GIMT: A tool for ontology and goal modeling in BDI Multi-Agent Design

Massimo Cossentino*, Daniele Dalle Nogare ‡, Raffaele Giancarlo‡, Carmelo Lodato*,
Salvatore Lopes* Patrizia Ribino*, Luca Sabatucci* and Valeria Seidita†*
*Istituto di Calcolo e Reti ad Alte Prestazioni - Consiglio Nazionale delle Ricerche
Email: {cossentino, c.lodato, lopes, ribino, sabatucci}@pa.icar.cnr.it
†Dipartimento di Ingegneria Chimica, Gestionale, Informatica, Meccanica
Email: valeria.seidita@unipa.it
‡Dipartimento di Matematica e Informatica
Email: raffaele.giancarlo@unipa.it, daniele.dallen@gmail.com

***Abstract*—Designing and developing BDI multi-agent systems would be facilitated by rising up the level of abstraction to use and by a methodological approach for managing it. To this aim it is common the integration of goal oriented analysis techniques with the design and implementation phases.**

In this fashion, our experience is that the use of an ontology in the early stages of the process is a great support for subsequent phases: goal modeling, agent design and implementation. However, we are aware that building and maintaining an ontology has to be supported by appropriate tools.

This paper proposes GIMT (Goal Identification and Modeling Tool) as a further step towards the creation of a complete methodological approach for developing multi-agent systems to be implemented in the JACAMO framework. GIMT is a CASE tool for supporting ontology building and goal modeling.

Besides the advantages offered by an automatic tool, the other novelty of this research is in the mapping between *metamodeling* based on Model Driven Engineering (MDE) and Domain Specific Modeling Languages (DSMLs) with the Eclipse plug-in development environment.

I. Introduction

The integration of a high-level programming paradigm, such as the BDI multi-agent systems, into a systematic approach, requires a revision of traditional instruments and tools for conducting the requirement analysis, in order to better fit with the well-known concept of goal. Much research exists in the literature to deal with the goal oriented requirement engineering. We aim at developing a complete software methodology that includes a goal-oriented analysis, the design of agent organizations, and the support to the implementation phase.

The fundamental common theme in all these activities is the presence of an ontology. It is necessary for the definition of the problem and of the solution domain, but also for the structure of messages and for organizing agent knowledge in beliefs.

Ontologies are essential for our approach. However, we have faced the problem that building and maintaining them is not trivial. A totally manual approach would add a significant burden to developer that may impact the personal productivity in the project.

The objective of our work is to provide a CASE tool for supporting designers in the goal identification activity. We exploit the results presented in [17] where two activities of the preliminary phase of a complete methodological approach were developed.

Model Driven Engineering (MDE) is an approach promoting the use of models as first-class citizens of a software development process [19]. These models are, in many usual cases, defined using general-purpose modeling languages like the UML: the standard modeling language in the field of object oriented software engineering. Conversely, for restricted domains, it is widely spread the use of so-called Domain Specific Modeling Languages (DSMLs) [14]. They are specifically designed to meet the needs of the target application domain by using specific concepts of the domain. A common practice of MDE is that DSMLs are defined using a metamodel expressed with a general-purpose metamodeling language like MOF [15], to name the most widely used one. These metamodels describe elements of the language the designer can instantiate at the immediate meta-level below in order to build its own DSML.

The results presented in this paper are founded on the use of MDE as a key element for the development of the proposed tool. The main contribution is the development of a CASE tool based on a DSML we have defined in order to support the activities of the requirement analyses proposed in our previous work [17].

It is well known that the joint use of DSMLs and CASE tools allows to guide and automate the model construction and transformation, resulting in an increase of both software development productivity and quality. The tool imposes domain-specific constraints and performs model checking for detecting and preventing many errors in the early phases of the process life cycle.

The tool we propose has been developed by using Graphiti [5], an Eclipse-based graphics framework that enables rapid development of state-of-the-art diagram editors for domain models. Graphiti can use EMF [3] based domain models very easily but can deal with any Java-based objects on the domain side as well.

The rest of the paper is organized as follows, in the next section we present the motivation of our work, in section III

we illustrate the methodological approach for which the tool has been developed, in section IV we present the tool, its functionality and some technological issues on its construction and finally in Section V we draw some conclusions.

II. Motivation and Objectives

Much research has highlighted advantages and strengths of MDE and DSMLs [8][19][7][14] in managing the complexity of domain concepts and domain abstractions while designing complex systems. These are fundamentals in our research. We have been working for a couple of years on the construction of a complete design process for modeling complex MAS systems, including organizations, hierarchical structures and norms [18][13][11][12].

In order to develop such kinds of systems, we need to manage abstractions such as organizations, goals, communications, messages, beliefs and so on. Thus we need a domain-specific modeling language that includes these specific terms as keywords. In addition, the methodological approach we are developing may be facilitated by the creation of CASE tools for supporting designers in each part of their work. However, customization of existing tools for different application domains is not always easy or even possible.

