary elements were supported by APES 2 (26.5% of
availability). Thus, the best results are obtained for the presented editor.

By the same token, the specificity is a secondary-relevant metric, which obtains
high results when fewer unnecessary elements are supported in the tool. As
one can observe in Table 1, EPF obtains the highest result for specificity,
95.2%. On the contrary the presented editor obtains the lowest value, 47.1%.
The presented editor support many modeling elements; thus, in many cases,
there are unnecessary elements for certain process models and the specificity
measurement is low.

In short, the presented editor supports more process modeling elements than
other existent tools. Thus, on one hand, the availability of concepts for mod-
elling processes is the highest (100%) but, on the other hand, in most cases
only a small subset of the elements are actually used (47.1%).

Moreover, another important aspect is the learning time. Learning time is the
time approximately necessary to learn to use a tool. For this experiment, the
same of tested and group of users were considered to measure the learning

<table>
<thead>
<tr>
<th>Technique/tool</th>
<th>Our avail.</th>
<th>Our specific.</th>
<th>APES 2 avail.</th>
<th>APES 2 specific.</th>
<th>EPF avail.</th>
<th>EPF specific.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ingenias UDP</td>
<td>100%</td>
<td>44.1%</td>
<td>26.7%</td>
<td>80.0%</td>
<td>92.9%</td>
<td>92.9%</td>
</tr>
<tr>
<td>Ingenias SCRUM</td>
<td>100%</td>
<td>50.0%</td>
<td>23.5%</td>
<td>80.0%</td>
<td>81.3%</td>
<td>92.9%</td>
</tr>
<tr>
<td>Agile PASSI</td>
<td>100%</td>
<td>47.1%</td>
<td>29.4%</td>
<td>100.0%</td>
<td>86.7%</td>
<td>100%</td>
</tr>
<tr>
<td>Total Average</td>
<td>100%</td>
<td>47.1%</td>
<td>26.5%</td>
<td>86.7%</td>
<td>86.9%</td>
<td>95.2%</td>
</tr>
</tbody>
</table>

Table 1
Comparing Availability and Specificity of several Process Modeling Editors
Our Technique/tool | APES2 Technique/tool | EPF Technique/tool
---|---|---
**Average Learning Time** | 2h 10min | 45min | 9h 5min

**Average Time for Defining the INGENIAS-UDP process** | 1h 1min | 38min * | 2h 35min

**Average of users pleasure(out of 5)** | 4 | 4.5 | 1.5

* Taking the limitations of the technique/tool into account, which include that certain aspects of the process cannot be modeled.

Table 2
Usability Measurement for each Technique/tool

time for each tool, taking into account the mechanism of measurement indicated in Section 6.2.3. In addition, the time for defining the UDP process within INGENIAS methodology context (see Section 4) is used to compare the efficiency of the editor. Finally, a group of users were requested to provide numbers out of five that represent the pleasure of use of each process editor tool.

In Table 2, the average of the aforementioned measures is presented. Researchers can download further details of the measures from the software section in [http://grasia.fdi.ucm.es](http://grasia.fdi.ucm.es). As one can observe, the best result is obtained by APES2. Users learn quickly APES2 (45 minutes in average); APES2 is efficient to use, of the three evaluated editors, it needs the least amount of time to define INGENIAS-UDP process. Finally, users are very pleasant of its use (4.5 out of 5), since the graphical user interface is very user-friendly. Secondly, the presented editor obtains good results for the corresponding measures, although the results are not as good as the APES2 results. The tool is easy to learn (learning time of 2 hours and 10 minutes ). The presented editor is efficient to use(1 hour approximately for defining INGENIAS-UDP) and the pleasure of users is high (4 out of 5) because of its flexibility and its homogeneity. Finally, the worst usability measurements are obtained for EPF. It requires the longest learning time (9 hours and 5 minutes), the least efficient process definition (2hours and 35 minutes for defining INGENIAS-UDP) and low pleasant of use (1 out of 5).

On the whole, the following conclusions are obtained for this quantitative comparison. Although APES2 is easy to use, it does not support most of the necessary concepts of process models, so it should be discarded for most users. Then, both the presented editor and EPF can model processes and can
complement each other. For the deeply-detailed process models, EPF does not support all the necessary modeling elements, so the presented editor is preferable. For superficial-detailed process models, EPF and the presented editor are suitable; however the presented editor is easier to use. Thus, the presented editor is recommended for most cases.

