Designing MAS Organizations with the support of a UML CASE tool

Massimo Cossentino[1], Carmelo Lodato[1], Salvatore Lopes[1], Patrizia Ribino[1], Valeria Seidita[2] and Antonio Chella[2]

[1] Istituto di Reti e Calcolo ad Alte Prestazioni, Consiglio Nazionale delle Ricerche - ICAR/CNR Palermo, Italy
{cossentino,c.lodato,lopes,ribino}@pa.icar.cnr.it
[2] Dipartimento di Ingegneria Chimica Gestionale Informatica Meccanica
Università degli Studi di Palermo, Italy
{seidita,chella}@dinfo.unipa.it

Abstract. The design of MAS organizations is a complex activity where a proper modelling notation may offer a significant advantage in enabling the conception of the best solution. The aid provided by a supporting tool significantly contributes to make the approach technically sound and it is a fundamental ingredient of a feasible strategy to the development of large MASs. This paper describes a notation for representing MAS organizations using the Moise+ metamodel. A UML profile and a specific CASE tool have been developed for supporting design production and automatic code generation. The proposed notation is applied to a classical write paper simulator example. Results include portion of the automatically generated code according to Moise+ specifications.

1 Introduction

Distributed and open systems are widely employed in the simulation and management of highly complex scenarios in dynamic environments. To this end, such systems should act in quasi-real time to changes occurring in the environment adopting the most suitable behaviour for reacting to the new conditions. Agents can provide a good way for solving complex problems because of their intrinsic nature. In fact they are very useful to both design and implementation levels [16][17].

The ability of simulating complex hierarchical organization provides further utility to the design of multi-agent systems (MAS from now on). In other words, organizations can be seen as a set of constraints [3] that rules the behavior of every single agent in a multi agent society.

The implementation of an organization in a MAS is normally decided at design time. The way in which a MAS may re-organize itself has then to be investigated from two different points of view, i.e. the design (methodological) and the implementation point of view. A robust approach to agent organizations comes from the work of Hubner et al. [9] where a definition of an organizational model (Moise+) is presented. MASs designed in accord to the Moise+ model are able to re-organize their processes and then react to what occurs in the environment.

Organizations are described in the Moise+ model by three main views: the structural, the functional and normative perspectives. In this model an organization is established a priori (created at design-time) and the agents ought to follow it. The structural and functional view are considered almost independent while the normative dimension is used for establishing a link between them. Furthermore, the Moise+ model is complemented with a development tool called J-Moise+[6], a Jason extension allowing developers to use Jason for programming agents and their organizations [1]. This is nevertheless a powerful tool, but it is not still adequately supported by a well defined methodological approach.

Some researchers have developed in the past other methodologies for MASs where some aspects of organization were modeled. In [18] the concepts of environment, roles, interactions and organizational rules are considered as organizational abstractions. Another example has been proposed in [2] where holarchy represents the organization structure of the MAS made of holons [4] hence the main element to be developed for building the MAS organization. Despite the number of methodologies only few of them cover the entire process lifecycle, from analysis to implementation, and above all very few is aided by tools.

In this paper a preliminary version of MoT (*Moise+ Tool*) is presented. MoT covers the phases from the agent organization design to Moise+ code generation. MoT is based on a UML compliant graphical notation to represent the Moise+ specific elements and on a code generator in order to produce the final XML code containing the Moise+ organizational specification.

MoT has been realized by using a known tool, Metaedit+ by Metacase [14][5], that offers a valid environment for domain specific modelling. Metaedit+ provides means for creating an ad-hoc modelling language with concepts and rules from a well specific problem domain, and notation to be used for drawing diagrams. For the purpose of the presented work we created a graphical notation for representing organization that is based on UML.

The advantages of graphically representing organizations are evident: first of all, graphical notations are more readable and understandable at a glance than any coding language, secondly it is usually easier to explain a graphical notation to stakeholders involved in the design (that are not technical designers) than read the application code with them. The possibility of involving stakeholders like system users enables the adoption of agile or extreme development approaches and improves the flexibility of conventional ones.

The remainder of the paper is organized as follows. In section 2 the Moise+ organizational model and Metaedit+ are introduced. In section 3 we explain the proposed tool with its diagrams and notation by using an example inspired by the Moise+ tutorial [7] example. Moreover, in this section we address the issues concerning the Moise+ code generation. Finally some discussions and conclusions are drawn in section 4 offering also a comparison with others MAS modeling proposals.

2 Background and Motivation

Since the beginning of computer science the need for adequately managing concepts related to the applications under development raised with the complexity of systems. A promising approach to this issue has been the definition of means for specifying what a system should do instead of how to do something. This approach led to formulation of the Model driven Engineering [13] (MDE) paradigm that deeply changed the way of thinking and then working of designers and programmers.

