Model Integration in Agent-Oriented Development

Rubén Fuentes-Fernández, Jorge J. Gómez-Sanz, Juan Pavón

Dep. Ingeniería del Software e Inteligencia Artificial, Universidad Complutense de Madrid, c/ Profesor José García Santestmases s/n, 28040 Madrid, Spain {ruben, jjgomez, jnpavon}@pdi.ucm.es

Abstract: Current agent oriented methodologies focus each one on some aspects of multi-agent systems and may leave aside others. For this reason, a developer may find useful to apply different specialized methods and tools for each aspect of the system under development, depending on the problem characteristics. Nevertheless, the integration of different elements into a specification elaborated with an already existing methodology is not trivial. This paper proposes addressing this issue by using mappings between different modelling languages. This will permit developers to work with the most convenient specification method for their problem at every moment. In order to reduce the number of required mappings, we propose an intermediate language called UML-AT, from which all of the correspondences are performed. Being rooted on the Activity Theory framework, which comes from Social Sciences, UML-AT supports most of the key agent-related concepts found in literature. The application of this integration approach in a development is illustrated with a case study about the implementation of FIPA protocols. The case study involves the use of a software support tool for the translation.

Keywords: Method Engineering, Integrated Development, Conceptual Mappings, Activity Theory, UML-AT.


Bibliographical notes: Rubén Fuentes-Fernández obtained his PhD on Computer Science in 2004 from the Universidad Complutense de Madrid, Spain. He is Associate Professor at this university since 2002 and member of the grasia! research group on Agent Technologies. Previously, he worked as consultant and project manager in database systems. His main interests are concerned with multi-agent systems methodologies and the application of techniques from Social Sciences to the development of such systems.

Jorge J. Gómez-Sanz has a degree in Software Engineering and completed a PhD in Computer Science in 2004 from the Universidad Complutense of Madrid, Spain. He is Associate Professor at this university since 2002 and member of the grasia! research group on Agent Technologies. Previously, he worked as consultant and project manager in database systems. His main interests are concerned with multi-agent systems methodologies and the application of techniques from Social Sciences to the development of such systems.

Juan Pavón-Mestras holds a PhD degree in Computer Science from the Universidad Politécnica de Madrid, Spain, (1988). He has been working in the R&D departments of Alcatel in Spain, France, and Belgium, and in Bellcore, USA, especially involved the development of software component architectures for distributed systems. Since 1998 he
1. Introduction

Agent Oriented Software Engineering (AOSE) comprehends a large number of methodologies and techniques to build Multi-Agent Systems (MAS). Existing approaches cover a wide range of perspectives about these systems and their development: what kind of specific features these abstractions should exhibit; what relevant information for analysis must be gathered; what tasks must be carried out to obtain that information and when; or how the specifications should be transformed into computational entities. These methodologies usually differ because of the diversity on existing views about what are agents and MAS. Each methodology relies on these views and tends to highlight a specific aspect of the system, the one the methodology considers the most important. The literature reports methodologies specialized in goals (e.g. KAOS in Dardenne, van Lamseweerde, and Fickas, 1993; Tropos in Castro, Kolp, and Mylopoulos, 2001), the mental state of agents (e.g. AALI/BDI in Kinny, Georgeff, and Rao, 1996; ADELFE in Bernon, Gleizes, Peyruqueou, and Picard, 2002), interconnected tasks (e.g. TÆMS in Decker, 1996; DESIRE in Brazier, Dunin-Keplicz, Jennings, and Treur, 1997), or interactions (e.g. PASSI in Cossentino, 2005; INGENIAS in Pavón, Gómez-Sanz, and Fuentes, 2005). An updated overview of the state of the art in AOSE is provided by Henderson-Sellers and Giorgini (2005). Hence, it is necessary considering which aspects are more relevant in the domain problem before starting to work. With respect to the not so relevant aspects, the chosen methodology may deal with them properly or not. If it does not, developers may be forced to combine other methodologies and techniques to provide a complete solution for the problem.

This practice of integrating techniques is quite common in Software Engineering. A developer using the Unified Process and UML as modelling language may decide to describe the system databases with a relational model. In principle, this is a problem of integration of two different ways of development. In practice, the developer considers both as independent problems. When constructing a database specification, the developer looks for normal forms and transforms the initial data structures accordingly. This information is incorporated into documents explaining how the database is organized and why. When using the database, the developer builds APIs modelled with UML and integrates these APIs into the system architecture. Clients of those APIs may have no knowledge at all about how the database is built. By means of this separation of concerns, integration of different techniques and specifications happens.

This way of working could be applied as well to MAS. A developer could use KAOS to elicit the goals and to identify the agents, models from TÆMS to specify the interconnection of tasks, and INGENIAS interaction models to describe the collaboration between the agents. Unfortunately, keeping the different specifications independent is harder in a MAS specification than in the previous example with UML and databases. It is more likely to happen that goals from KAOS had to be available to specify tasks in TÆMS models, and both of them to characterize interactions with INGENIAS. Thus, the specification of the real system would result from the combination of different specification documents. How this combination happens would be a problem left to the readers of these different documents, i.e. the developers and engineers.
Model Integration in Agent-Oriented Development

A solution for the abovementioned incompleteness of existing methodologies that avoids users combining specifications would be creating customized methodologies for specific problem domains with method engineering (Beydoun, Gonzalez-Perez, Low, and Henderson-Sellers, 2005; Henderson-Sellers, 2005). The method engineering approach defines a generic metamodel to represent the abstract syntax and semantics of agent-oriented modelling languages. This metamodel is conceptually derived according to the needs of representing new methods. When the language of a methodology is represented in terms of the metamodel, the tasks and products of that method can be added to a repository. This repository allows generating ad-hoc methodologies for specific projects. Indirectly, one is constructing new semantic models as well for the new methodology. An example of extension of the metamodel can be found in (Beydoun, Gonzalez-Perez, Low, and Henderson-Sellers, 2005) and one of the generation of a method from the repository in (Henderson-Sellers, 2005). In this way, engineers and developers work with a unique methodology, so they do not need to share information between different models. Despite its advantages, there is still an absence of guidelines about what concepts the metamodel should include (except that they appear in some agent-oriented methodology) and no tool exists yet to work with the approach automatically. Besides, the cost of creating a brand new methodology cannot be disregarded. It could be affordable only if the new methodology is going to be used repeatedly.

