

Integrating Agent-Oriented Methodologies with UML-AT

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

Dept. Sistemas Informáticos y Programación

Universidad Complutense Madrid

c/ Profesor José García Santesmases s/n, 28040 Madrid, Spain

{ruben, jjgomez, jpavon}@sip.ucm.es

ABSTRACT

There are many methodological approaches for Agent-Oriented Software Engineering, each one focusing on some features of multi-agent systems, but leaving others underdefined. For this reason, it would be interesting to have the possibility of applying different methods, according to their suitability to each particular problem domain and system view. Here, a key issue is how to integrate the information resulting from different methods in a common specification. We propose the use of an intermediate language called UML-AT, which would enable bidirectional transformations between models in different languages. These transformations allow representing views of those models in the language of choice at every moment. UML-AT is a UML profile based on the Activity Theory (AT) framework, which includes the concepts to describe societies of actors that are both autonomous and intentional. The translation with UML-AT mappings makes the integration process independent of any given methodology. At the same time, these mappings can be a basis to study missing features in design languages, according to what is needed in the study of the intentional and social aspects in multi-agent systems and human organizations following AT. This integration architecture by means of UML-AT mappings is illustrated with a case study about a legacy tourism application. It uses a support tool that automates the translation.

Categories and Subject Descriptors

D.2.11 [Software Engineering]: Software Architectures – domain-specific architectures, languages, patterns.

General Terms

Documentation, Design, Standardization, Languages.

Keywords

Multi-Agent Systems, Method Engineering, Conceptual Mappings, Activity Theory, UML-AT, Tropos, INGENIAS.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

AAMAS'06, May 8-12, 2006, Hakodate, Hokkaido, Japan.

Copyright 2006 ACM 1-59593-303-4/06/0005...\$5.00.

1. INTRODUCTION

Multi-Agent Systems (MAS) are used in heterogeneous application domains. Generally, a MAS is considered as a group of intentional autonomous entities (agents) that interact to satisfy some goals. Despite of this, distinct needs generate differences in methodologies about what the relevant features of MAS are and how to analyze them. Agent-Oriented Software Engineering (AOSE) includes methodologies that, for instance, focus on goals (e.g. KAOS [4]), the mental state of agents (e.g. AAIL/BDI [9]), expert systems (e.g. MAS-CommonKADS [8]), recursive interconnected tasks (e.g. DESIRE [2]) or interactions (e.g. INGENIAS [15]). This specificity makes sense to explore concrete aspects of MAS. However, the development of real systems needs to gather information about several of these aspects and, frequently, from different points of view (e.g. customers, managers, analysts, or deployers). Agent-oriented methodologies facilitate the study of some features on which they are focused, whereas they give little support in the analysis of other features that they consider as less relevant.

In the absence of a general modelling language for MAS, developers can cope with the previous problem using models from different methodologies. For every view of their MAS, they could choose the languages, methodologies, and tools that are best-suited, according to their own requirements and experience. This approach has the problem of how to integrate the information from different sources: the information of a view is gathered with some models, but it influences/appears on other views, which may be specified using different models. For instance, developers can use KAOS [4] to elicit the goals and models from DESIRE [2] to specify the task decomposition; in this setting, goals from KAOS should be available in DESIRE models, without duplicating information or extra specifying effort. This idea is analogous to the method engineering approach (as in [1]), which defines a generic metamodel that is a representational infrastructure to unify existing methodologies into a single specification. Although in [1] the authors illustrate its application with two agent models, this is done in an informal way and no tool exists yet to work with it automatically. We similarly consider the definition of such a generic language, which is UML-AT. Nevertheless, instead of deriving it conceptually, our proposal is the result of experience in its use for validation of intentional and social properties of MAS [5]. Moreover, we offer tool support with the *INGENIAS Development Kit* (IDK) [15], which serves to facilitate its application and to validate the specification of UML-AT.

UML-AT [5] is defined as an UML profile that includes the concepts of the Activity Theory (AT) [16]. AT provides a

conceptual and analytical framework to understand motivations, individual-society relationships, and networks of interacting activity systems embedded in organizations. Considering AT as the foundation for this intermediate language comes from the fact that it shares with the agent-oriented approach the view of systems as organizations of intentional entities. Besides, there are precedents of its application to AOSE, for instance, in requirements engineering [7], the analysis of contradictions [5], and the design of agent communication infrastructure [14]. These features of AT are shared with other theories from Social Sciences that has been used in agent-oriented research. Some relevant examples are the Reciprocal Role Taking by G. H. Mead, the Actor-Network Theory by Callon & Latour, the foundation of the BDI by M. E. Bratman, or the Biological Theory of Knowledge by H. Maturana (see [11] for an overview of theories from Social Sciences in Distributed Artificial Intelligence). The main difference is that AT is a whole trend of social research, which has a holistic approach covering from the social movements to the individual cognition in the frame of historical development. This allows the study of MAS and their human context from different points of view using an integrating theoretical foundation.