Nowadays, several environments exist for DSMLs, some of them are commercial such as Metaedit+ [21] and Actifsource [1], some are open source and some others are developed as plug-ins for popular IDEs such as Eclipse Modeling Project [2] and Microsoft's DSL Tools [6]. In particular, Eclipse is an open and flexible framework, providing the API for adding functionalities perfectly integrated in the whole working environment. Indeed, the Eclipse framework does not support any specific task but the composition of a set of plug-ins gives Eclipse a particular "configuration" for specific needs. The Eclipse community is committed at providing plug-ins for generating graphical modeling tool. Recently, the Graphical Modeling Project reached good results by using two interesting technologies: Graphiti [5] and Graphical Modeling Framework (GMF) [4]. The distinctive feature in the Graphiti technology is the employment of the Ecore metamodel for representing all the elements and relationships at the base of the graphical view[1].

Ecore metamodel is the modeling language used for creating models in the diagram editor. In other words, the use of a metamodel corresponds to the definition of a DSML. Since in our work [20] we use metamodels for representing the abstractions to be managed during design processes and to be reported in the models, we have a grounded theory for representing the domain abstractions for each kind of application and upon which to construct a DSML.

One of the objectives of our research is (i) to support designers in the early phase of goal oriented requirements analysis for BDI MASs and (ii) to provide a CASE tool in order to aid the design of the goal model.

This line of research exploits the results of [17]. This is part of a broader activity towards the creation of a complete methodological approach for developing multi-agent systems

[1]We discuss our point of view on developing CASE tool with Graphiti and Ecore in section IV, for more details see [5].

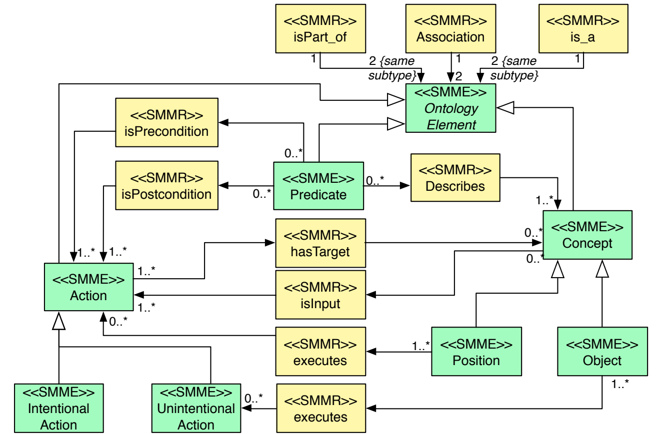

Fig. 1. The Problem Domain Description Metamodel.

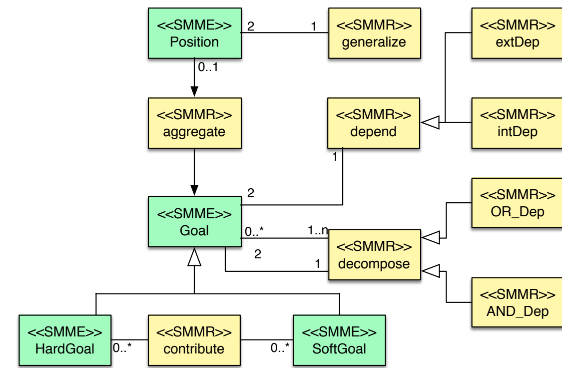

Fig. 2. The Goal Description Metamodel.

to be implemented in the JACAMO framework [9]. The result of [17] is a couple of design activities for identifying goals from the ontological representation of the problem domain.

In this paper, we complement this portion of the methodology with a CASE tool. It has been developed as an Eclipse plug-in, for supporting designers in these two activities (Problem Domain Description and Goal Description). The Goal Identification and Modeling Tool (GIMT) has been conceived for being a MDE tool that automatically aids designers in some of their activities.

III. Goal Identification Approach

In the proposed methodology [17] ontologies are used very early in the process. Goal Description is the preliminary activity in which the analysts observe (and model) the strategic objectives of the software system and of its environment. This is done with a close iteration with another activity: the Problem Domain Description. This activity is responsible of creating a significant and transferable understanding of the portion of the world where to introduce the software. The collaboration of the two activities produces the elements of the environment (together with their significant states) that are collected and therefore used for formalizing user goals, thus avoiding ambiguities, discovering tacit knowledge and simplifying the comprehension among stakeholders.

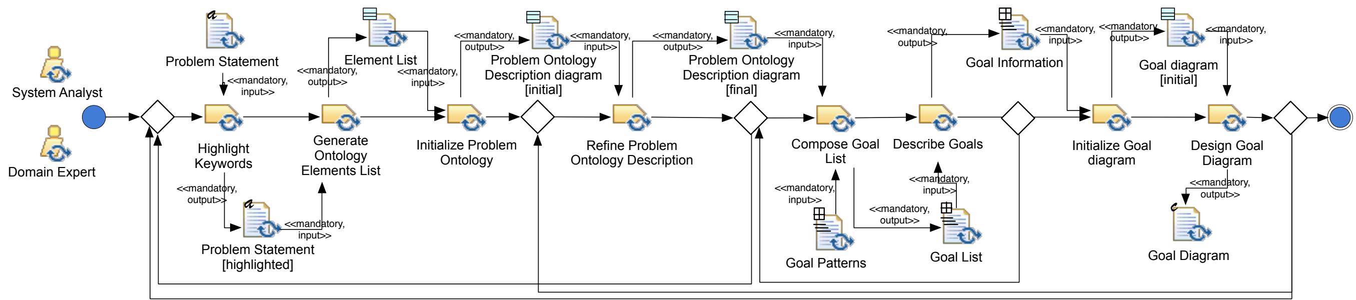

Fig. 3. The flow of work from Problem Statement to Goal Model.