Designers and developers are no more involved in the specification of each single detail of the system using a programming language but they can model the needed functionalities and the architecture of the system. This fact presents many advantages like the increasing goodness of the softwares produced, the easiness and the rapidity of conveying information among team members and the possibility, through the use of model transformation techniques, of automatically generating code. However this latter issue is not still supported by adequately technology.

Our work focuses on the creation of a notation and a CASE tool, created as an instance of a meta-CASE tool (Metaedit+), for supporting the methodological activities involved in the development of organizational MASs. In so doing we exploited the Moise+ organizational model and the features of Metaedit+ for creating a graphical environment allowing the designer to implement concepts and rules of the Moise+ model in specific design diagrams and to automatically produce portions of code.

In the next subsections an overview of Moise+ and Metaedit+ is given.

2.1 Moise+

Moise+ [8][9] is an organizational model for MASs based on a few key elements to characterize an organization. It provides MASs with an explicit definition of their organizations. The organizational specification is useful both to the agents to clearly know their organizational structure and their particular purpose and to the organization framework, to ensure that the agents follow the specifications. More specifically, Moise+ looks at organization as a three dimensional element characterized by structural, functional and normative dimension.

Looking only at the structural dimension, an organization can be seen as a set of *Roles* linked by *Relations* and clustered into *Groups*. The functional dimension enriches the model showing the global objectives of the organization. It gives some information about the plans and the way for reaching the organizational global goals by means of *Social Schemes*. In these schemes the functionalities of the organization are represented as *Goals* grouped into *Missions*.

Finally, the normative dimension is fundamental into the Moise+ model because it shows the connecting elements, the *Norms*, between the functional and structural dimension of an organization. It defines the behavioral rules to be observed by *Roles* in order to reach the organizational global goal. Defining the norms basing on Moise+ means to create links between Roles and Missions. Actually, Moise+ supports two kinds of norms: the *Permission* and the *Obligation* norms.

Practically, designing an organization using the Moise+ model means to define an Organizational Specification (OS) which is the union of the structural, functional and normative specification corresponding to each dimension. An OS is an XML file with a precise structure that defines the features of the previously mentioned elements. In the following a portion of Moise+ XML code representing the skeleton of a classical Organizational Specification is reported. This code shows not only the main elements to be defined inside each specification but also the order in which the elements have to be defined.

```
< organisational − specification >
   < structural − specification >
      < role − definitions > . . .
      < group − specification > . . .
      < formation − constraints > . . .
   < /structural − specification >

   < functional − specification >
      < scheme >
         < goal > . . .
         < mission > . . .
      < /scheme >
   < /functional − specification >

   < normative − specification >
      < normtype =?role =?mission =? > . . .
   < /normative − specification >
< /organisational − specification >
```

Fig. 1. Moise+ XML code representing an organizational specification

In section 3 we present the proposed CASE tool developed in order to easily realize organization with Moise+.

2.2 Metaedit+

Recently designers manifested the need for changing CASE tools in order to customize them for their demands and to meet the features of different application domains. This customization is not possible with every CASE tool because tools constrain how the designer can do their work, how they can draw diagrams/models or manage tool concepts. Generally tools allow to use only fixed methods and notation.

What Metaedit+ proposes is a way for overcoming this limitation by adding the notion of meta-CASE tool to that of CASE tool. The meta-CASE tool is based on a three layers architecture in which the lowest level is the model level, hence the system design. The middle level contains a model of the bottom level, the model of a model is called metamodel. Metamodel contains concepts and rules for creating models. These two levels are already present in a CASE tool but the metamodel is imposed by the creators of the tool thus implying the previous said rigidity.

With the introduction of the third layer (the meta-metamodel one) Metaedit+ establishes concepts and rules for creating metamodels, indeed Metaedit+ offers the possibility of modifying the metamodel by following the rules established in the meta-metamodel, thus overcoming the constraints of CASE tools and having the possibility of specifying modeling languages that can then be used with the right tool. Metaedit+ is at the same time a CASE tool and a meta-CASE tool, by using the meta-CASE tool the designer may specify her/his own modeling language that (s)he can use by instantiating the meta-CASE tool in the CASE tool.

MetaEdit+ is based on a specific metamodeling language, GOPPRR that means Graph, Object, Property, Port, Relationship and Role. They are the metatypes used for defining modeling languages and each of them has its own semantic. Graph is the individual model, usually a diagram, the object is the main element of the graph, the relationships connect objects, the role connects relationships and objects, port gives the possibility to add semantics to the role and the property. The structure and the semantic of each modeling language can be described by a metamodel created by using these metatypes.