An alternative to method engineering consists in creating a system able to connect consistently the specifications from different methodologies. In the above scenario, it would be necessary to remember which part of which specification is connected to the others. That is, which models can be understood by looking at a single specification and which ones require to look at other specifications as well. A system performing these tasks faces one main obstacle, namely the incompatibility among existing methodologies support tools. The system must access the content of the different specifications and be notified whenever something changes in any of them. This problem could be solved by means of unified APIs implemented by tools. However, even if those APIs existed, it would remain the information integration problem, i.e., how integration of the different specifications can happen. As an answer, this paper contributes with an integration framework composed of two elements: a process for information sharing across specifications and an intermediate language called UML-AT.

The information integration process makes a distinction among pieces of information which are shared and those which are not. A concept or a network of concepts in a specification is shared when its creation can involve the creation of equivalent concepts or networks of concepts into other related specifications. The production of new pieces of information would follow the semantics associated to each individual specification method. Similarly, their deletion will involve the deletion of concepts or networks of concepts into other connected specifications, and according to the semantics of each model. It is not suggested to share every piece of information. This in fact may not be possible, since there are concepts in some methodologies which cannot be represented in others. Those elements that cannot be shared would remain in the scope of the individual specifications. The influence of non-shared concepts in the resulting combination of specifications is close to zero. After all, non shared concepts are labelled that way because they cannot be expressed into other specification methods.

The information sharing act is realized by means of sets of mappings that describe correspondences between networks of concepts of different methods. Having those mappings, it is straightforward to follow them and obtaining the equivalent elements in
each methodology, if they exist. These mapped pieces of information would stand for the shared concepts or networks of concepts across specifications mentioned before.

The definition of mappings could be direct, just telling what networks of concepts connect with what. However, it is not practical in the sense that given N methodologies, the number of mappings between methods needed to work with all of them at the same time is N²-N. It is better to choose an intermediate representation and create mappings to this one. This way, the number of links needed is 2*N. These formulas tell that for N<3 it is better to choose the first alternative. Nevertheless, the second alternative is better in the real case, where the number of existing methodologies is quite bigger than three. With this choice, by reusing the mappings from a few developers, the whole community would obtain the benefit of the freedom of specification method. Besides, new created methodologies would have to provide a single pair of mappings to be incorporated in the pool of eligible methodologies.

The nature of this intermediate representation was chosen to be close to agent oriented specification methods and generic enough to capture as much of these methods as possible. With this purpose, a semi-formal language called UML-AT was considered. This language bases on the concepts of the Activity Theory (AT) (Vygotsky, 1978), a framework from Social Sciences with precedents of use in AOSE (Fuentes, Gómez-Sanz, and Pavón, 2004; Omicini, Ricci, Viroli, Castelfranchi, and Tummolini, 2004). The Social Sciences roots of UML-AT make it generic enough as intermediate language. Since the consider concepts from AT are rather abstract, necessarily, UML-AT is high-level as well. At the same time, UML-AT is close to AOSE, though it does not depend on it. UML-AT captures the intentional nature of human beings. This is highly related with AOSE where most approaches do consider agents as intentional entities, i.e. entities that execute actions with a purpose. Besides, human societies have inspired relevant advances in agent research. See, for instance, the works about the BDI model (Bratman, 1987) or social relationships (Castelfranchi, 1989). As pointed out before, there are precedents of the transfer of knowledge from AT to agent research, such as (Omicini, Ricci, Viroli, Castelfranchi, and Tummolini, 2004) and (Fuentes, Gómez-Sanz, and Pavón, 2004). The first one illustrates how the perspectives from Social Sciences can help to detect missing concepts in methodologies, in this case focusing on the relevance for MAS of the coordination artifacts that mediate communication. The second work shows how AT techniques for requirements elicitation and the management of contradictions are adapted for their use in agent-oriented methodologies. Despite these connections with the agent research, UML-AT it is not dependent on any particular view of MAS, since its main concern are human groups. Therefore, it should not be expected any bias from concrete agent research.

The rest of the paper is devoted to discuss the elements of this introduction with further details. Section 2 and 3 describe the components of the integration framework. UML-AT is presented in section 2 and the components of mappings between this language and those of agent oriented-methodologies in section 3. The specification integration process, which is described in section 4, uses these elements to deal with specifications in different languages. The framework is applied in a case study about the implementation of FIPA protocols in section 5. Finally, section 6 discusses the overall approach according to the results of the experimentation and points out some open issues.
2. UML-AT

The foundations of UML-AT can be found in the framework of AT, which will be introduced in section 2.1. Starting from this theory, UML-AT proposes a description of AT concepts using UML. This structure has its own semantics, inspired in the original AT ones. The UML representation for AT will be presented in section 2.2.

Activity Theory

Activity Theory (AT) is a paradigm for interdisciplinary research in Social Sciences that has its origin in the socio-cultural approach initiated in early 1920s by Vygotsky (1978). It is a comprehensive framework to understand and represent social activities in complex settings, and then appropriate for agent-oriented systems. AT constitutes the conceptual and analytical basis of this research, including its metamodel. The main reason for this choice is that AT offers not only a useful modelling framework adaptable to study MAS, but also knowledge and techniques that can provide new insights in the intentional and social issues of these systems. About modelling, AT works with concepts of intentional entities and purposeful activities similar to those common in the agent-oriented approach. However, AT focuses on the activity itself and its socio-historical context instead of on the subject or group that executes the activity. When the interest of the analysis is on the process and the related people, this perspective is more suitable than that common in agent-oriented methods that is focused on the agents and their interactions (Sierhuis and Clancey, 2002). This kind of domains where interactions between the system and humans are key is one of the main areas of application where developers can profit from the particular features of the agent approach (Luck, McBurney, Shehory, and Wilmott, 2005).

Moreover, other disciplines of Computer Science concerned about these domains have already adopted AT as one of their main theories to understand and analyze the development of human working activities. Besides this modelling advantage, the knowledge and experience of AT in the study of human societies can inspire new research in the agent-oriented field. For instance, Fuentes, Gómez-Sanz, and Pavón (2004) describe how the principle of the evolution of human societies as a consequence of their contradictions can be extrapolated to explain emerging behaviours in MAS.

The term activity (Leontiev, 1978) refers to a process intended to transform some object in order to satisfy the needs of the subject of that activity. The scope of its analysis comprehends the contextualized processes of the system as a whole. These contextualised processes are called activity systems (Engeström, 1987) and constitute the minimal meaningful context to understand human actions. An activity system includes the subject (i.e. a participant or a group) whose agency is chosen as the point of view in the analysis and the acted on object as well as the dynamic relations among both. All these relations between subject and object are not direct; rather, they are mediated by various factors, including tools, community, rules, and division of labour.

