Towards a Design Process for Modeling MAS Organizations

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, 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 methodological approach may offer a significant advantage in enabling the conception of the best solution. Moreover, 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. In this paper, we introduce a portion of methodological approach devoted to design MAS organizations and a preliminary version of a specific case tool, named MoT (*Moise+ Tool*), for supporting activities from design production to automatic code generation. MoT provides four kinds of diagrams based on a definite graphical notation for representing organizational elements. Our process 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 behavior for reacting to the new conditions. Agents can provide a good way for solving complex problems and they are very useful to both design and implementation levels [13][14].

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 [4] 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. [10] where a definition of an organizational

model (Moise+) is presented. MASs designed in accord with 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+[7], 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 [16] the concepts of environment, roles, interactions and organizational rules are considered as organizational abstractions. Another example has been proposed in [3] where holarchy represents the organization structure of the MAS made of holons [5], 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 we introduce a portion of methodological approach devoted to design MAS organizations and a preliminary version of a specific case tool, named MoT (*Moise+ Tool*), for supporting our approach from design production to automatic code generation.

In particular, 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 [12][6], that offers a valid environment for domain specific modeling. Metaedit+ provides means for creating an ad-hoc modeling language with concepts and rules from a well specific problem domain, and notation to be used for drawing diagrams.

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 present our tool with the definite notation. Section 4 shows a portion of the design process for developing organizational MAS with the related work products. Such process is explained applying it to an example inspired by the Moise+ tutorial [8]. Moreover, in this section we address the issues concerning the Moise+ code gen-

eration. Finally some discussions and conclusions are drawn in section 5 together with 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 [11] (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+ [9][10] 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 layered 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[3]) wants to be a tool supporting 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. Fig. 2 shows a screenshot of MoT.

[3] MoT is available at http://www.pa.icar.cnr.it/aose/MoT.html

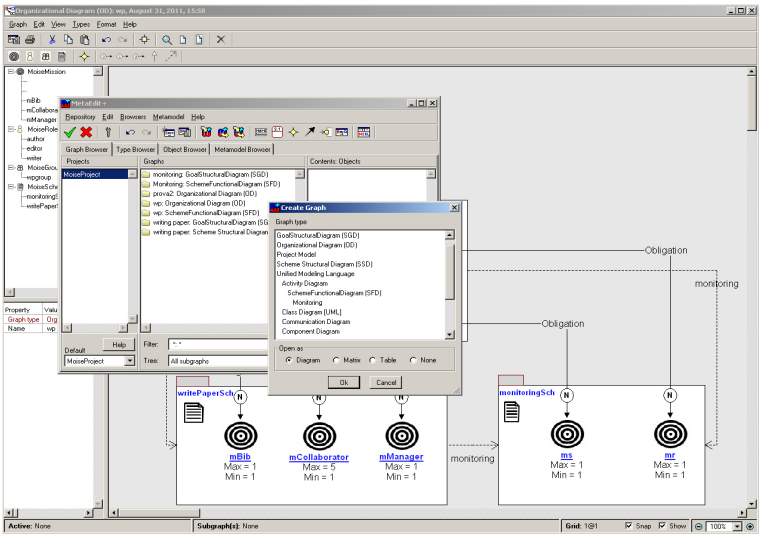

Fig. 2. A Screenshot of MoT

MoT is based on the metamodel shown in Fig. 3, 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 Plan. A Group is usually responsible of at least a Scheme and a Scheme can be adopted to monitor the execution of another Scheme. A Scheme contains several Missions composed 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.

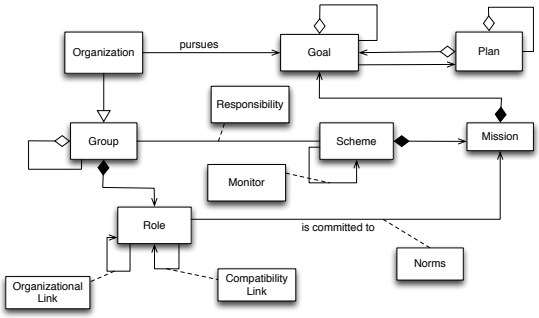

Fig. 3. Metamodel adopted in the MoT

In the following subsections we present the adopted notation.