In addition to the previous features Metaedit+ offers an optimum support to the UML modeling language on which a lot of design methodologies are based. Finally Metaedit+ offers some preinstalled reports, or the possibility of creating new ones by using a specific language, the Metaedit Reporting Language (MERL). The report is a small program defined and working onto every diagram and, in addition to other facilities it offers, there is the document generation in html format or others and the generation of code skeleton in various programming languages (Java, C, C++, . . .). The more the description of each single element of the diagram is precise and detailed the more the produced code is complete.

This latter functionality has been highly exploited in order to create a report for each single newly introduced diagram of the proposed work and to generate the corresponding xml code.

3 An organization design tool: MoT

The *Moise+ Tool* (MoT) wants to be a tool that covers all the phases from the agent organization design to Moise+ code generation. MoT has been realized by using Metaedit+. It owns a graphical notation to represent the Moise+ specific elements and a code generator in order to produce the final XML code containing the Moise+ organizational specification.

MoT is based on the metamodel shown in Fig. 2, it describes an organizational structure for MASs adapted from Moise+. The core element of the metamodel is the organization that pursues some objectives (Goals), each of them reachable executing a particular scheme. A scheme contains several missions responsible of a set of goals. In addition, an organization is composed of several roles. When an agent adopts a role it is committed to a mission that is regulated by means of norms. The organizational link and the compatibility link respectively define social exchanges and compatibility relations among agent roles.

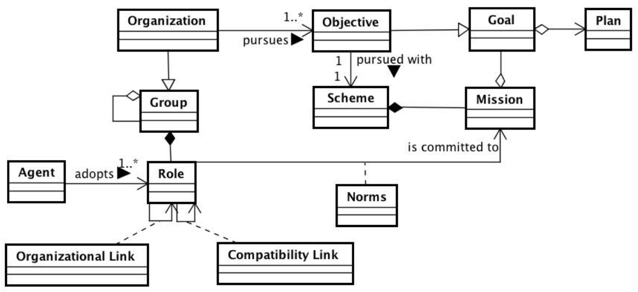


Fig. 2. Metamodel adopted in the MoT

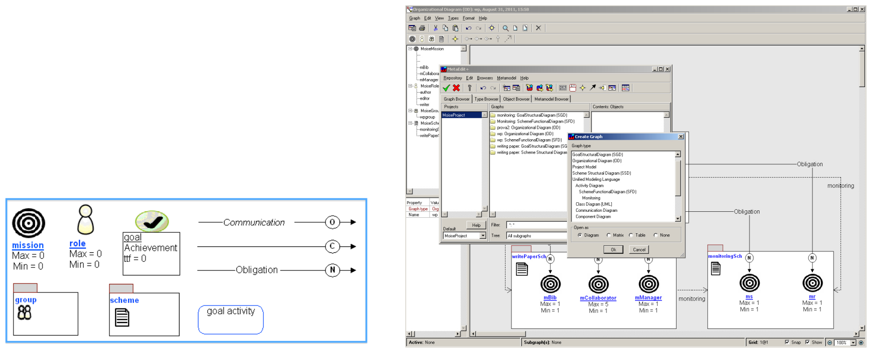
In the following subsections we present the adopted notation, its usage into specific diagrams and the corresponding code generation of the proposed tool.

3.1 Diagrams and Notation

As we have previously said, the adopted notation is UML compliant and, since a detailed description of UML is out of the scope of the paper, in this section we define only the constructs applied to design organizations with Moise+.

Considering the features of Moise+ (see section 2.1) and its metamodel (see Fig. 2 we can infer that in order to model organizational MASs we need four kinds of diagram: the *Organizational Diagram - OD*, the *Scheme Structural Diagram - SSD*, the*Goal Structural Diagram - GSD* and the *Goal Functional Diagram - GFD* that we will detail in the following. These diagrams can be composed using the notation we present in this paper. The notation allows to represent all the concepts involded in modeling and desinging organizational MASs and has been created as a UML profile. In the following subsections each diagram will be detailed with the aid of the the classical example ("Writing paper") reported in the Moise Tutorial [7]. Fig. 3(a) shows the graphical elements of the proposed notation which meanings and definitons will be given later in the paper.

A) Organizational Diagram. The Organizational Diagram is an extended UML class diagram for designing the structural and normative aspect of an organization. The OD focuses on Moise+ elements such as Roles, Groups, Missions, Schemes and different kinds of relationships.