Objects can be raw materials, conceptual understandings, or even problem spaces. The activity is directed at objects that are molded or transformed into outcomes with the help of physical and symbolic, external and internal tools. These outcomes are the products that satisfy the subject’s need (i.e. the objective) that motivates the performance of the activity. The community of a system refers to those subjects who share the same objects. Communities are defined by their division of labour and rules. Specifically, divisions of labour can run horizontally as tasks are spread across members of the community with equal status, and vertically as tasks are distributed up and down divisions.
of power. Last, *activity systems* are constrained by the formal (systematic, general, and expected), informal (idiiosyncratic adaptation), and technical (mandated and, potentially, written) rules, norms, and conventions of the *community* that are encompassed under the term *rules*.

It is also important to note that an *activity system* is made up, and embedded in at the same time, of networks of nested *activities* and *actions* (i.e. tasks that pursue goals non-relevant by themselves but that contribute to a primary goal). All of which could be conceived of as separate *activity systems* depending on the observer’s perspective. In such networks, factors that affect the current *activity system* could be the *outcome* of previous actions.

From an AT perspective, the examination of any phenomenon must consider the dynamics among all these components across time and space. These mediating elements make explicit the influence of the environment in which the *activity* happens and the historical and social development of it. Although the agent-oriented research considers activities in MAS as social interactions with their regulations, this knowledge about the developmental context is an aspect largely missed. Nevertheless, such a context is a mandatory prerequisite to truly understand the whole system and its ruling principles and to improve its engineering (Winograd and Flores, 1986).

**UML-AT: Representing Activity Theory with UML**

Some of the AT concepts are adopted for the conceptual basis of UML-AT. Table 1 shows their rewritten definitions for UML-AT. It should be clarified in advance that some concepts have slightly different meanings to those in AT. This is the case for the concepts of *artifact* and *tool*. An *artifact* is usually regarded in AT as a synonym of *tool*, while UML-AT uses it for a generalization concept; on the other side, a *tool* in UML-AT is defined as a complementary concept for *object* and *outcome* in the *activity system*. The reason for the changes is that these definitions are intended to be used by development teams and automated support tools in Software Engineering settings (see section 4 for more about this topic). An accurate description according to AT would require a very rich and complex language (Social Sciences use natural language) hardly applicable in this context. Thus, both the definitions in Table 1 and UML-AT make some simplifying assumptions. In this sense, future work will enrich UML-AT with formal descriptions that allow reducing the gap between definitions in AT and UML-AT.

Concepts from Table 1 are represented in UML-AT, a modelling language that allows describing social systems composed by intentional and autonomous actors with the concepts of AT. To ease the adoption of UML-AT by the AOSE community, it has been defined as an extension of UML (OMG, 2005) using the mechanism of profiles. Concepts are expressed as stereotypes that inherit from the metaclass *Class* and relations are expressed as stereotypes that inherit from the metaclass *Association*. Additional constraints, for instance the stereotypes that a relation links, are expressed with Object Constraint Language. Figure 1 shows the main concepts of AT with UML-AT. To simplify the depiction of this and following diagrams, concepts are represented with stereotypes and relationships as UML associations.
### Model Integration in Agent-Oriented Development

**Table 1.** AT concepts for the agent paradigm.

<table>
<thead>
<tr>
<th>AT concepts</th>
<th>Definition with the agent paradigm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activity System</td>
<td>Set of modelling elements representing a total or partial view of the MAS.</td>
</tr>
<tr>
<td>Activity</td>
<td>Task that pursues some relevant objective for an agent or group of agents in the MAS.</td>
</tr>
<tr>
<td>Subject</td>
<td>Agent, role, or organization. In general, whatever active element that behaves according to the Rationality Principle can be modelled as a subject. The Rationality Principle states that “Every action taken by an agent aims at achieving one of its goals” (Newell, 1982).</td>
</tr>
<tr>
<td>Object</td>
<td>An element transformed by a task and that can be considered as the reason/origin for that task. It is a prerequisite for the task.</td>
</tr>
<tr>
<td>Outcome</td>
<td>The product of a task that satisfies the goal of that task or is an evidence to evaluate that goal.</td>
</tr>
<tr>
<td>Objective</td>
<td>Need satisfied by a task.</td>
</tr>
<tr>
<td>Tool</td>
<td>A resource used in a task that is neither the object nor the outcome of that task. It is a prerequisite for the task.</td>
</tr>
<tr>
<td>Community</td>
<td>Set of subjects that interact in the workflows of a MAS.</td>
</tr>
<tr>
<td>Rules</td>
<td>Set of norms that affect the behaviour of the MAS and do not belong to the division of labour.</td>
</tr>
<tr>
<td>Division of Labour</td>
<td>Organization of the subjects to carry out the workflows of the system. It includes the capabilities of the involved agents, the resources that they control, and their power relationships, among others.</td>
</tr>
<tr>
<td>Artifact</td>
<td>Whatever element use to model a MAS.</td>
</tr>
</tbody>
</table>

**Figure 1.** Basic elements of an activity system with UML-AT.

In addition to the concepts in Figure 1, UML-AT has been extended with other elements to ease reasoning over specifications in agent-oriented software processes. These additional elements comprehend, for instance, inheritance, connect, decompose, and contribution relationships. Inheritance allows expressing semantic hierarchies where
an element can be substituted by any of its descendants. Connect relation between activities expresses that the source activity of the relation produces artifacts (e.g. tools, subjects, or rules) that are needed for the target activity. The decompose relationship applies from artifacts to artifacts and indicates that the source artifact can be achieved or composed by the addition of the target artifacts. Finally, contribution relationships are inspired by i* (Yu, 1997) and show how artifacts influence each other.

The full specification of UML-AT is available at http://grasia.fdi.ucm.es/at.uml-at. Here, the remark in the introduction about the evolution of the language must be remembered. The current version of UML-AT is the result of the experiments already made. They cover methodologies like ADELFE (Bernon, Gleizes, Peyruqueou, and Picard, 2002), INGENIAS (Pavón, Gómez-Sanz, and Fuentes, 2005), and Tropos (Kolp, Giorgini, and Mylopoulos, 2006). These methodologies are characterized by their focus on hierarchies of goals and interactions considered as chains of purposeful high-level tasks, and by using mainly graphic languages for their models. The study of new methodologies will surely bring new concepts from Social Sciences to the language and fix the meaning of existing ones.