Fig. 4. The Notation for MoT

3.1 Notation

MoT provides four kinds of diagrams: 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. Such notation allows to represent all the concepts involved in modeling and designing organizational MASs according to the metamodel shown in Fig. 3. This notation has been created as a UML profile. It allows to represent the following concepts (graphically shown in Fig. 4).

Roles - A Role is a UML class depicted as a sticky man. Its properties are represented in the form of class attributes. 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). According to the Moise+ definition, the membership of an agent to a group constrains the agents that can cooperate with it.

Goals - 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. 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.

Missions - 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 Schemes - A Social Scheme or simply Scheme in Moise+ is composed of a functional goal decomposition tree (where the root is the objective of the Scheme and the goals are decomposed within global plan) and a set of missions (where the responsibilities for the sub-goals are grouped to be distributed among Roles). In our tool, we prefer split its structural aspect (described in the SSD and represented by means of a UML package) from the functional one (described as a plan in the GFD). Thus, the schemes in the SSD are represented only as a set of associated missions while the GFD shows further features.

Relationships - The elements of the model can be logically related one another using several kinds of relationships. We omit to define the common UML relations that we use in MoT diagrams. In the following we describe those specially introduced for specifying Moise+ concepts.

- ***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 by the first relation shown in Fig. 4 and can be characterized by means of a label showing the type.
- ***Compatibility Link*** - It is always plotted between two roles and establishes the possibility for an agent to simultaneously play the two roles. It is graphically represented by the second relation shown in Fig. 4. When the link is oriented, it means that the agent playing the source role can play the target role but not the vice-versa.
- ***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 by the third relation shown in Fig. 4. This link is characterized by the NormLinkType propriety and can take two values: *Obligation* and *Permission*. In MoT this link is a directed arc that starts always from a Role to a Mission. It expresses that an agent playing the role is obliged/allowed to fulfill the mission.

In the following section we introduce all phases of our methodological approach with the related work products (MoT diagrams). Our design process will be detailed with the aid of the the classical example ("Writing paper") reported in the Moise Tutorial [8].

4 Methodological Approach

We aim at defining a complete methodological approach ranging from requirements analysis to code production and system deployment. Such a methodology will include a goal oriented analysis (with some features inspired by the i* [15] and Tropos [2] approaches), the design of organizations that will be described below and the design of agents based on the Jason platform. The scope of this paper is limited to the organizational part of this work and therefore (also for

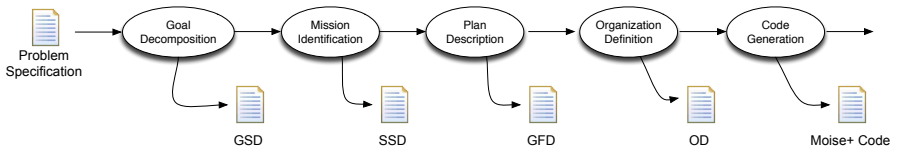

Fig. 5. Portion of the Design Process for Organizational Multi-Agent Systems

space concerns) we skip the initial part of the methodology (requirements analysis) and the final one (the agent design and what follows it). In other words, this section introduces only the portion of our methodological approach devoted to instantiate the metamodel shown in Fig. 3. The diagrams we illustrated in the previous section are used for representing the outcome of this portion of design process, as it is sketched in Fig. 5. Let us assume that the problem specification document is already existing and it provides a list of system goals obtained for instance with a Tropos or i* like design process. The aim of our methodology is to model organizational multi-agent systems principally by means of goals, their decomposition, missions and roles; in the following table, we highlight the work product where each metamodel element of Fig. 3 is instantiated.

Table 1. Summary of instantiated element.

Work Product	**Metamodel Element**
Goal Structural Diagram	Goal
Scheme Structural Diagram	Scheme, Mission
Goal Functional Diagram	Plan
Organizational Diagram	Role, Group, Monitor, Norm, Organizational Link, Compatibility Link

In the following subsections are detailed all phases of our approach shown in Fig. 5.