(a) The Defined Notation Elements

(b) Screenshot of MoT

Fig. 3. MoT: Notation and Screenshot

The methodology for developing MAS organizations with Moise+ is out of the scope of this work, but for the sake of clarity we can say that building any Organizational Diagram is articulated in two different phases. During the first phase all the elements concerning the Moise+ structural specification are established in the Scheme Structural Diagram. The second phase aims to define the Norms the agents should obey when they adopt a Role.

In a generic OD the key elements are:

Roles - A Role is a UML class depicted as a sticky man. Its properties are represented in the form of class attributes. In order to add a role in an OD, the *MoiseRole* object must be selected from the toolbar of the OD view (see Fig. 3(b)). The main features of a MoiseRole are: a RoleName, a MaxAscribe and a MinAscribe representing the cardinality of the role in the organization. An abstract role, as usual, is identified using an italic font.

Groups - A Group is represented by means of a package with a sticky men icon. It may contain several structural elements (Roles) and other grouping elements (sub-groups). The root group represents the entire organization. In order to add a group in an OD, the *MoiseGroup* object must be selected from the toolbar of the OD view. According to the Moise+ definition, the membership of an agent to a group constrains the agents that can cooperate with it.

Mission - In the Moise+ model, a Mission is defined as a coherent set of authorized goals to achieve. In order to represent a mission in MoT we have used a UML class graphically depicted as a dartboard. Here the attributes' compartment contains values for the minimum and the maximum commitments to the mission.

Social Scheme - According to Moise+, a Social Scheme or simply Scheme is basically a goal decomposition tree where the root is the objective of the Scheme and where the responsibilities for the sub-goals are distributed into missions. The schemes in an OD are represented only as a set of associated missions while the next diagrams show further features.

In an OD, a Scheme is modeled by means of a package with a little sheet icon, where classes (i.e. missions) are grouped. The package's name corresponds to the social scheme id. There can be more than one Scheme in an OD thus representing the existence of different schemes in the same organization with different objectives.

Relationships - According to Moise+, the elements of the model can be logically related to one another using several kinds of relationships. The relationships allowed in an Organizational Diagram are:

- ▷ ***Organizational Link*** - It defines the way in which social exchanges between agent roles occur. Moise+ model defines three types of Organizational links: *communication* representing exchange of information; *authority* defining control power; *acquaintance* representing knowledge about other agents. In MoT these relations are graphically represented as shown in Fig. 3(a) and can be characterized by means of a label showing the type. In order to add an Organizational link in an OD, firstly the *MoiseOrganizationalLink* relationship from toolbar of the OD view must be selected, secondly the source and the target role among instantiated roles in the diagram must be chosen.
- ▷ ***Compatibility Link*** - It is always plotted between two roles and establishes the possibility for an agent to play the two roles simultaneously. In MoT this relation is graphically represented as shown in Fig. 3(a). When the link is oriented, it means that the agent playing the source role can play the target role but not the vice-versa. In an OD, a Compatibility link is added analogously to the Organizational one.
- ▷ ***Generalization*** - It, as usual, specifies a relationship between roles in which specialized roles inherit features of the general role.
- ▷ ***Norm*** - In the Moise+ model, a role is usually linked by means of Norms to one or more missions defined in a particular scheme. In our tool, we have defined a new link type named *MoiseNormLink* graphically represented as shown in Fig. 3(a). This link is characterized by the NormLinkType propriety and can take two values: *Obligation* and *Permission*. It expresses that an agent playing the role is obliged/allowed to fulfill the mission. In MoT this link is a directed arc that starts always from a Role to a Mission.

Fig. 4 shows the Organizational Diagram for the Writing Paper organization. In this example, a set of agents aims to write a paper. For this purpose, an organization with one group (*wpgroup*) and two roles (*Writer* and *Editor*) is defined. These roles are an extension of the abstract role *Author*. An agent can play several roles only if they are compatible. As exemplified in figure 4, an agent playing the writer role can play the editor role at the same time and vice-versa because they are linked by a bidirectional *MoiseCompatibilityLink*. Moreover, in this diagram are also represented the organizational links existing between roles. For example, the MoiseOrganizationalLink between editor and writer role is of the type Authority. This means an agent playing the role editor in the writing paper organization has some kind of control on agents playing the writer role.