In particular, the aim of *Problem Domain Description* (PDD) activity is to identify and to describe the problem domain elements and their relationships. This activity adopts the metamodel depicted in Fig. 1. Main elements are: concept (anything about which something is said), action (the cause of an event by an acting concept) and predicate (a property, a state or more generally a clarification to specify a concept). In particular we use to distinguish intentional actions (implying a kind of consciousness to act) from unintentional actions (purely reactive), and objects (concepts that can perform only unintentional actions) from positions (concepts with intentions).

This activity enables the *Goal Description* (GD) activity that grounds on the ontology and exploits some recurrent structures (Goal Patterns) that lead to identify goals (in terms of actor, and state transitions) and dependencies among them. The work product resulting from this activity is the Goal diagram whose system metamodel is shown in Fig. 2. This metamodel grounds on two main concepts: Hard Goal and Soft Goal. A HardGoal [10][16] represents a strategic interest of an actor. It is satisfied absolutely when its subgoals are satisfied, and that satisfaction can be automatically established. A SoftGoal [10][16] is a goal having no clear criteria for deciding whether it is satisfied or not.

Fig. 3 goes into details of the flow of work for identifying goals, starting from the problem statement and passing through the use of ontology. System Analysts and Domain Experts collaborate in Problem Domain Description and in Goal Description Activity.

In the ***Problem Domain Description Activity***, tasks are:

- *Highlight Keywords* - starting from an informal textual document describing the problem domain, an underlined text document where nouns have been highlighted, according to their grammatical function in the sentence, is produced;
- *Generate Ontology Elements List* - previously highlighted nouns are listed with respect to their types (Position, Object, Predicate, Intentional or Unintentional Action);
- *Initialize Problem Ontology* - a first draft of the problem ontology description diagram is prepared by using the previous list and a specific notation for representing each element of the diagram (the SMME and SMMR in the metamodel);
- *Refine Problem Ontology Description* - it is a refinement of the POD, by analyzing the the underlined Problem Statement. The final version includes relationships among ontology elements;

In the ***Goal Description Activity***, tasks are:

- *Compose Goal List* - this is the first activity for identifying goals, starting from the POD diagram. The Goal Patterns document is used for searching patterns in the POD diagram by following the guidelines illustrated in [17];
- *Describe Goals* - here the description on each goal is completed by describing the elements a goal is composed of: *goal type*, *name*, *state*, *who* is responsible for the goal and *dependencies*;
- *Initialize Goal Diagram* - a first draft of the goal diagram is prepared by using the previous goal list and a specific notation for representing each element of the diagram;
- *Design Goal Diagram* - it provides a structured description of the goals, their dependencies and the positions responsible for goal achievement, by using a specific notation.

The advantage of this methodological approach is correlating goals with the corresponding portion of textual requirements, thus supporting an iterative approach and a future evolution of the system. In order to support the designer's work during the goal identification phase, we developed a MDE tool, as an Eclipse plug-in, for which a specific modeling language has been created. This tool is introduced in the next section.

IV. Goal Identification and Modeling Tool

The Goal Identification and Modeling Tool (GIMT) is a CASE tool for supporting designers in performing all the tasks of the Problem Domain Description and Goal Description activity (see Fig. 3) for the representation of system requirements and the domain formalization, as proposed in [17].

GIMT has been realized as an Eclipse plug-in by using the Eclipse Modeling Framework (EMF) and Graphiti. It implements the Domain Specific Modeling Language we defined in order to create the models for representing the problem ontology and the goal identification.

The key element for the implementation of GIMT is the Model definition. The EMF provides a modeling and code generation framework for Eclipse applications based on a layered structure for data models. The information type of the sets of model instances is defined in a so-called core model, corresponding to metamodel in the Essential MOF (EMOF). Ecore is the metamodel adopted for core models. It contains the following elements: EPackage, EClass, EDataType, EAttribute and EReference.

Usually, in our work we use a system metamodel for formalizing the definition of all the elements and relationships, and for representing constraints on the enactment of design processes or part of them. Examples of metamodels are reported in Figures 1 and 2. They respectively describe the Problem Domain Description and Goal Description activities. The metamodelling techniques [20] are based on a metamodelling layered architecture, that follows the principles of MOF and OMG [15]. As anticipated before, constructing plug-in in Eclipse implies the definition of models that are based on Ecore. Therefore we map our metamodel elements (SMME and SMMR) on EClasses and the relations on ERelations, as specified in the Ecore notation. Moreover, starting from an EMF model, a set of Java classes have been generated for the model.

Summarizing, GIMT is based on two metamodels, one for each supported diagram, (see Fig. 1 and Fig. 2) and on a graphical notation to represent the specific design elements. GIMT has been conceived to give the user the possibility to create two different diagrams for representing the problem domain ontology and the goal model according to the guidelines we previously introduced.

In the following subsections we present the adopted notation and its usage into the specific GIMT diagrams.