4.1 The Goal Decomposition Phase

The *Goal Decomposition* phase (see Fig.5) of the proposed design process involves activities for the decomposition of the identified goals and the identification of their dependencies. During this phase, goals are refined by means of an AND/OR decomposition. This allows to determine a hierarchical structure among goals and to individuate the dependencies between a high level goal and its subgoals. A dependency among goals implies that a given goal is constrained

by another one for its fulfillment. In particular, an AND dependency means that all subgoals must be satisfied in order to fulfill the original goal. Vice versa, in an OR dependency the original goal is satisfied when any one of its children is fulfilled. This phase results in the *Goal Structural Diagram* where the *goal* (see metamodel Fig. 3) is instantiated.

In MoT, the GSD is an extended UML class diagram where the Goal is the only Moise+ element permitted.

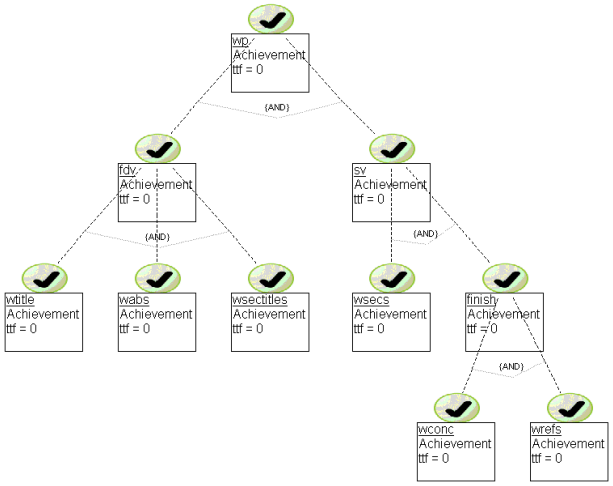

Fig. 6. Goal Structural Diagram for the Writing Paper example - GSD

In order to add a Goal in an GSD the *MoiseGoal* object from toolbar of the GSD must be selected. In this diagram, goals are related to other goals by means of an AND or OR dependency relation.

Fig. 6 shows the GSD for the "Writing paper" example. In this example, a set of agents wants to write a paper. In order to solve the problem, an organizational strategy is adopted. In this instance, we don't want to argue about the design choice. Vice versa, we accept the solution proposed in [8] because we want to show how it is represented in a GSD.

As we can see in Fig. 6, the global objective of the organization (to be created) is decomposed into two sub-goals *fdv* (first draft version) and *sv* (submit version). The *fdv* goal is, in turn, decomposed into three sub-goals: write a title (*wtitle*), an abstract (*wabs*), and the section titles (*wsectitles*). For the other hand, in order to fulfill the *sv*, it is necessary to write the sections (*wsecs*) and to finalize the paper (*finish*), that is to write the conclusion (*wconc*) and the bibliography (*wrefs*). The GSD for the "Writing paper" example highlights the root goal of the organization may be reachable only if all its subgoals have been satisfied.

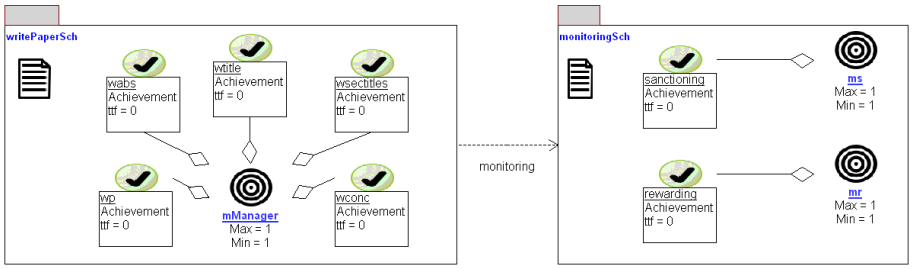

Fig. 7. The Writing Paper Example - SSD

This is because all the relations linking goals with the related subgoals are AND relations.