3. Mappings

Mappings contain the knowledge to perform the sharing of information between specifications. This sharing is based on bidirectional translations of structures between languages as section 4 explains. This section discusses the structures needed to represent these mappings sub-section 3.1 and how to build mappings in 3.2.

Structures for mappings

The following discussion about mappings is driven by Figure 2 and Figure 3. Figure 2 shows a UML description of the elements that mappings comprehend, and Figure 3 describes an example of the component of a mapping for the translation of a single relationship from UML-AT to INGENIAS (Pavón, Gómez-Sanz, and Fuentes, 2005).

A mapping is an account on how structures in a source language correspond to structures in a target language. That is, what networks of elements have to appear in the models of the target language as a consequence of the appearance of certain networks of elements in the models of the source language. According to Figure 2, the mapping is composed by several instances of a match, each one representing the translation between two sets of structures, the source and target patterns. The source patterns are described in the source language of the translation and the target patterns in the target language. The instances of source patterns represent the structures to find in the original specifications and the target patterns the structures to insert in the translation when the source patterns are found. Note that there can be several instances of source pattern and target pattern involved in one match. This serves to deal with translations among networks of concepts. Using several patterns helps to disambiguate elements to be translated since it gives a context for each involved element. An element in one diagram may be interpreted in several ways, but if we consider the same element together with a network of other elements, its meaning becomes more concrete. This is important in order to deal with the semantic differences among the source and target language.
Model Integration in Agent-Oriented Development

Figure 2. Structure of mappings expressed with UML class diagram.

Figure 3. Example of match from UML-AT to INGENIAS for a relationship. The upper model is the source pattern in UML-AT and the lower models their target patterns in INGENIAS.
The content of source patterns and target patterns are graphs. Graphs are described according to a meta-modeling language called GOPRR (Lyytinen and Rossi, 1999). It is assumed that any source or target specification can be represented in terms of GOPRR, which is true in most cases, though such representation may not exist at this moment. Having concrete GOPRR translations of each methodology is a technical problem outside of the scope of this paper. However, obtaining those translations seems to be feasible without losing information. The proof comes from the implementation of GOPRR in the tool MetaEdit+, which comes with a demonstration of how UML class diagrams can be expressed with GOPRR. Since most methodologies express their meta-models in UML class diagrams following MOF conventions, (see Bernon, Cossentino, and Pavón, 2005), those diagrams could be understood from the GOPRR point of view as well. In the examples and figures representing source and target patterns, elements will be depicted using diagram notation directly, instead of declaring what element in the diagram comes from what GOPRR element. This makes the patterns more readable and saves the readers from the low level details. Nevertheless, for the sake of understanding of such diagrams as patterns, it is necessary that readers learn the basics of GOPRR.

Summarizing GOPRR, graphs are groups of entities connected through roles with relationships, where all these elements can have related properties. An entity in a diagram appears as a box containing the name of the element. A relationship is drawn as a line connecting two entities. A role is the part of the relationship that connects with one entity. The properties associated to previous elements are the name of the element, its type, and the assigned value. The name is a container for the identifier of a concrete instance of the element, the type identifies the generic semantic class of the element, and the value represents the actual or default content of the property.

Properties can have fixed or variable values. Variables indicate that the actual value of that property will be instantiated in the translation. Variables are recognised because they are always labelled as the letter v plus a number, as seen in Figure 3. Its actual value is determined during the translation. The translation process looks for instances of the source patterns in the specifications to translate; when an instance is found, it grounds the variables with the actual information from the models, hence assigning a concrete value. Besides, values can be shared between the source and target patterns to forward information from the original specifications to the translation. Value sharing occurs when using the same variable name within the mapping scope in different patterns.

Figure 3 shows an example of match for translation. It declares that the UML-AT structure described by the source patterns, which is composed by a subject, an activity, and an objective and the depicted relationships, will appear in the INGENIAS models of its translation as the structures described by the target patterns. Figure 3 also shows that the values of the properties can be fixed text (e.g. the types of relationships) or variables (e.g. v0, v1, and v2). The name of the subject in the source pattern with UML-AT, that is the variable v1, is also the name of the agent in the agent model of the translation to INGENIAS.
Model Integration in Agent-Oriented Development

Mappings from/to UML-AT

Until this point, this section has introduced the data structures to represent mappings. Another issue, which is even more important, is how to conceptually conceive these correspondences with UML-AT. This task requires a good knowledge about the concepts of both AT and the specific agent-oriented methodology. As a help for this process, Table 1 already defined the concepts of the AT in terms of common abstractions of the agent paradigm with some assumptions that ease the automated detection of concepts. As explained in section 2, such assumptions are needed because of the different focus on AT conceptual framework, which is on activities, and the agent paradigm, which is on agents, and the detail of semantic information that would require an AT analysis as performed in Social Sciences. These definitions, along with previous experiences developing correspondences, allow providing some heuristics for the initial step of the conceptualizing process.

The concepts of activity, subject, and objective have restrictive definitions in Table 1 that can be fulfilled by few concepts in MAS. Therefore, these concepts are a good starting point to identify the elements of an activity system. In fact, the concept of activity is very close to that of high-level tasks in the agent-paradigm, in the sense that it is intended to satisfy a primary need of its subject; which corresponds to an AT objective. As opposed to high-level tasks, there are low-level tasks that enable the achievement of those high-level tasks but do not satisfy per se any relevant goal of the agent. This differentiation is equivalent to the existing in AT between activities, actions and operations (Leontiev, 1978). However, in most agent-oriented methodologies such differentiation between needs (that correspond to high-level tasks) and enabling goals (that correspond to low-level tasks) is blurred when considering the modelling primitives. For this reason, mappings usually makes equivalent AT activities and agent tasks, and the user must decide if a distinction between activities, actions, and operations makes sense for his problem. Subjects can be identified with those abstractions that behave according to the Rationality Principle (Newell, 1982). For the agent paradigm, these abstractions are usually agents, roles, and groups, when these are allowed to execute tasks. A community comprehends those subjects that participate in a common workflow or act over the same components of the MAS or its environment. Objectives represents the primary needs that agents try to satisfy. As it happens with activities, agent-oriented modelling primitives do not distinguish between high-level and enabling goals. For this reason, objectives are usually identified with agents’ goal.