7 Conclusions and Future Work

The definition of the software process engineering model of a MAS methodology makes it easier to learn and use the methodology. This paper provides a technique to define process models for agent-based development, using SPEM techniques. In particular, the technique is used for the definition of two different development processes for the INGENIAS methodology: UDP and SCRUM.

In addition, the paper also presents a Software Process modeling tool which implements completely the SPEM proposal from OMG. This process editor is based on Model-Driven Development and it is generated from a metamodel. Thus, the creation of this tool is cost-effective. This paper provides some details on the creation process which can be useful for the development of other modeling tools.

Furthermore, this work can guide a MAS team of developers through the steps of definition of a development process for MAS construction using an agent framework (the IAF) and different processes of development. The description provided in the paper about the recommended steps to follow, can simplify the definition of processes for non expert engineers. Moreover, the paper provides examples of how a methodology can incorporate several processes. That is, INGENIAS may follow a RUP process of development or can incorporate other specific processes over it, such as SCRUM.

In the future, other methodologies and processes can be defined using the technique and tool proposed in this paper. The models constructed in this way can assist the MAS designer in selecting the appropriate methodology and process model for a specific MAS, team of development or other circumstances.

Furthermore, definition of the models constitutes also the basis for comparing the defined MAS processes with others by means of certain metrics applied to the model.

Other important feature than can be addressed is to divide the model of the process in pieces, called fragments. Fragments from different processes can, after, be integrated to define a new process. This approach is one of the current
lines of investigation in the field (see Henderson-Sellers and Gonzalez-Perez (2005)). Authors consider that the technique and tool proposed in this paper can be useful for achieving this goal. Adding a new way of classifying elements in M1 layer, different from their type, will allow providing new organized packages containing elements in the model. After that, some repositories, which facilitate the definition of new software process models, can be created by the community. The new software process would be composed of pieces of software processes in the repository. This idea of repositories is similar to the work presented by Fisher Fischer et al. (2005).

Some issues related to the editor tool are also left for future work. The Graphical Editing Framework (GMF) can be used for adding a graphical user interface to the generated tool. This new interface can be friendlier than the current user interface.

A Definition of the Metamodel for generating the presented Editor Tool

This section describes some details about the definition of the metamodel, from which the presented editor tool was generated by means of the Eclipse Modeling Framework (EMF) (Budinsky, 2003). Practitioners can download the complete metamodel from section “additional material for papers” section in software section in http://grasia.fdi.ucm.es/.

In order to facilitate the understanding of the definition of the metamodel, the most relevant elements of the elements of the ECore meta-modeling language are enumerated:

- **EClass.** The single Ecore element that contains EAttributes or EReferences. The instances of EClass can extend other instances of EClass.
- **EAttribute.** The Ecore element that contains values of primitive types (integer, strings, characters, etc).
- **EReference.** It represents a binary relationship between two EClasses. The instance of the first EClass contains the instance of the EReference. The second EClass is the EType of the instance of the EReference. There are the two following kinds of EReferences:
  - **containment:** In the model, the XML element representing the container contains a new XML element. This XML element represents the referenced object.
  - **non-containment:** In the model the XML element representing the container just contains a path to the XML element representing the referenced object. The referenced object can be either in another position of the document or in another document.
• **EPackage.** It represents a package of elements of the metamodel.
• **EEnum.** It represents the enumerations as data types
• **EEnumLiteral.** It represents a possible value in an enumeration.

The representation of the editor-tool metamodel is based on the classification of the model elements into entities and relations. In this metamodel, an entity is represented by an **EClass**. Each attribute of the entities (Feature 1) can be represented by the **EAttributes** of the **EClass**. These entities are organized into a hierarchy. The EClass inheritance is used to represent this hierarchy. A single abstract entity on the top of the hierarchy is added. Its name is **GeneralEntity.** In this manner, some attributes all entities inherit can be defined. For example *id* or *description* attributes are defined for most of the entities.