4.2 Mission Identification Phase

The *Goal Structural Diagram* is the input of the *Mission Identification* phase where the main aim is to identify *Roles*, *Missions* and *Schemes*. This phase starts with the Roles Identification activity. Roles are identified by looking at *Positions* coming out from the previously report Tropos or i*-like analysis phase. We consider a *Position* characterized by its own competencies in order to fulfill its goals. Often Roles are identified in an iterative refinement process working in this way: some candidate Roles are identified, their consistency is verified against the Missions that is possible to assign them (see description of next activity). Roles are splitted or merged according to what emerges from the analysis of Mission assignments. Instantiating Missions is useful for the definition of organizations complying with the Moise specification. Thus it is necessary to establish how to group goals coming from the previous Goal Decomposition Phase. Practically, we group the leaf goals of the diagram into missions according to the previously identified candidate Roles, starting from the GSD. The Role involved in pursuing a Goal is sometimes the same Role who has a direct interest in its achievement, other times the goal is under the responsibility of other Roles. We assume that information about Roles responsibility are coming from the requirements analysis phase and guide this activity. The analysis of mission assignment to Roles may be useful for identifying Roles needing too many capacities or incoherent profiles. This may lead to split the candidate Role. Other times, missions analysis may indicate that similar Roles exist and their merging may be advisable. At the end of this iterative activity, missions are grouped into Schemes according to the high level goal to be satisfied. This series of iterations produces the *Scheme Structural Diagram*, where the *Mission* and *Scheme* metamodel elements (see Fig. 3) are instantiated.

In MoT, the SSD is an extended UML class diagram where main elements are Goal, Scheme and Mission. In the SSD, 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. In a SSD can be represented more than one Scheme, thus representing the existence of different schemes in the same organization with different objectives.

The Goal element is the same previously defined in the GSD and imported in the SSD view. An SSD allows MAS developers to design the structure of the Social Schemes in terms of goals and missions. In this diagram, we can also specify the composition of each single mission with related goals. Some of these goals are labelled as the root goal of the related Scheme.

As regards relationships among elements, 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 will be distributed/committed to Roles.

Fig. 7 shows a portion of the SSD for the "Writing paper" example. It is composed of two Social Schemes, *writePaperSch* and *monitoringSch*. The portion of *writePaperSch* scheme reported in Fig. 7 shows how the *mManager* mission is a composition of five goals: *wp*, *wtitle*, *concl*, *wabs*, *wsectitles*. This mission concerns the general management of the writing process. While the illustrated portion of *monitoringSch* scheme includes two missions: *ms* and *mr* mission formed by *sanctioning* and *rewarding* goal respectively. These missions concern the employment of sanctioning and rewarding policies in order to enforce rules. In the SSD, it is also possible to underline the dependences among different Social Schemes. As Fig. 7 shows, the Scheme *writePaperSch* is related to the *monitoringSch* Scheme through a "monitoring" dependency relationship. This is because the scheme *monitoringSch* is adopted in order to ensure the correct execution of the *writePaperSch*.

4.3 The Plan Description Phase

The *Plan Description* phase allows to establish the precedence relations among goals, that is the temporal sequence in which the goals are to be fulfilled. Establishing precedence relations among goals allows to consider different design choices. This phase is assisted by the *Goal Functional Diagram*. At this stage of the process, the functional aspect of goals (*Plan*) is determined.

The Goal Functional Diagram represents the functional view of the root goals of the schemes. In other words, it depicts how the task/activity related to each subgoal must be executed in order to fulfill the scheme root goal (that is the plans to reach the root goal). It is important to highlight that there are three different types of goal fulfillment: *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 simultaneously reached. Finally, a choice relationship indicates that it is possible to choose the goal to be achieved.

A GFD in MoT is realized by means of a UML activity diagram where the Goal is represented by an activity where the name is the goal's id. In a GFD the plans to reach the goals are also defined. 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 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.