The use of UML-AT as a *lingua franca* for the translation has several advantages in AOSE. First, the translation process becomes independent of any specific methodology. The process remains unchanged and UML-AT evolves to satisfy the map requirements of the agent-oriented methodologies. Second, it reduces the number of mappings required for a methodology to those from and to UML-AT. The set of underlying AT concepts is rich enough to support most of the intentional and social concepts considered in AOSE. In fact, AT analysis of these features in human groups is usually more sophisticated than the current studies in agent-oriented methodologies. So, the third advantage is that UML-AT is a test-bed to detect missing concepts for these features in design languages, given that human organizations are a natural metaphor for MAS. This kind of studies improves the methodologies and also helps to choose the proper modelling language for specific views of a MAS, as these studies evaluate its expressive power.

The rest of this paper is organized as follows. Section 2 briefly introduces AT and the UML-AT language. Section 3 describes the mappings and the process to specify them. Section 4 shows the translation process that uses those mappings while Section 5 describes the support tool that implements them. This process and tools are applied to a case study about the use of Tropos and INGENIAS methodologies for the development of a MAS for cultural information, in Section 6. Finally, Section 7 discusses some final issues about the integration framework.

2. THE ACTIVITY THEORY

Activity Theory (AT) [16] is an analytical framework from Social Sciences intended to study human groups through their contextualised practices. AT considers people embedded in a context that is temporal, cultural, and social. People's practices and their context reciprocally influence each other and this interaction is called *activity*.

The *activity* is the fundamental unit of analysis in AT. An *activity* is a transformation process driven by people's needs. The *activity* satisfies these needs with the *outcome* obtained

from the transmutation of an *object*. The active agent that carries out the *activity* is known as the *subject*. Those *subjects* who share an *object* constitute a *community*.

The principle of mediation plays a central role in the AT. An *activity* always contains various artefacts (e.g. instruments, signs, procedures, machines, materials, laws, forms of work, or organisations). These artefacts have a mediating role: relations between elements of an *activity* are not directed, but mediated. *Tools* shape the way human beings interact with their context. A *tool* can be anything used in the transformation process of the *object*, including both material tools and tools for thinking. The relationship between *subject* and *community* is mediated by *rules* (e.g. laws, social conventions, or norms), and the relationship between *object* and *community* is mediated by the *division of labour*, which establishes how the *activity* is distributed among the members of the *community*. The *division of labour* considers the role each individual in the *community* plays in the *activity*, the power each wields, and the tasks each is held responsible for.

These concepts constitute the common core of the vocabulary for AT as agreed in its research community. To complete the vocabulary of the AT for MAS, some other entities (e.g. *activity systems*, *artifacts*, or *roles*) and relationships (e.g. mediations, contributions, or decompositions) were made explicit (since they already appear in some AT studies) or added to the previous core. An *activity system* is the context of an *activity*, and includes all the elements that appear in the AT core. An *artifact* is a generalization that can represent any concept of an *activity system*.

The set of AT concepts for MAS were translated into UML using profiles [13] to facilitate its use in AOSE processes. The resulting language is called UML-AT. The representation of an *activity system* in UML-AT, plus the *artifact* concept and mediation relationships, appears in Figure 1. The full specification of UML-AT can be found at <http://grasia.fdi.ucm.es/at/uml-at>.

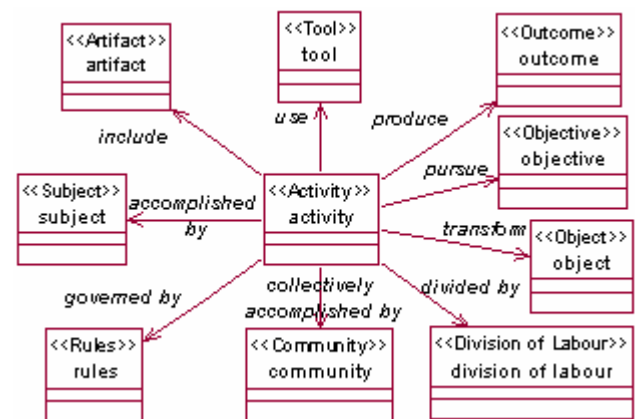


Figure 1. Activity system with UML-AT.

In its current version, UML-AT supports the most widespread concepts used in AOSE to describe intentional and social features of MAS. This allows representing the concepts of a given agent-oriented methodology as UML-AT diagrams and,

thus, translating MAS specifications obtained with that methodology to UML-AT. In the case of concepts from agent-oriented methods that cannot be mapped as UML-AT patterns, they will lead to the addition of new AT concepts to UML-AT.