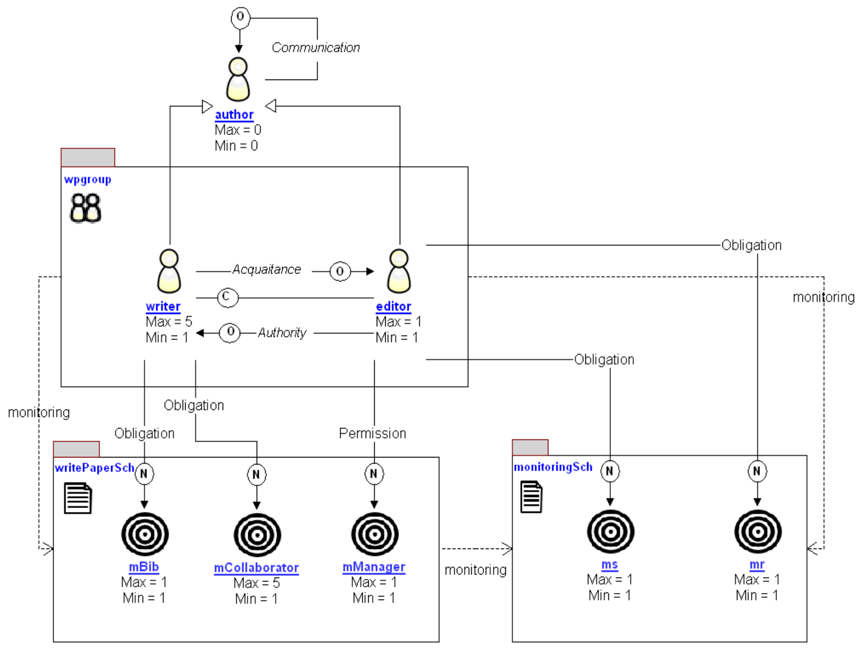


Fig. 4. The Writing Paper Example - OD

Finally, the instantiated roles are linked by means of MoiseNormLinks to related missions. In the portion of diagram reported in Fig. 4, one of the *Writer*'s mission is *mbib* (i.e. getting references for the paper). The *norm* linking that mission to the role is an Obligation, that is the agent playing the Writer role must commit to this mission. The *Editor*, instead, may commit to the mission *mManager* because the link is a Permission norm.

B) The Scheme Structural Diagram. The Scheme Structural Diagram is the second kind of diagram supported by our tool. This diagram allows MAS developers to design more in detail the structure of the Social Schemes in terms of goals and missions. While in an Organizational Diagram we can see the missions belonging to a Scheme, in this diagram we can specify the composition of each single mission with related goals. One of these goals is labelled as the root goal of the Scheme. A SSD is basically a UML class diagram and its main elements are Goal, Scheme and Mission.

Goal - In order to represent a goal in MoT we have used a UML class graphically depicted as circle with a check. Each goal element is characterized by a name and by a collection of attributes. Each attribute corresponds to a specific feature of the Moise+ concept of goal. In order to add a Goal in an SSD the *MoiseGoal* object from toolbar of the SSD must be selected. As regard the attribute compartment, it basically contains the *GoalType* propriety that represents the two kinds of goal namely achievement and maintenance and the *ttf* attribute value prescribing the time requested for fulfilling the goal. The default type for every goal is achievement.

The Mission and the Scheme are the same previously defined and imported in the SSD view.

As regard relationships among elements, in this diagram we only use two kinds of relationship: the aggregation and the dependency. The latter is used for representing how two different schemes depend each others, the former is used for relating missions and goals. With respect to Moise+, goals are aggregated into missions that can be distributed/committed to agents.

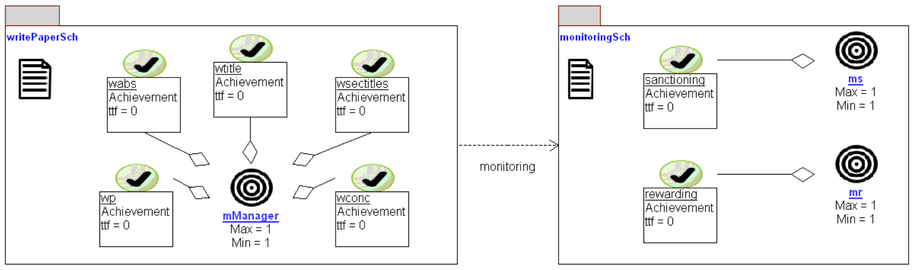


Fig. 5. The Writing Paper Example - SSD

Fig. 5 shows a portion of the SSD for the write paper example. As we previously said, a Social Scheme is modeled by means of a package containing missions and goals. Within a package the structural composition of goals and missions is defined. Thus, the SSD for writing paper example is composed of two Social Schemes, *writePaperSch* and *monitoringSch*. The portion of writePaperSch scheme reported in Fig. 5 shows how the *mManager* mission is a composition of five goals: *wp*, *wtitle*, *concl*, *wabs*, *wsectitles* that respectively aim to write the paper, the title, the conclusion, the abstract and the title of each section. While the illustrated portion of monitoringSch scheme shows *ms* mission formed by only *Sanctioning* goal. In the SSD, it is also possible to underline the dependences between different Social Schemes. As Fig. 5 shows, the Scheme *writePaperSch* is related to the *monitoringSch* Scheme through a "monitoring" dependency relationship.