A. *Diagrams and Notation*

GIMT supports the Problem Ontology Description and the Goal diagrams. The graphical notation defined for our DSML and implemented in these diagrams is shown in Fig. 4 and it refers to the abstractions we use in our metamodels (see Fig. 1 and Fig. 2).

As regard the Problem Ontology Description diagram, it adopts the notation (see the left side of Fig. 4) defined for representing the concrete elements and the relationships of the Problem Ontology Description Metamodel (see Fig. 1).

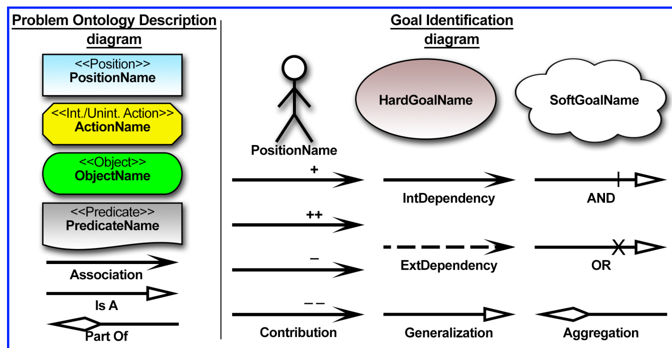

Fig. 4. The notation adopted in GIMT for Problem Ontology Description and Goal diagram.

POD diagram elements - Each element of a POD diagram is represented by means of a graphical notation and a specific stereotype. In GIMT, each element is also colored differently to easily distinguish it, especially in large diagrams. Elements in a POD diagram may be:

- *Intentional and Unintentional Action* that are represented as a cut corners rectangle with an Action Name. They are differentiated by means of the stereotype. They may be also connected to other elements such as Object, Position and Predicate.
- *Object* is represented as a round corner rectangle with an Object Name. Object may be input or target of an action. It may be connected with predicates or other objects.
- *Position* is represented by a simple rectangle with a Position Name. A position may be connected only to actions and other positions.
- *Predicate* is depicted as a little sheet with a Predicate Name. It may be connected with Position, Action and Object.

POD diagram relationships - The elements in a POD diagram can be related to each other by different kinds of relationship, whose semantic is quite intuitive[2]. Relationships in a POD diagram may be:

- *Association* that is represented as usual by an arrow. Elements in the diagram can be logically related with each other by using Associations.
- *Is A* is represented by the traditional symbol for generalization. Is-a is the relationship between an ontological element and one of its refinements.
- *Part Of* is represented by the traditional symbol for composition. This relationship represents the whole-part relationship among ontological elements.

The Goal diagram adopts the notation (see the right side of Fig. 4) defined for representing the concrete elements and the relationships of the Goal Description Metamodel (see Fig. 2).

Goal diagram elements - The graphical notation of the Goal diagram is derived from a well-known graphical notation [10]. Elements in a Goal diagram may be:

- A *Hard Goal* is represented by an ellipse with a Name. It is always connected with at least one Position. It can be also related with other Hard Goals and Soft Goals.
- A *SoftGoal* is depicted as a cloud with a Name. It can be related to Hard Goals.
- A *Position* is referred to the same element described in the POD diagram, but in this diagram it is depicted as a sticky man.

[2]Detailed definitions can be found in [17].

Goal diagram relationships - The elements of a Goal Diagram can be logically related to each other by using the following kinds of relationships:

- *Aggregate* is the only relationship that can exist between Positions and Goals. Its notation looks like the UML "part-of" relationship: a line starting with an empty diamond.
- *Contribute* that is the relationship that can relate HardGoal with SoftGoal and vice-versa. There are four different types of contribution: positive, strongly positive, negative and strongly negative [10]. Each one is depicted with its own dedicated notation, as shown in Fig. 4.
- *Depend* relates two different goals. It may be an Internal Dependency or an External Dependency. Notations are respectively a simple arrow and an arrow with dashed line.
- *Generalize*, very similar to the UML inheritance, it specifies a relationship between Positions, in which specialized positions inherit features from the general position.
- *Decompose* relates two Goals. There are two kinds of decomposition: AND and OR. Both of them are depicted as an arrow and a specific symbol (see Fig. 4).

B. Ecore Metamodel Mapping

Adopting a design process for developing a system generally means managing a set of abstractions (concepts of the domain) that may be instantiated in one ore more work products. In this work we use system metamodelling as one of the fundamental elements for the construction of CASE tools. In particular, our technique [20] prescribes that a system metamodel is composed of:

- *Element* (SMME) is a construct of the metamodel that can be instantiated into elements of the system model;
- *Relationship* (SMMR) is a construct used for representing the existence of a relationship between two (or more) instances of elements. For instance, the aggregation relationship between two instances of the element class is an instance of the "association" SMMR.
- *Attribute* (SMMA) is a construct used for adding properties to SMMEs.
- *Operation* (SMMO) is a construct used for describing additional proprieties of an SMMEs.

Eclipse EMF makes the domain concepts explicit. It distinguishes between model and metamodel and uses the metamodel for ruling the structure of a model. The Ecore metamodel is the part of the EMF metamodel dedicated to define the following elements:

- *EClass* represents a class, with zero or more attributes and zero or more references.