According to the structural features mentioned in Section 3.2, the relationship representation is selected. For this selection, the work presented by García-Magariño et al. (2007b) is considered. That work associates sets of structural features with different relationship representations. Each relationship is represented by the elements which follow below:

- An EClass for the relationship body. Their EAttributes represent the relationship attributes. (Feature 2) There are several relationships (Feature 5). These relationships are classified into a hierarchy, with a top abstract relationship, called **GeneralRelation.** This provides a mechanism for defining attributes for all the relationships. For example, the attribute *label* is defined for all the relationships. In Figure A.2, some relationships of the **Foundation::Core** package are shown.
- An EClass for each end of the relationship. A relationship can have two or more ends. In other words, the relationships can be N-ary. The relationship-end attributes (Feature 3) are represented by the EAttributes of the EClass. The *multiplicity* attribute is an example of relationship-end attribute. Some relationship-end examples are shown in Figure A.3.
- Some containment EReferences to link the relationship body with the ends of the relationship. Each relationship can have any number of ends (Feature 4).
- Some non-containment EReferences to connect each relationship-end with an entity. The relationship links to entities previously created. Hence a single entity can be linked to several relationships. Each relationship-end has got an EReference for each possible entity type. In this manner, each relationship-end is constrained to a set of entities types. In the model-level, a relationship-end must connect only with a single entity. Thus, the other end EReferences must be set to *null.*
The specification contains all the elements of the model and structures them. As the editor tool is generated from the metamodel, the appropriate structure of the specification is key for the tool usability.

In the specification, elements are grouped according to the original SPEM packages. The packages are Core, Actions, State Machines, Activity Graphs, Model Management, Basic Elements, Dependencies, Process Structure, Process Components and Process Lifecycle.

In the ECore metamodel, the specification is defined with the following elements.

- An EClass called Specification. The instance of this entity represents the root of the model in M1. The Specification EClass has a containment EReference for each original SPEM package.
- An EClass for each original SPEM package. The name of these EClasses are the initials or abbreviations of the original SPEM package names. Each EClass representing a SPEM package has two EReferences. The first EReference points to an entities container of the corresponding SPEM package. The second one points to a relationships container of the corresponding SPEM package.
- An EClass for representing an entity container for each SPEM package. Each EClass has EReferences to the entities of the corresponding SPEM package.
- An EClass for representing a relationship container for each SPEM package. Each EClass has EReferences to the relationships of the corresponding SPEM package.

The aforementioned EClasses are situated in an EPackage called root.

The elements for the different SPEM packages are organized into EPackages with the same structure. For each SPEM package, the metamodel contains the following EPackages:

- An EPackage with the suffix entities. This EPackage contains the entities of the corresponding SPEM package.
- An EPackage with the suffix relations. This EPackage contains the relationships of the corresponding SPEM package.
- An EPackage with the suffix ends. This EPackage contains the relationship-ends of the corresponding SPEM package.

In Figure A.4, the EPackages for the SPEM packages are shown. Some abbreviations are used. For example, the EPackages beginning with sm denote the elements of the Foundations::State Machines SPEM package.
As mentioned before, all the entities, relationships and ends are represented with the same ECore element, the EClass. Keeping apart the entities, the relationships and the ends make it possible to distinguish these for the tool and the human beings. Readers not familiar with EMF may get confused by this representation. The most intuitive representation would be to use the EReference (which is the ECore element to link the EClasses) for representing the relationships. However, as mentioned in Section 3.2, SPEM relationships need attributes in the model-level. The EReference does not support the attributes in the model-level. Thus, besides the entities, the relationships also need to be represented with EClasses. The relationship-ends need to be represented with EClasses, as well, for similar reasons.

In conclusion, the structure of the models specification of the presented tool is similar to the SPEM structure. This implies that the usability of the generated tool increases, because the users are familiar with the SPEM specification (Group, January 2005).

In SPEM, some data types must be defined. In particular, some enumerations are necessary. All the enumerations are defined in a EPackage called data_types. An example of enumeration is the Aggregation Kind. The enumeration literals are ak_none, ak_aggregate and ak_composite. This enumeration definition with the EMF editor is shown in Figure A.5.

Acknowledgments

This work has been supported by the following projects: Methods and tools for agent-based modeling supported by Spanish Council for Science and Technology with grants TIN2005-08501-C03-01 and TIN2005-08501-C03-03 co-financed with FEDER funds and Grant for Research Group 910494 by the Region of Madrid (Comunidad de Madrid) and the Universidad Complutense Madrid.

References


URL http://www.lina.sciences.univ-nantes.fr/Publications/2001/BB01a


García-Magariño, I., Gómez-Rodríguez, A., González, J. C., 2008. Definition of