In the next subsections, we represent the root goal of the Scheme from two points of view (structural and functional) adopting two related diagrams: the Goal Structural Diagram and the Goal Functional Diagram.

C) The Goal Structural Diagram. The Goal Structural Diagram provides a goal analysis by dividing goals into subgoals through and AND or OR decomposition. This allows to find the minimal set of subgoals for satisfying the

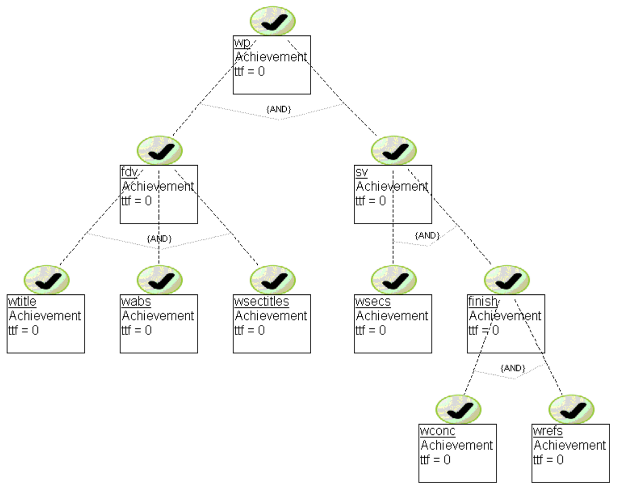


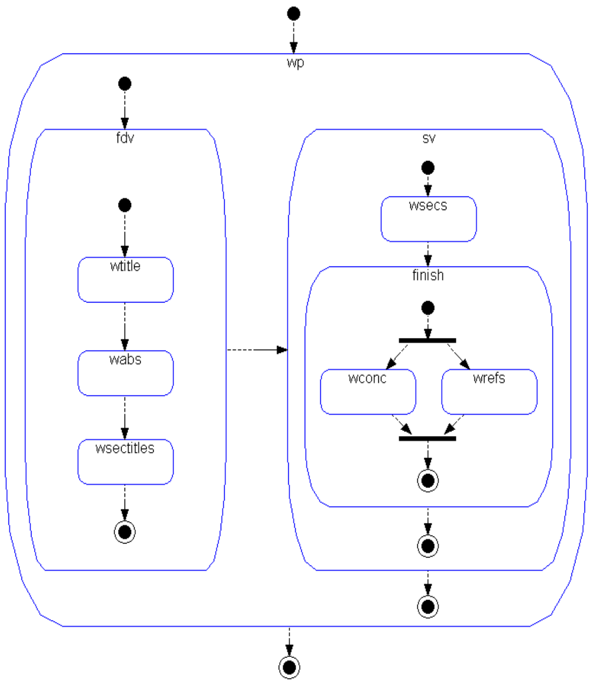
Fig. 6. Goal Structural Diagram of the WritingPaperSch root goal - GSD

global goal and to determine conflict among goals. Practically a GSD provides the possibility to do some kind of reasoning in order to make the root goal of the associated scheme reachable.

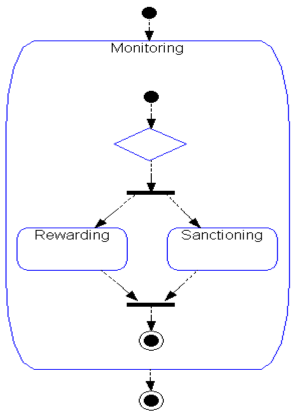
In MoT, the GSD is an extended UML class diagram where the goal is the only Moise+ element permitted. In this diagram goal are related to other goals by means of an AND or OR dependency relation. As we can see in the Fig. 6, in the *WritingPaperSch* scheme, the root goal of the writing paper organization may be reachable only if all its subgoals have been satisfied because the relations linking goals with subgoals are AND relations.