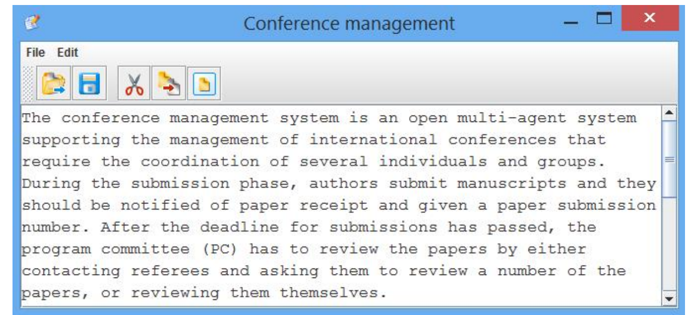

Fig. 5. Problem Statement Editor

- *EAttribute* represents an attribute which has a name and a type.
- *EReference* represents one end of an association between two classes. It has a flag to specify if it represents a containment and the cardinality.
- *EDataType* represents the type of an attribute, e.g. int, float or java.util.Date.

For space reasons we do not provide further details about how to create an EMF editor plug-in but it is worth to note that the mapping is one to one in most of the cases: any element of the system metamodel has a direct counterpart with an element of the Ecore metamodel. Therefore, it is pretty straight to obtain an Ecore metamodel starting from a system metamodel (like those in Fig. 1 or Fig. 2).

C. Using GIMT

GIMT provides three kinds of editor: a textual editor for manipulating the problem statement and two diagram editors for modeling respectively the Problem Ontology Description (POD) diagram and the Goal diagram. Moreover, GIMT has been endowed with some functionalities that allow to automatically perform transitions between the tasks of our process.

This section aims at describing how to employ GIMT as a CASE tool for the portions of design process described in Fig. 3. The Conference Management System (CMS) case study[3] has been used for exemplifying the description.

At this stage it should be clear that the flow of work described in Fig. 3 is logically divided into two main portion of work: the Problem Domain Description activity and the Goal Description activity. The first devoted to deliver POD diagram in its final version and the second resulting in the creation of the Goal diagram. The GIMT tool aims at supporting the designer in the creation of these two work products.

As it can be seen in Fig. 3, the first three tasks of the Problem Domain Description Activity: *Highlight Keywords*, *Generate Ontology Element List* and *Initialize Problem Ontology*. These have to be performed following the guidelines briefly introduced in Section III. GIMT provides a text editor for loading or creating textual documents (such as a Problem Statement), in order to make easy the use of our guidelines. It also comes with particular editing functionalities that allow the identification of ontology elements.

[3] A complete description of the CMS case study may be found in [22].