As opposed to the previous elements of the activity system, the concepts of object, outcome, and tool are difficult to identify in a MAS. Their definitions are precise in AT (Engeström, 1987; Bednyi and Meister, 1997), but they can be identified with almost any element used to model a MAS depending on the context. For instance, in the case of the object and the tool, they are both elements used by an activity. Distinguishing what element is transformed into the outcome, i.e. the object, and what supports the transformation, i.e. the tool, requires knowing the semantics of the involved activity. Finding these AT concepts by a mere analysis of the specifications is far beyond the expressive power of most agent-oriented modelling languages. This kind of analysis is the only one that can perform automated tools. For this reason, in a first refinement of mappings, these concepts correspond to whatever element that can be used or generated by an activity.
The rules and the division of labour define behavioural patterns for the MAS. Many methodologies define these behaviours assigning roles. The difference between rules and the division of labour emerge from their relation with the object. The division of labour determines the organization of the community to transform the object, while rules are more generic, they emerge from the society and are not tied to the transformation process.

Finally, an activity system can be constituted by multiple interconnected activities described with different levels of detail. This abstraction must be mainly considered as a useful way of grouping concepts that are not relevant for the analysis in the current level of detail.

These heuristics and the resulting mappings are only the first step in the development of a set of fully functional correspondences between two languages. The introduction to AT (see Section 2) states that the minimal meaningful unit for analysis is the activity system. Thus, a correct interpretation of information demands the full context of the activity and understanding the involved elements. So, this initial set of mappings constitutes the basis to build more complex ones in a bottom-up manner. Considering this, the match in Figure 3 is only a first approach. A graph so reduced could translate a very simple concept that has almost the same meaning in UML-AT and INGENIAS. Richer concepts require more complex networks of elements for their description. For instance, mappings from UML-AT to INGENIAS currently include 28 matches with one source pattern each (UML-AT only has one type of diagram) and an average of 9 entities and 11 relationships per pattern. In general, matches that do not provide an enough detailed context can generate translations that are not accurate and introduce mistakes in the resulting specifications.

4. Integration Process

The integration process pursues the coordination of the different specifications a developer can handle in a development. The integration process would carry on according to the following scenario. A developer chooses a set of specification elements to be expanded with another methodology, the target methodology, what implies using another specification method, the target specification method. Chosen elements, those which are going to be shared, are translated to UML-AT using some predefined translation rules, and then translated again to the target specification method. It may be the case that there is no suitable combination of translation rules to and from UML-AT. This would indicate that the chosen elements are not adequate for sharing. The developer would be suggested to reconsider this set and to look for changing it. In the opposite situation, there may be more than one combination of valid translations. In that case, the developer would be informed of the possibilities and the effect of each translation. When the developer chose an alternative, the selected translation rules would be applied. On finishing the translations, there is a new set of instances of concepts belonging to the target specification method. Afterwards, the developer would work with the target methodology, expanding the initial set of concepts according to the guidelines provided by the target methodology. When finished, the developer would update the already shared concepts and the new ones created as a result of applying the target methodology. Already shared concepts update is fast since it is already known which mappings where applied to produce them. Therefore, it is easy to trace back those mappings to obtain the original concepts and perform the appropriate changes. Those elements not shared yet
Model Integration in Agent-Oriented Development

and obtained as a result of the work with the target methodology would be shared starting again with activity 1.

In the abovementioned scenario, the translation process (see Figure 5), is the basis for the integration that appears in Figure 4. In fact, activities 1 and 2 in Figure 4 are instances of the process in Figure 5. Activity 1 is the translation process with target language UML-AT and activity 2 is the translation process with source language UML-AT. Activity 1 in Figure 5 generates a set of new concept instances in UML-AT and activity 2 in the target specification method. Both create new entries in a Repository of Translations, which contains tuples indicating the matches (see section 3.1) and instantiation functions (see below in this section), used in the translation as well as the elements participating in it (either used in the process or created as a result of) and an identifier of the specification each one belongs to. Activity 3 closes the loop of the integration process incorporating modifications in the translated specifications to the original ones using mainly the information in the Repository of Translations. The case of new elements created after the translation as a result of working with the target methodology is different. Most of times, the user should want these new elements to be stored in the target specification without affecting the original or other specifications. In some cases, these new elements should be incorporated as new contributions to other specifications, i.e. as new shared elements. To do so, the integration process is applied again, but over the new elements in the target language. Therefore, the target becomes the source.

Figure 4. UML state diagram for the integration process.

How translation activities happen is described next. They follow the translation process in Figure 5, no matter if it starts from a specification produced by a methodology or from UML-AT. It makes no distinction since it bases on the application of mapping rules which map networks of concepts from a language to another. The translation process consists in four main steps. The first one is to select the mapping to apply (activity 1 in the figure). Then, for every match in the mapping, the process carries out three steps: to locate instances of the source patterns of the match in the specifications, which is activity 4; to generate the instantiation functions for the source patterns according to the matching, that corresponds to activity 6; to instantiate the target patterns
of the match (see section 3.1 for details about mappings structure) and to add them to the translation, which are respectively activities 7 and 8. Let describe with more details these processes beginning with the translation.

Figure 5. UML state diagram for the translation process with mappings.

The translation process begins with activity 1, which selects the mapping to apply over the MAS specifications. The choice is given by the language of the specifications, which is the source language, and the target of the translation. If more than one mapping is applicable, the user is consulted. After selecting one mapping alternative, the process carries out a cycle that applies all the matches in the mapping to the specifications and that is governed by condition 2 and task 3.
With a given selected match, activity 4 looks for groups of elements in the specifications of the MAS that have the same structure of the source patterns of the match. A sub-graph from the specifications matches with the source patterns if it has the same entities connected by the same relationships and the properties of these elements have compatible values. This comparison must consider equivalence relationships like, for instance, that an artifact from UML-AT can represent any UML-AT concept. When a matching happens, it describes an instantiation function that describes the correspondence of the variables in the source patterns with the elements in the actual specifications according to that matching. This function is the result of activity 6. The selected match along this instantiation function is stored in a Repository of Translations. The information in this repository allows in the overall integration process tracing and forwarding changes of the same information between its different views, as this section explains later.

Using the instantiation function obtained in activity 6, activity 7 generates an instance of the target patterns of the match where all its variables are substituted by concrete values from the specifications. These instantiated target patterns are the translation of the sub-graph from the specifications identified in task 4.

Finally, activity 8 adds the new information, which is the grounded target patterns, to the translated specifications.