D) The Goal Functional Diagram. The Goal Functional Diagram represents the behavioral view of the organization, how the task/activity related to each subgoal must be executed in order to fulfill the scheme root goal. It is important to highlight there are three different types of goal execution: sequential, parallel and choice. If two goals are related with a sequential relationship then the target goal can be reached only after that the source goal is reached. If two goals are related with a parallel relationship then both goals can be reached simultaneously. Finally, a choice relationship indicates that it is possible to choose the goal to be achieved. A GFD is realized by means of a UML activity diagram. The elements of this diagrams are:

The **Goal** is here represented by an activity where the name is the goal's id. According to Moise+ model, in a GFD it is possible to decompose goal in sub-goals by means of a plan operator. There are three different kinds of plan operator: *sequence*, *parallelism* and *choice*, the first means that a goal g_i (having two sub-goals $g_{i,i}$ and $g_{i,i+j}$) can be achieved only if the sequence of $g_{i,i}$ and $g_{i,i+j}$ is terminated. All of them can be easily represented by means of the UML



(a)



(b)

Fig. 7. Goal Functional Diagrams of the Writing Paper Example - GFD

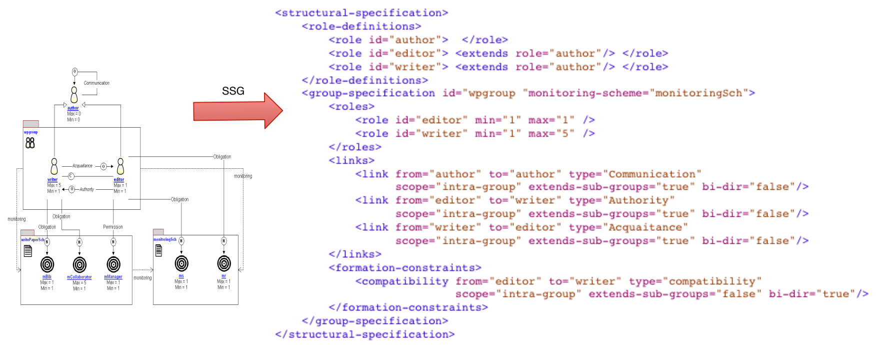
activity diagram syntax, for instance the parallelism is represented through the fork and the choice through the decision diamond. Sequence is represented by a straight arrow line.

Fig. 7(a) and Fig. 7(b) show the Goal Functional Diagrams built for the *wp* and *monitoring* goals of the Writing Paper organization which are the root goals of *writePaperSch* and *monitoringSch* (defined in the previous section) correspondingly. The GFD of the *writePaperSch* (see Fig. 7(a)) explains how to achieve the root goal of the scheme. In detail, the fulfillment of the *wp* goal (i.e. write a paper) depends on the achievement of the *fdv* (first draft version) and *sv* (submit version) goal. The *sv* goal is reachable only after that the *fdv* is satisfied. In turn, *fdv* is achieved executing the atomic goals *wtitle*, *wabs* and *wsectitles* sequentially.

In subsection 3.2 the code generation is addressed.

3.2 Code generation

In the previous sections we have defined the domain-specific modeling language in order to design agent organizations to be implemented in Moise+. The resulting metamodel containing the domain concepts with their relations and notation



```
<structural-specification>
    <role-definitions>
        <role id="author">  </role>
        <role id="editor"> <extends role="author"/> </role>
        <role id="writer"> <extends role="author"/> </role>
    </role-definitions>
    <group-specification id="wpgroup "monitoring-scheme="monitoringSch">
        <roles>
            <role id="editor" min="1" max="1" />
            <role id="writer" min="1" max="5" />
        </roles>
        <links>
            <link from="author" to="author" type="Communication"
                  scope="intra-group" extends-sub-groups="true" bi-dir="false"/>
            <link from="editor" to="writer" type="Authority"
                  scope="intra-group" extends-sub-groups="true" bi-dir="false"/>
            <link from="writer" to="editor" type="Acquaitance"
                  scope="intra-group" extends-sub-groups="true" bi-dir="false"/>
        </links>
        <formation-constraints>
            <compatibility from="editor" to="writer" type="compatibility"
                           scope="intra-group" extends-sub-groups="false" bi-dir="true"/>
        </formation-constraints>
    </group-specification>
</structural-specification>
```

Fig. 8. Moise+ Structural Specification generated code from OD

is shown in Fig.2. In this section we specify the mapping from model to code by defining a domain-specific code generator.

In MetaEdit+, code generators are defined in the Generator Editor using the MERL scripting language. MERL enables navigating trhough the elements of the user designed diagrams accessing the data according to the defined metamodel. Moreover, MERL allows translating the design data into the formats required by the generation target language.

For our purposes we have defined the main generators associated with the diagram types defined in the section 3.1. Each generator is responsible of producing a Moise+ specification portion.

Specifically, the *Structural Specification Generator* (SSG) and the *Normative Specification Generator* (NSG) analyzing the elements designed in the Organizational Diagram produces the XML portion code concerning the Moise+ Structural and Normative Specification respectively.