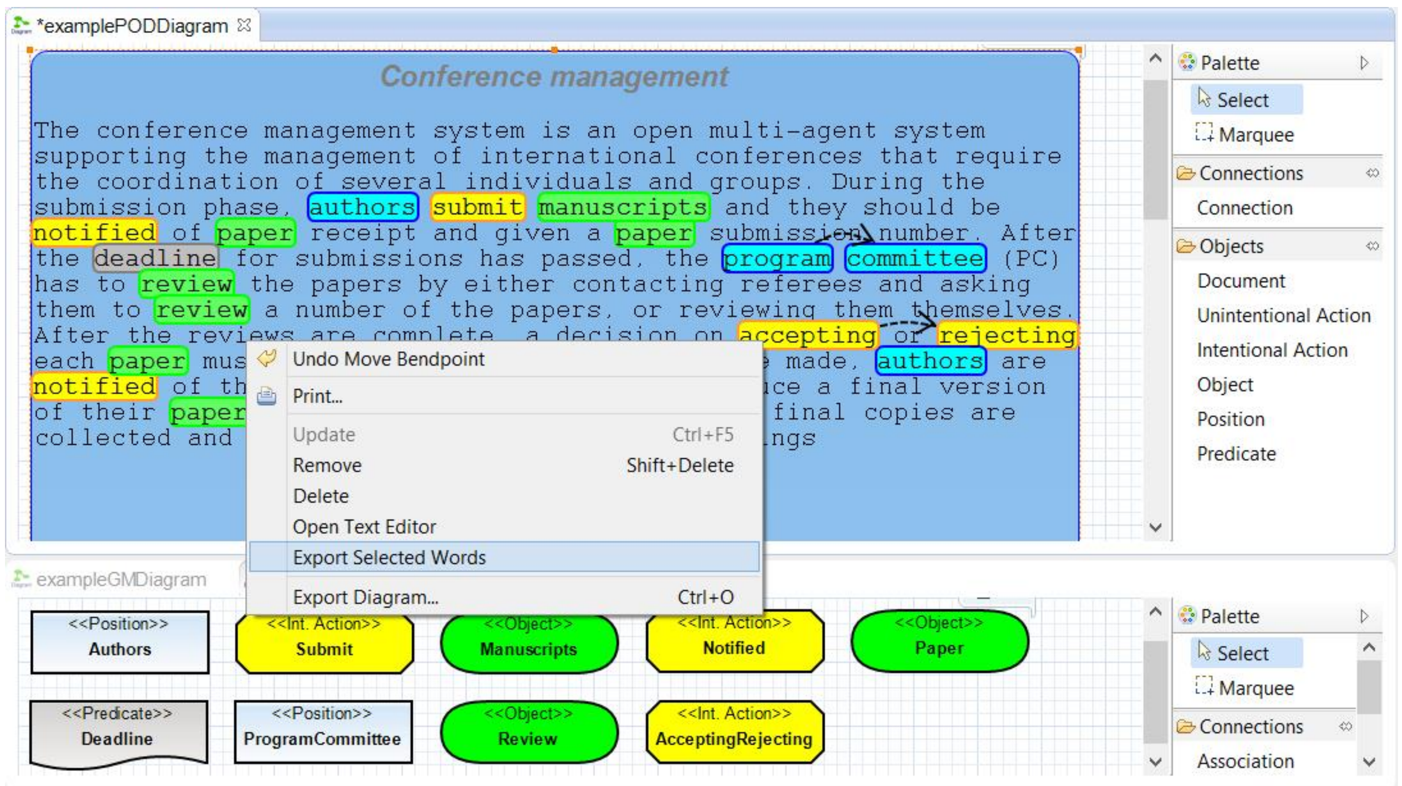

Fig. 6. Transition from Problem Statement Highlighted to Problem Ontology Description Diagram Initial.

The *Document* is the key element of the Problem Statement Editor (PSE), shown in Fig. 5. In a Document the designer can highlight words thus identifying specific ontological elements (such as Intentional Action, Position and so on) according to our guidelines [17]. For instance in the CMS problem statement (see upper side of Fig.6) the words *authors* and *manuscript* have been identified respectively as a Position and an Object. Then, these ontological elements can be automatically exported into a POD diagram where they will be represented according to their notation (see lower side of Fig. 6).

Fig. 6 represents the portion of the work devoted to produce the POD diagram in its initial version (see Fig. 3). In fact, in the upper part of Fig.6, the words highlighted with appropriate labels correspond to ontological elements that are added to a list (the *Element List*[4] work product). Then, elements may be exported from the list to be added to the *Problem Ontology Description diagram [initial]* for its refinement.

Moreover, the GIMT Problem Statement Editor provides also functionality for grouping words to be identified as a unique ontological element by means of a *Connection*. A Connection is a link that allows to relate also two non-consecutive text portions. For instance, in the CMS case study (see upper side of Fig. 6) we want to identify the words *accepting* and *rejecting* as the same Intentional Action. To do this, we firstly select the two words and then we relate them by means of a Connection (represented as a dashed arrow). As a consequence, GIMT will automatically refers them as a unique ontological element named *AcceptingRejecting* (see lower side of Fig. 6). Hence, it is worth to point out that the *Initialize Problem Ontology* task can be performed automatically with GIMT by means of an exporting functionality implemented in the Problem Statement Editor.

In order to complete the Problem Domain Description Activity, the last task is the *Refine POD* (see Fig. 3). GIMT is also endowed with an appropriate diagram editor that allows to handle the structural aspect of problem ontology elements by organizing them and by adding relationships or other elements. The upper side of Fig. 7 shows the POD diagram for the CMS case study. This diagram has been obtained by refining its initial version (see the lower side of Fig. 6) derived from the previous task. The POD editor allows to edit a POD diagram, by managing elements imported from the previous phases, but also adding new ontological elements and relationships. For instance, the Position *author*, the Intentional Action *submit* and the Object *manuscript* previously identified (lower side of Fig. 6) have been opportunely related (top/right side of Fig. 7).

The *Refine POD* task completes the Problem Domain Description activity and the POD diagram [final] is the resulting artifact. So far, the second activity of our process can start (Fig. 3): the Goal Description. As our guidelines prescribe, the first task is *Compose Goal List* that identifies specific patterns from the POD diagram. Fig. 8 shows an example of a common pattern[5] that can be discovered in a POD diagram. It is worth to note that a pattern is characterized by compartments: i) a generic schema of the ontological elements and ii) the

[4]The *Element List* is not clearly visible to the designer. It maintains information about ontological elements and it may be exported by the tool.

[5]Further detail of patterns can be found in [17].