The application of a given match ends when there are not remaining non-translated instances of its source patterns (choice 5 in Figure 5). The application of a mapping finishes when all its matches has been applied (choice 2 in Figure 5).

5. Case Study: Implementation of FIPA Protocols

This section illustrates the use of the integration framework with a case study about the implementation of FIPA protocols (http://www.fipa.org). In this example, these protocols are designed with the models belonging to AUML, which are different from those used in the development of the other aspects, those from INGENIAS.

AUML (Bauer and Odell, 2005) is a proposal to extend UML (OMG, 2005) in order to specify some aspects of MAS, especially interactions. This set of models has been extensively used for the specification of FIPA protocols. The AUML models in its current state (Bauer and Odell, 2005) are just intended for analysis and early design.

In this setting, a development team using AUML to specify its protocols would like to turn to INGENIAS (Pavón, Gómez-Sanz, and Fuentes, 2005) to develop the rest of MAS aspects. INGENIAS follows a MDA approach to the development of MAS. It provides tools that support the transformation of MAS specifications to source code for several target platforms. Specifically, the transformation approach of INGENIAS, implemented by the tool called the IDK, is based on the use of templates. These templates contain source code for the target platform with variables that have to be filled with elements from the specifications. The IDK processes the templates according to the MAS specifications and generates the source code for the target platform. Then, developers may refine the resulting code, if there is information that neither the specifications nor the templates contain.

As AUML and INGENIAS use different modelling primitives, the information described with each one cannot be directly represented with the other. Besides, there are not modelling tools that support an integrated specification that uses both languages. Following the approach presented in this paper, this problem can be addressed by using
UML-AT as an intermediate language for integration of results in the different methods and tools. The remaining section is decomposed in two sections. Section 5.1 discusses the mapping needed for this case study. Section 5.2 focuses on the application of the integration process for some specifications.

Mappings

The present case study requires the availability of two mappings: from AUML to UML-AT and from UML-AT to INGENIAS. As the building of both mappings is quite similar, the next discussion focuses on the mapping from UML-AT to INGENIAS. The interested reader can see some examples of matches from AUML to UML-AT in Table 2 and from UML-AT to INGENIAS in Table 3 and Table 4. The matches in Table 3 are concept to concept (belonging to the first iteration of a mapping development process) and the ones in Table 4 are more elaborated ones between graphs. In the tables, elements in italics are new ones introduced by the match in the translation. Because of differences in the meaning of primitives, some matches need to add further elements in the translated specifications to maintain the consistency in the target specifications. In order to keep the explanation about the mapping from UML-AT to INGENIAS simple, the following argumentation only considers correspondences from concept to concept (those in Table 3), instead of correspondences between complex graphs (like those in Table 4), which are more precise semantically (see section 3 for an explanation about this topic).

As it was explained in Section 3, the development of the mappings should begin with those concepts that have more restricted meanings, as they are easier to identify. These concepts are subject, activity, and objective.

According to the definition in Table 1, a subject can be represented by any concept of INGENIAS that follows the Rationality Principle (Newell, 1982). Agent and role satisfy this definition, as they represent actors that execute tasks trying to attain goals.

Table 2. Examples of mappings from AUML to UML-AT.

<table>
<thead>
<tr>
<th>Activity Model</th>
<th>activity 1 → [connect] → activity 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence Model</td>
<td>community(send_cfp 1) → [decompose] → subject(agent 1)</td>
</tr>
<tr>
<td></td>
<td>community(send_cfp 1) → [decompose] → subject(agent 2)</td>
</tr>
<tr>
<td></td>
<td>activity(cfp) → [collectively accomplished by] → community(send_cfp 1)</td>
</tr>
<tr>
<td></td>
<td>activity(cfp) → [accomplished by] → subject(agent 1)</td>
</tr>
<tr>
<td></td>
<td>activity(cfp) → [collectively accomplished by] → service</td>
</tr>
<tr>
<td></td>
<td>activity(cfp) → [produce] → outcome(agent 2)</td>
</tr>
<tr>
<td></td>
<td>activity(cfp) → [use] → tool(content)</td>
</tr>
<tr>
<td>Class Model</td>
<td>community(organization) → [decompose] → subject(role)</td>
</tr>
<tr>
<td></td>
<td>activity(service) → [accomplished by] → subject(role)</td>
</tr>
<tr>
<td></td>
<td>activity(service) → [collectively accomplished by] → organization</td>
</tr>
</tbody>
</table>
Table 3. Examples of mappings one-one between concepts from UML-AT to INGENIAS.

<table>
<thead>
<tr>
<th>UML-AT</th>
<th>INGENIAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activity System</td>
<td>Task, Workflow, Interaction</td>
</tr>
<tr>
<td>Subject</td>
<td>Agent, Role</td>
</tr>
<tr>
<td>Object</td>
<td>Resource, Application, Interaction, Interaction Unit, Fact, Belief, Mental State, Autonomous Entity Query</td>
</tr>
<tr>
<td>Outcome</td>
<td>Fact, Belief, Mental State</td>
</tr>
<tr>
<td>Objective</td>
<td>Objective, Mental State</td>
</tr>
<tr>
<td>Tool</td>
<td>Resource, Application, Task, Workflow, Interaction, Interaction Unit, Fact, Belief, Event, Mental State, Autonomous Entity Query</td>
</tr>
<tr>
<td>Community</td>
<td>Group, Organization</td>
</tr>
<tr>
<td>Rules</td>
<td>Mental State, Autonomous Entity Query, Interaction, Group, Fact, Belief</td>
</tr>
<tr>
<td>Division of Labour</td>
<td></td>
</tr>
</tbody>
</table>

Table 4. Examples of mappings many-many between graphs from UML-AT to INGENIAS.