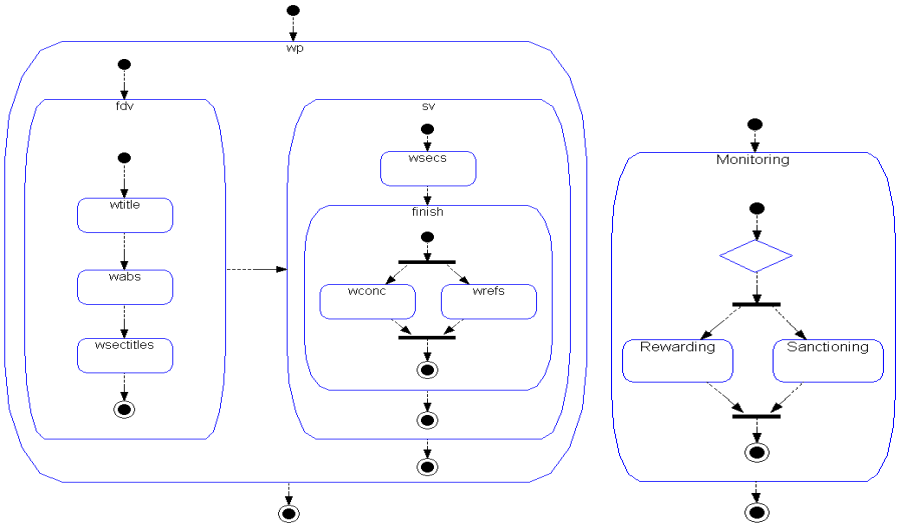

Fig. 8. Goal Functional Diagrams of the Writing Paper Example - GFD

Fig. 8(a) and Fig. 8(b) show the Goal Functional Diagrams built for the *wp* and *monitoring* goals for the Writing Paper example which are the root goals of *writePaperSch* and *monitoringSch* (defined in the previous section) correspondingly. The GFD of the *writePaperSch* (see Fig. 8(a)) explains how to achieve the root goal of the scheme. In detail, the fulfillment of the *wp* goal depends on the achievement of the *fdv* and *sv* 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.

4.4 The Organizational Definition Phase

Finally, the *Organizational Definition* is the core phase of our methodology and it includes several steps. This phase uses the work products coming from the

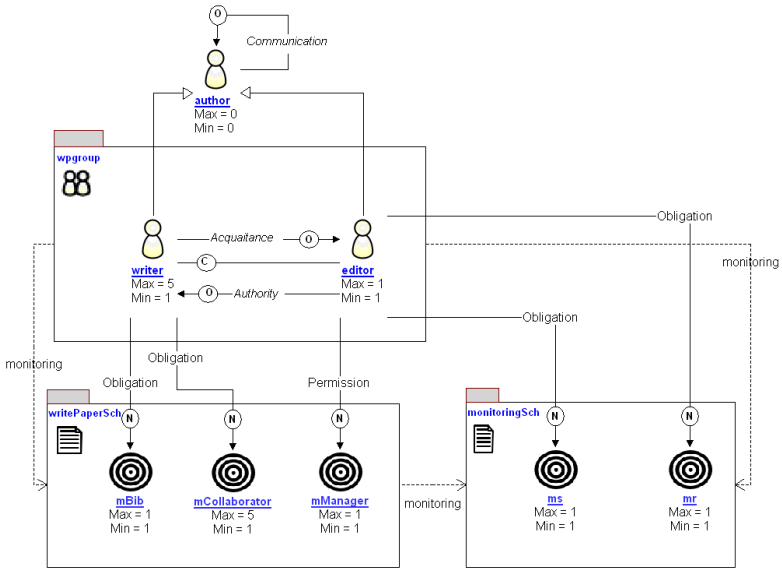

Fig. 9. The Writing Paper Example - OD

previous ones. Thus, it is almost natural finalizing the Roles of the Organization to be created, by means of the elements previously identified and instantiated. As a consequence, we can determine the kind of rule (*Norms*) that bind the Role to the mission and also the work teams (*Groups*) to Roles belong.

The set of all formed work groups compose the entire *Organization*. The last steps of this phase are to establish which Roles are compatibles (*Compatibility Links*) with some others according to the policy adopted in the organization and which subordination relations exist among Roles (*Organizational links*). This phase is assisted by the *Organizational Diagram*.

In MoT, an 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.

Fig. 9 shows the Organizational Diagram for the Writing Paper example. A possible solution for this problem, can be provided defining an organization with one group (*wpgroup*) and two roles (*Writer* and *Editor*). These roles are an extension of the abstract role *Author*. As exemplified in Fig. 9, 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 the organizational links existing between roles are also represented. For example, the MoiseOrganizationalLink between editor and writer role is of the type Authority. This means that an agent playing the role editor in the writing paper organization has some kind of control on agents playing the writer role.