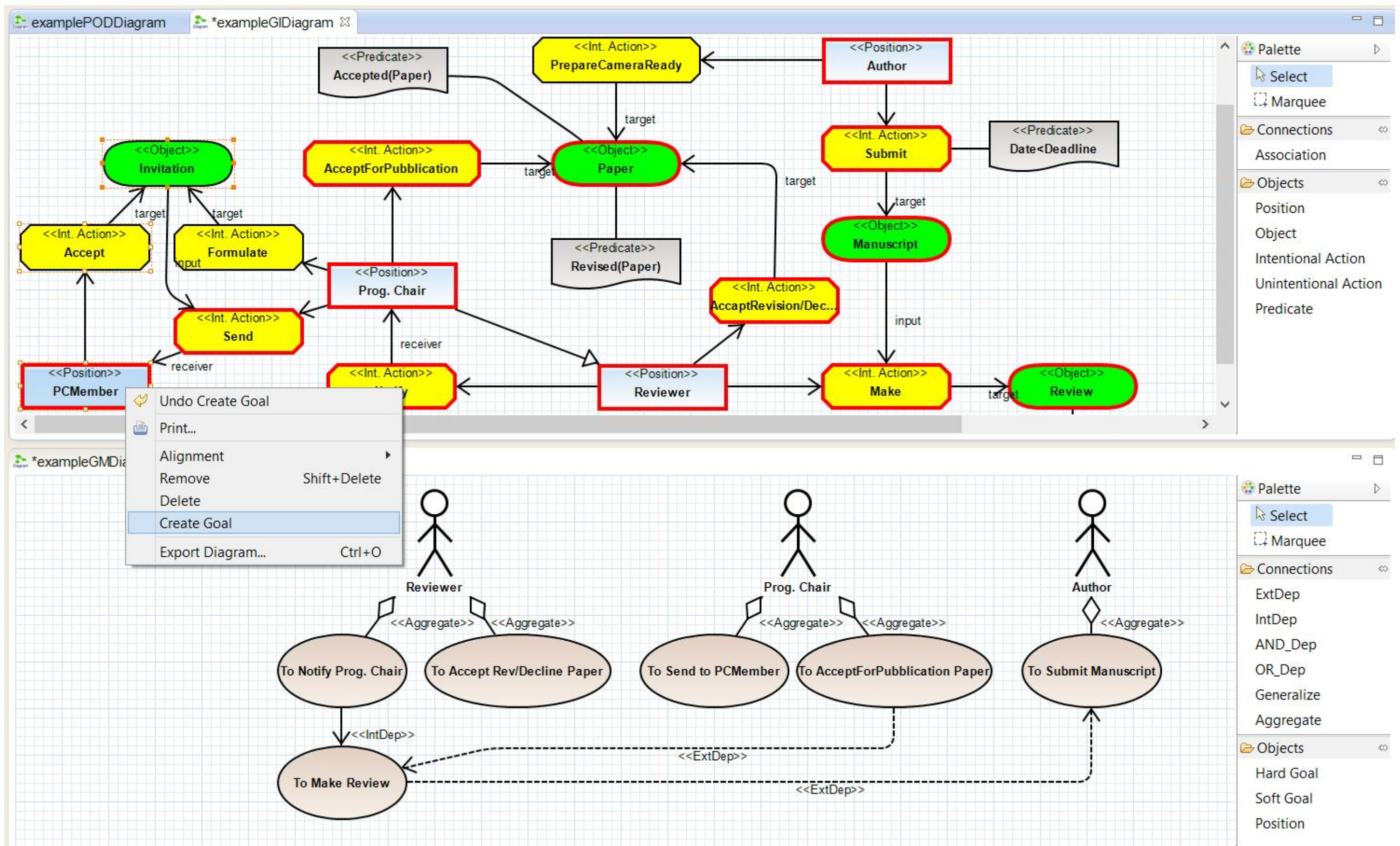

Fig. 7. Transition from Problem Ontology Description Diagram to Goal Model.

description of goal information that can be extracted if the pattern matches (goal type, goal name and so on). For instance, the ontological elements triple, *author*, *submit* and *manuscript* in the upper side of Fig. 7 corresponds to the pattern shown in Fig. 8. The POD editor supports this task, by allowing multiple selection of elements, thus generating the goal related to the specific pattern and adding it to the goal list. A thicker red border is used to highlight the elements already selected by the designer.

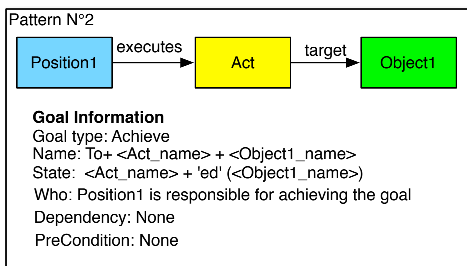

Fig. 8. Pattern extracted from [17].

The POD editor also provides a functionality in order to automatically perform the second task of the Goal Description activity, that is *Describe Goals*. This task consists in setting the goal information according to the identified pattern. A specific form (see Fig.9) allows to add all the information that is not automatically settable, such as the information about the Dependency. For instance, the triplet previously identified corresponds to a goal with the following, automatically set, information(see Fig. 9): Goal Type=*Achieve*, Name= *To submit manuscript*, State= *submitted manuscript*, Who= *author*.

Finally, GIMT supports the *Design Goal Diagram* task by providing a Goal Diagram Editor. Previously created goals may also be automatically exported in a goal diagram. The functionality implemented in the Goal Diagram Editor allows the designer to describe more in detail the dependencies between the imported goals and the Positions that perform them. For instance, by applying the pattern of Fig. 8, another goal can be extracted from the POD diagram of the CMS case study: *To make review* (Fig. 7). The goal *To make review* and the goal *To submit manuscript* in the initial version of the Goal Diagram were not related. By reasoning on the diagram with the support of our guidelines, the designer is able to discover a Dependency among these two goals and to refine the diagram.

It is worth noting that our process is iterative. Thus, at

Set Goal Informations

Goal type:	Achieve
Name:	to Submit Manuscript
State:	submitted(Manuscript)
Who:	Author
Dependency:	
PreCondition:	Date<deadline

OK Annulla

Fig. 9. The Goal Information frame.

each iteration it is important to trace (forward and backward) the ontology model with the goal model. GIMT maintains a traceability of the elements during model transformation. Each diagram of GIMT, in fact, uses a single persistence file model. A typical scenario that can occur is, for example, the deletion of a goal from the goal diagram. In this case, the linked ontological elements will be affected. The tool automatically removes the ticker border in the POD diagram. This functionality allows also to create a direct link between a goal and portion of problem statement from which it derives. This is important when conflicting or inconsistent goals are discovered. Thus, the possibility to go back to the textual description that originated the problem may help to solve the inconsistency.