| objective 1 $\rightarrow$ [decompose] $\rightarrow$ objective 2 | - Tasks and Goals Model  
goal(objective 1) $\rightarrow$ [GTDecomposes] $\rightarrow$ goal(objective 2) |
| activity 1 $\rightarrow$ [connect] $\rightarrow$ activity 2 | - Organization Model  
task(activity 1) $\rightarrow$ [WFConnects] $\rightarrow$ task(activity 2) |
| subject $\rightarrow$ [pursue] $\rightarrow$ objective | - Organization Model  
agent(subject) $\rightarrow$ [GTPursues] $\rightarrow$ goal(objective) |
| activity $\rightarrow$ [accomplished by] $\rightarrow$ subject $\rightarrow$ [pursue] $\rightarrow$ objective | - Agent Model  
agent(subject) $\rightarrow$ [WFResponsible] $\rightarrow$ task(activity)  
agent(subject) $\rightarrow$ [GTPursues] $\rightarrow$ goal(objective) |
| community $\rightarrow$ [decompose] $\rightarrow$ subject 1 $\rightarrow$ [decompose] $\rightarrow$ subject 2 | - Tasks and Goals Model  
task(activity) $\rightarrow$ [GTResolves] $\rightarrow$ goal(objective) |
| activity 1 $\rightarrow$ [connect] $\rightarrow$ activity 2 | - Interaction Model  
interaction 1 $\rightarrow$ [IHasSpec] $\rightarrow$ GRASIASpecification 1 |
| activity 1 $\rightarrow$ [accomplished by] $\rightarrow$ subject 1 $\rightarrow$ [transform] $\rightarrow$ object(subject 1) | - Interaction Model  
agent(subject 1) $\rightarrow$ [UInitiates] $\rightarrow$ InteractionUnit(activity 1) |
| activity 1 $\rightarrow$ [transform] $\rightarrow$ object(subject 1) $\rightarrow$ subject 2 $\rightarrow$ [accomplished by] $\rightarrow$ activity 2 $\rightarrow$ [transform] $\rightarrow$ object(subject 2) | - Interaction Model  
agent(subject 2) $\rightarrow$ [UCollaborates] $\rightarrow$ InteractionUnit(activity 1)  
InteractionUnit(activity 1) $\rightarrow$ [UIPrecedes] $\rightarrow$ InteractionUnit(activity 2) |

An objective from AT is some state of the world that an actor wants to become true. An objective can be represented by the concepts of INGENIAS goal and mental state. The goal is an atomic objective in the sense that it can only be decomposed in further goals. The mental state allows enumerating the mental entities (like facts, goals, or events...
from the environment) and their relationships whose existence satisfies the need. As it commonly happens in agent-oriented modelling languages, there is no distinction between high and low level goals, so there is a practical equivalence in this stage between objective and agents’ goals.

Activities are characterized as purposeful tasks performed by subjects. Table 3 contains three possible translations for the activity: task, interaction, and workflow. The concept of task in INGENIAS is an immediate translation for an activity; a task is performed by an agent to satisfy a goal. An interaction is a sequence of tasks, executed under certain conditions, that fulfills an overall goal and individual goals for the participants. The consideration of the workflow as an activity is a bit more subtle. A workflow is a set of tasks and their related elements (like facts or objectives) that can be executed by an agent. It just has a consumption relationship with goals. The language of INGENIAS allows workflows to include tasks without goals. If it was not interpreted that these tasks share the objectives of their workflows and that the consumption relation between workflows and goals is one of satisfaction, these tasks would not fulfill any goal. This last assumption violates the Rationality Principle, as agents would execute tasks without purpose. Therefore, it must be interpreted that tasks share the goals of their workflows and these satisfy the objectives that they consume. For this reason, workflows are also considered as a possible translation for activities. Pay attention to the fact that, given the modelling primitives in INGENIAS, it is not possible to distinguish between AT activities, actions, and operations without further context.

The concept of community can be translated by several concepts in INGENIAS. INGENIAS considers that the management of the complexity of MAS, both from the point of view of the system itself and its development, demands that agents are organized with higher order structures. For this purpose, INGENIAS explicitly introduces the concepts of groups and organizations. Besides, those actors that participate in the same workflows can be considered as belonging to an implicit community. Both types of communities comprehend actors (i.e. agents and roles) that interact to carry out some tasks, and share resources and objectives.

The concepts of rules and the division of labour describe patterns and restrictions to the behaviour of the subjects in the community. INGENIAS allow representing this information with mental states, autonomous entity queries, interactions, groups, facts, and beliefs.

The concepts of object, outcome, and tool are the most difficult to translate. They correspond to almost the same abstractions and the differences in a specific translation depend on the semantics of the involved activities. For instance, an INGENIAS Fact that is used by a task can be either a tool (the Fact is needed to generate the outcome but this is not the result of the transformation of the Fact), an object (the outcome is the result of the transformation of the Fact), or the outcome itself (the element that satisfies the goal of the activity is the Fact). However and despite the similarities, there are some differences in the potential translations of these concepts. A tool can be translated in INGENIAS by a resource, application, task, workflow, interaction, interaction unit, fact, belief, event, mental state, and autonomous entity query. Objects and products can be translated to almost the same concepts as a tool, but there are exceptions. These exceptions are task, workflow, interaction, interaction unit, and event, since they cannot be transformed or generated by an activity.

Finally, an activity system is a description of one or several activities and its context. It can be modelled in INGENIAS with tasks, workflows, and interactions. As it happens
with the communities, the translation can consider implicit activity systems as arbitrary groups of elements in a MAS specification.

These mappings are immediate correspondences of AT concepts with INGENIAS abstractions. As it was explained in Section 3, these are only the first building blocks to develop matches that are complete graphs. The final mappings should provide a context as specific as possible to improve the accuracy of their translations. Examples of these contextualised matches are those in Table 4, which are applied in the next section.

Integration process

The mappings of the previous section enable the integration process between specifications from AUML and INGENIAS. This section considers a sharing information setting where the specification of the FIPA ContractNet protocol is made with both AUML and INGENIAS. AUML provides the models to describe the participants (see Figure 6) and the sequence of messages (see Figure 7) and INGENIAS the models to specify the rationality of agents (see Figure 10) carrying out the protocol and the processes to implement it. The information about the agents and tasks for the INGENIAS models is taken directly from the specifications with AUML and the specifications of goals involved in the protocol is exported from INGENIAS models to AUML ones. Besides, once that INGENIAS specifications are complete, code is generated for the target platform of choice. To simplify the discussion, this example focuses of all the possible alternative paths on the one composed by the speech act cfp followed by refuse. Besides, as the translation process is the same independently of the involved languages, the discussion focuses of the step from UML-AT to INGENIAS, although brief examples are provided for other translations.

Figure 6. Agent initiator in the FIPA ContractNet protocol specified with a class model from AUML.
Figure 7. FIPA ContractNet protocol with a sequence diagram from AUML.