The Functional Specification Generator (FSG) generates the Moise+ Functional Specification. This (as hinted in the section 2) shows how the organizational goals can be reached and how to compose the missions to be assigned to a specific role. For these reasons, the FSG is obtained merging two sub-generators: the former maps the elements described in the Goal Functional Diagram in the XML code concerning the Moise+ goal decomposition tree; the latter traduces the design data of the Scheme Structural Diagram in the portion of XML code representing the composition of the missions.

The Fig. 8 shows the portion of structural specification generated by means of application of SSG to the OD of the writing paper example.

4 Conclusion

In order to fully exploit the powerful of agents nowadays research is directing towards multi agent systems organized in the same way the humans do. The design and implementation of this kind of system obviously requires to manage abstraction that can be used for modeling norms, goals, social schemes ad so on. Above all it requires supporting tools for guiding the designer from the analysis to the implementation in simple and less costly fashion.

The work presented in this paper is a step towards the creation of a design process for developing MASs organized in hierarchical structures that can be implemented with Moise+ and supported by a CASE tool using a specific notation for representing organizations.

We developed a CASE tool by using Metaedit+ that allows to generate specific code for each kind of diagrams, in so doing we are able to support the designer in producing organizational multi agent systems models and then implementing them in a semi automatic way. The Moise+ metamodel is at the base of our tool that thanks to automatic generation of code from diagrams lets the designer free from the heavy work related to the manual production of organization XML code.

Finally it is worth noting that the use of Metaedit+ constitutes a first experiment that produced a very good results in terms of CASE tool for supporting design but the approach we adopted for the creation of the UML profile for representing organizations is general enough for being applied with every kind of tools since it is grounded on the creation of a metamodel that complement the one of Moise+ with that of UML. For the future we are planning to develop a CASE tool as an extension of Eclipse that might let us overcome the age-old limit of Metaedit+ in managing images and easily positioning elements in the diagrams.

Acknowledgment. This work was realized within IMPULSO and partially supported by the EU project FP7-Humanobs and by the FRASI project.

References

1. Raphael H. Bordini, Jomi Fred Hübner, and Michael J. Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason.* Wiley-Interscience, 2007.
2. M. Cossentino, N. Gaud, V. Hilaire, S. Galland, and A. Koukam. ASPECS: an agent-oriented software process for engineering complex systems. *Autonomous Agents and Multi-Agent Systems*, 20(2):260–304, 2010.
3. V. Dignum and F. Dignum. Modelling agent societies: co-ordination frameworks and institutions. *Progress in Artificial Intelligence*, pages 7–21, 2001.
4. K. Fischer, M. Schillo, and J. Siekmann. Holonic multiagent systems: A foundation for the organisation of multiagent systems. *Holonic and Multi-Agent Systems for Manufacturing*, pages 1083–1084, 2004.
5. Isazadeh H. and Lamb D. A. Case environments and metacase tools. 1997.
6. Jomi Fred Hübner. J-moise+ programming organizational agents with moise+ and jason (2007).
7. Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. Moise tutorial. (for moise 0.7).

8. Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. Moise+: towards a structural, functional, and deontic model for mas organization. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*, page 502. ACM, 2002.
9. Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. Developing organised multiagent systems using the MOISE+ model: programming issues at the system and agent levels. *International Journal of Agent-Oriented Software Engineering*, 1(3):370–395, 2007.
10. INGENIAS. Home page. `http://grasia.fdi.ucm.es/ingenias/metamodel/`.
11. L. Padgham, M. Winikoff, S. DeLoach, and M. Cossentino. A unified graphical notation for aose. *Agent-Oriented Software Engineering IX*, pages 116–130, 2009.
12. Juan Pavòn, Jorge J. Gòmez-Sanz, and Rubén Fuentes. The INGENIAS methodology and tools. In *Agent Oriented Methodologies*, chapter IX, pages 236–276. Idea Group Publishing, 2005.
13. Douglas C. Schmidt. Model-driven engineering. *Computer*, 39(2):25–31, Feb. 2006.
14. Juha-Pekka Tolvanen and Matti Rossi. Metaedit+: defining and using domain-specific modeling languages and code generators. In *OOPSLA '03: Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 92–93, New York, NY, USA, 2003. ACM Press.
15. H. Van Dyke Parunak and J. Odell. Representing social structures in uml. *Agent-Oriented Software Engineering II*, pages 1–16, 2002.
16. M. Wooldridge and N.R. Jennings. Intelligent Agents: Theory and Practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.
17. Michael J. Wooldridge. *Introduction to Multiagent Systems*. John Wiley & Sons, Inc. New York, NY, USA, 2001.
18. F. Zambonelli, N. R. Jennings, and M. Wooldridge. Developing multiagent systems: The Gaia methodology. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 12(3):317–370, July 2003.