V. Conclusions

GIMT has been designed and developed in order to overcome the limitations of the manual approach proposed in [17] especially when it has to be applied to large size problems. Such an approach grounds on two main models (an ontology and a goal model) and on a set of guidelines allowing model to model transformations. Hence, GIMT has been conceived as a CASE tool based on a DSML opportunely defined to support the goal oriented requirement analysis proposed in [17]. It has also been endowed with some functionalities that support the designer in applying the guidelines to perform the activities of the approach (i.e: the Problem Domain Description and the Goal Description activity). Moreover, in some cases GIMT automatically executes some tasks of the process.

The work illustrated in [17] is a part of a broader work towards the creation of a complete methodological approach for developing multi-agent systems to be implemented in the JACAMO framework. Thus, new models will be included in order to face other design issues. Hence, we decided to develop GIMT as an Eclipse plug-in by using the Graphity framework, thus ensuring us a rapid development and integration of new diagram editors and a great flexibility to extend our DMSL.

We found in Ecore a very easy and fast way to produce DSML due to its almost one to one mapping with our metamodeling techniques.

References

[1] Actifsource, available at http://www.actifsource.com.

[2] Eclipse Modeling Project, available at http://www.eclipse.org/modeling/.

[3] EMF - Eclipse Modeling Framework, available at http://www.eclipse.org/modeling/emf/.

[4] GMF - Eclipse Graphical Modeling Framework, available at http://www.eclipse.org/modeling/gmp/.

[5] Graphiti - Graphical Tooling Infrastructure, available at http://www.eclipse.org/graphiti/.

[6] Microsoft visual studio sdk including domain-specific language tools, available at http://msdn.microsoft.com/en-us/library/bb126259.aspx.

[7] U. Aßmann, S. Zschaler, and G. Wagner. Ontologies, meta-models, and the model-driven paradigm. *Ontologies for Software Engineering and Software Technology*, pages 249–273, 2006.

[8] C. Atkinson and T. Kuhne. Model-driven development: A metamodeling foundation. *IEEE Software*, 20(5):36–41, September/October 2003.

[9] O. Boissier, R.H. Bordini, J.F. Hübner, A. Ricci, and A. Santi. Multi-agent oriented programming with jacamo. *Science of Computer Programming*, 2011.

[10] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.

[11] M. Cossentino, A. Chella, C. Lodato, S. Lopes, P. Ribino, and V. Seidita. A notation for modeling jason-like bdi agents. In *Complex, Intelligent and Software Intensive Systems (CISIS), 2012 Sixth International Conference on*, pages 12–19. IEEE, 2012.

[12] M. Cossentino, C. Lodato, S. Lopes, P. Ribino, V. Seidita, and A. Chella. A uml-based notation for representing mas organizations. In *WOA*, pages 133–139, 2011.

[13] M. Cossentino, C. Lodato, S. Lopes, P. Ribino, V. Seidita, and A. Chella. Towards a design process for modeling MAS organizations. In Massimo Cossentino, Michael Kaisers, Karl Tuyls, and Gerhard Weiss, editors, *Multi-Agent Systems*, volume 7541 of *Lecture Notes in Computer Science*, pages 63–79. Springer Berlin Heidelberg, 2012.

[14] S. Kelly and J.P. Tolvanen. *Domain-specific modeling: enabling full code generation*. John Wiley & Sons, 2008.

[15] OMG meta object facility, version 2.4.2, april 2014. doument formal/2014-04-03, available at http://www.omg.org. http://www.omg.org/technology/documents/formal/mof.htm.

[16] J. Mylopoulos, L. Chung, and E. Yu. From object-oriented to goal-oriented requirements analysis. *Communications of the ACM*, 42(1):31–37, 1999.

[17] P. Ribino, M. Cossentino, C. Lodato, S. Lopes, L. Sabatucci, and V. Seidita. Ontology and goal model in designing bdi multi-agent systems. In *WOA@AI*IA, Proceedings of the 14th Workshop "From Objects to Agents" co-located with the 13th Conference of the Italian Association for Artificial Intelligence (AI*IA 2013), Torino, Italy*, volume 1099, pages 66–72, December 2-3 2013.

[18] P. Ribino, C. Lodato, S. Lopes, V. Seidita, V. Hilaire, and M. Cossentino. A norm-governed holonic multi-agent system metamodel. In *Agent Oriented Software Engeneering (AOSE)*, 2013.

[19] D.C. Schmidt. Model-driven engineering. *Computer*, 39(2):25–31, Feb. 2006.

[20] V. Seidita and M. Cossentino. Metamodeling: Representing and modeling system knowledge in design processes. In *In Proceedings of the 10th European Workshop on Multi-Agent Systems, EUMAS 2012*, pages 103–117, 2012.

[21] J.P. Tolvanen and M. 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.

[22] F. Zambonelli, N. Jennings, and M. Wooldridge. Organizational rules as an abstraction for the analysis and design of multi-agent systems. *Journal of Knowledge and Software Engineering*, 2001.