Figure 8. UML-AT description of the agent initiator specified in Figure 6.
As stated before, Figure 6 and Figure 7 are samples of the starting specifications of the protocol according to AUML. The integration process in Figure 4 begins with task 1 that translates from the original specification to the intermediate language UML-AT. In this case, the translation uses the mappings in Table 2. Figure 6 is the class diagram that specifies the agent initiator which translation to UML-AT is shown in Figure 8. In the same fashion, the sequence of messages composed by cfp and refuse appearing in the sequence diagram in Figure 7 is Figure 9. Now, let explain the translation from UML-AT to INGENIAS with more details, that is task 2 in Figure 4. This explanation follows the steps of the translation process in Figure 5.

Task 1 in the translation process is the selection of the mapping to apply. In this case the translation is from UML-AT to INGENIAS, so the selected set of matches is the one in Table 4. Choice 2 and task 3 represent the iteration over all the matches in the mapping. When a match is selected, task 4 looks for its source patterns (i.e. the ones on the left column of Table 4) over the specifications. For the fourth match in Table 4 related with relationships accomplished by-pursue, two correspondences are found. They belong to the communicative acts cfp and refuse and their related subjects and objectives. These networks have not been previously translated and task 6 generates the instantiation functions for that correspondences. The target pattern (i.e. the one on the right column of Table 4) of this correspondence proposes to insert groups of elements in the Agent model and the Tasks&Goals model of INGENIAS. Task 7 generates the models in Figure 10 using these target patterns and the instantiation functions from task 6. Finally, task 8 inserts these instantiated networks in the translated INGENIAS models. In the Agent model, the subject is inserted as an agent, the objective as a goal, and the activity as a task. These elements are linked by new relationships WFResponsible and GTPursues. For the Tasks&Goals model, a new relationship GTSatisfies is added to connect the goal with the task that represents the activity.
Figure 10. INGENIAS description of the sequence cfp-refuse in the FIPA ContractNet protocol.
Another relevant mapping for this translation is the last one in Table 4. It identifies an interaction between several subjects. The relevant data for it is that there are activities whose objects are other subjects. According to AT, this kind of activities can be messages exchanged between agents, as the interaction "changes" the receiver of the message. A new aspect in this mapping with respect to the previous one is the presence of the elements interaction 1 and GRASIASpecification 1. Section 3 explains that in order to keep the consistency of specifications, mappings need sometimes to introduce new elements in the translated specifications. Interactions in INGENIAS are declared and described in Interaction models: interaction 1 represents the new found interaction, while GRASIASpecification 1 is the new Interaction model added to specify interaction 1 in detail.

The other mappings from Table 4 applied in this translation are simpler than these ones and do not deserve further discussion. The final result of the process over the specifications from Figure 8 and Figure 9 can be seen in Figure 10. The result is split in different models of an INGENIAS specification.

A further step in the development of the specifications would be the modification of some elements in the INGENIAS specifications to be added in the A UML models. As stated in section 4, modifications in properties of translated elements are trivial because the Repository of Translations provides a complete account of their related original concepts. For instance, changing the name of the Initiator agent or the activities and translate them back to A UML models is trivial. However, new elements involve a complete translation process to the original language of these new elements. Besides, in the case that the new elements have ambiguous translations or do not admit a direct representation in UML-AT, user’s interaction is needed to customize the finally integrated concepts. For instance, INGENIAS can specify the condition that triggers an interaction unit as a fact that is a prerequisite for it. In UML-AT, that condition is a tool of the activity that translates the interaction unit. The repository informs that the activity is the translation of a message from A UML. The common translation for a tool of a message would be as its content, while the correct one in this case is as its guard condition.

6. Conclusions and Future Work

This paper presents a framework aimed to facilitate the integration of specifications from different agent-oriented approaches (i.e. methodologies, languages, or systems). This is in line with the method engineering approach as it would enable in practice that developers use at every moment the method that they consider best suited to work with each aspect of a MAS.

The integration framework comprehends several elements: an intermediate language called UML-AT; mappings that describe translation rules; and processes to use the previous elements. UML-AT is based on AT, a sociological framework that provides a conceptual set of primitives that is rich enough to represent the most usual intentional and social concepts present in agent-oriented modelling languages. Mappings describe sets of matches that are couples of graph instances with a structure in a source language that translates into another in a target language. These mappings contain the knowledge applied in the translation process that carries out the integration of models in the
approach. There are several advantages of using UML-AT as intermediate language and mappings for integration:

- As UML-AT is a neutral language, the approach is independent of any agent-oriented methodology.

- Mappings support integration as pattern matching between structures. This reduces the burden of understanding the integration process and allows focusing on the conceptual correspondences.

- UML-AT is extensible. New concepts can be introduced in the language by adapting its meta-model specification. This could be required in certain cases to add features from new considered agent-oriented modelling languages.

- The approach is easily scalable to new methodological approaches. Adding a new methodology just requires to develop the mapping between its associated modelling language and UML-AT.

- The integration framework encourages comparisons between methods. Mappings are an analytical tool to consider the differences between agent-oriented methodologies. In the same way, it can be also applied to integrate concepts from Social Sciences.

- There are already support tools. The translation process is automated by the Activity Theory Assistant (http://ingenias.sourceforge.net).

The main drawback of this process is related with the potential multiplicity of translations. Translations of groups of elements are not always unique, that is, several matches can be applicable to the same piece of information (or a sub-graph of it) in the original specifications. Sometimes this multiplicity cannot be avoided, especially when in the specification being considered a network of elements is subject to different possible interpretations. Current agent oriented specifications, as produced by most agent oriented software engineering methodologies, are not formal, hence they have some degree of ambiguity. This statement bases on the review of ten agent oriented methodologies as presented by Henderson-Sellers and Giorgini (2005), and in the intents of unifying different agent oriented methodologies (Beron, Cossentino, and Pavón 2005), where meta-models for different methodologies are presented. Assuming that there is an inherent ambiguity in the specifications, the decision taken in this work was to alleviate the problem by involving the user. It is the user who would choose the most proper translation at every moment. The number of mapping alternatives offered to the user varies on the situation but it is a number over ten. Reducing this number remains as future work, but our impression is that it could be reduced by using machine learning techniques to obtain the most adequate mapping in each situation.

7. Acknowledgements

This work has been funded by the Spanish Council for Science and Technology under grant TIN2005-08501-C03-01, the Direcccion General de Universidades e Investigación de la Consejeria de Educacion of the Comunidad de Madrid, and the Universidad Complutense de Madrid (Research Group 910494).
8. References


R. Fuentes-Fernández, J. J. Gómez-Sanz, and J. Pavón