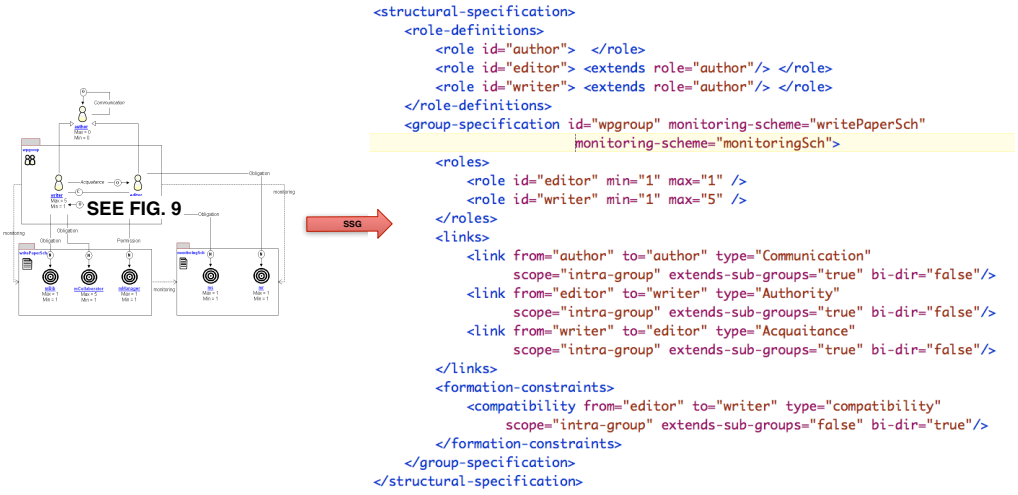

Fig. 10. Moise+ Structural Specification generated code from OD

Finally, the instantiated roles are linked by means of MoiseNormLinks to related missions. In the portion of diagram reported in Fig. 9, 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.

4.5 Code Generation Phase

The last phase of our process is the Code Generation. This phase is devoted to produce the Moise+ organizational specifications of the MAS to be developed. MoT supports this phase generating Moise+ code automatically.

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 is shown in Fig.3. In this subsection, instead, we want to specify the mapping from model to code by defining a domain-specific code generator using MetaEdit+. In MetaEdit+, code generators are defined in the Generator Editor using the MERL scripting language. MERL enables navigating through 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) produces the XML portion of code concerning the Moise+ Structural and Normative Specification by respectively analyzing the elements designed in the Organizational Diagram.

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 of 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. 10 shows the portion of structural specification generated by means of application of SSG to the OD of the writing paper example.

5 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 abstractions that have to 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.

This paper introduce a first 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. In particular, we illustrated a portion of our methodological approach devoted to instantiate the main elements necessary to create an agent organization with Moise+.

Moreover, 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 the basis for 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 very good results in terms of CASE tool for supporting design activities. 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. Paolo Bresciani, Paolo Giorgini, Fausto Giunchiglia, John Mylopoulos, and Anna Perini. Tropos: An agent-oriented software development methodology. *Autonomous Agent and Multi-Agent Systems (8)*, 3:203–236, 2004.
3. 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.
4. V. Dignum and F. Dignum. Modelling agent societies: co-ordination frameworks and institutions. *Progress in Artificial Intelligence*, pages 7–21, 2001.
5. 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.
6. Isazadeh H. and Lamb D. A. Case environments and metacase tools. 1997.
7. Jomi Fred Hübner. J-moise+ programming organizational agents with moise+ and jason (2007).
8. Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. Moise tutorial. (for moise 0.7).
9. 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.
10. 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.
11. Douglas C. Schmidt. Model-driven engineering. *Computer*, 39(2):25–31, Feb. 2006.
12. 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.
13. M. Wooldridge and N.R. Jennings. Intelligent Agents: Theory and Practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.
14. M. Woolridge and M.J. Wooldridge. *Introduction to Multiagent Systems*. John Wiley & Sons, Inc. New York, NY, USA, 2001.
15. E. Yu. Modeling organizations for information systems requirements engineering. *Requirements Engineering*, Proceedings of IEEE International Symposium on (1993):34–41, 1993.
16. Franco Zambonelli, Nicholas R. Jennings, and Michael Wooldridge. Developing multiagent systems: The Gaia methodology. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 12(3):317–370, July 2003.