3. MAPPINGS WITH UML-AT

Mappings contain the information required to transform the specifications of a MAS between UML-AT and the language of an agent-oriented method. A mapping is a correspondence from structures in a source language to structures in a target language, where both patterns can contain constant and variable values. These structures exchange information through shared variables. The following discussion about mappings is illustrated with the example in Figure 2. It shows a mapping from a relationship defined with the INGENIAS methodology [15] to UML-AT. This is an excerpt of the XML files for mappings that use the tool that is later introduced in Section 5.

Figure 2 shows the main elements required to define mappings. The tags *language_source* and *language_target* indicate the languages of the source specifications (e.g. INGENIAS) and the

translated ones (e.g. UML-AT). A *match* describes a correspondence, in a given translation level, between two structures: the *source pattern*, which is enclosed within the *source* tags, and the *target pattern*, which is surrounded by the *target* tags. The translation level gives a sorting for the application of the mappings, as the Section 4 explains. The first translation cycle use the mappings of level 0; every subsequent cycle increases the level by 1. Both INGENIAS and UML-AT work with information represented as graphs. Their languages are specified with a meta-modelling language called GOPRR. GOPRR [10] describes *graphs* in terms of *relationships*, whose extremes are *roles* that connect with *entities*. All of these elements can have related *properties*. The elements within *source* and *target* tags reflect these primitives. A *relationship GTFailsource* in an INGENIAS view is translated to a *relationship AT_Contribute* in an UML-AT diagram, which has the same name as the *GTFailsource* (shared variable “v2”) and a value *Avoided* of the attribute *contribution*.

<pre> <correspondences xmlns:xsi= "http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="Mapping.xsd"> <language_source name="INGENIAS"/> <language_target name="UML_AT"/> <match> <level value="1"/> <source> <entity> <name> <v> v0 </v> </name> <type> Fact </type> </entity> ... <relationship> <name> <v> v2 </v> </name> <type> GTFails </type> <role> <name> <v> v3 </v> </name> <type> GTFailssource </type> <attribute> <name> origin </name> <value> true </value> </attribute> <attribute> <name> player </name> <value> <v> v0 </v> </value> </attribute> </role> ... </relationship> </source> </pre>	<pre> <target> <entity> <name> <v> v0 </v> </name> <type> AT_Object </type> </entity> ... <relationship> <name> <v> v2 </v> </name> <type> AT_Contribute </type> <attribute> <name> Contribution </name> <value> Avoided </value> </attribute> <role> <name> <v> v3 </v> </name> <type> AT_Contribute_Source </type> <attribute> <name> origin </name> <value> true </value> </attribute> <attribute> <name> player </name> <value> <v> v0 </v> </value> </attribute> </role> ... </relationship> </target> </match> </correspondences> </pre>
--	--

Figure 2 Excerpt of mapping from INGENIAS to UML-AT.

The example in Figure 2 corresponds to a simple correspondence. Real mappings usually relate graphs. The reason for that is that the meaning of the elements in the specifications strongly depends on their relationships with others. Very simple mappings can generate translations that are not completely accurate and introduce mistakes in the resulting specifications.

The main difficulty when developing these mappings is to determine what the correct correspondence is for a given *source pattern*. This task demands a deep knowledge of the concepts involved in UML-AT and the agent-oriented methodology, but it has only to be done once for a given methodology. There are some available heuristics to carry out this process. These are based on the description of AT concepts from the point of view of the agent paradigm that Table 1 summarizes.

Table 1. AT concepts for the agent paradigm.

AT concepts	Definition with the agent paradigm
Activity System	Set of modelling elements representing a total or partial view of the MAS.
Activity	Task that pursues some relevant objective for an agent or group of agents in the MAS.
Subject	Agent, role, or organization. In general, whatever active element that behaves according to the Rationality Principle ¹ can be modelled as a <i>subject</i> .
Object	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.
Outcome	The product of a task that satisfies the goal of that task or is an evidence to evaluate that goal.
Objective	Need satisfied by a task.
Tool	A resource used in a task that is neither the <i>object</i> nor the <i>outcome</i> of that task. It is a prerequisite for the task.
Community	Set of <i>subjects</i> that interact in the workflows of a MAS.
Rules	Set of norms that affect the behaviour of the MAS and do not belong to the <i>division of labour</i> .
Division of Labour	Organization of the <i>subjects</i> 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.
Artifact	Whatever element use to model a MAS.

The definitions of Table 1 give some hints to develop mappings in agent-oriented methodologies. The concepts of *activity*, *subject*, and *objective* have restrictive definitions that can be fulfilled by few concepts in a MAS. So, these concepts are a

¹ The Rationality Principle states that “Every action taken by an agent aims at achieving one of its goals” [12]

good starting point to identify the elements of an *activity system*. A *community* can only be constituted by those *subjects* that participate in a common workflow. On the contrary, the concepts of *object*, *outcome*, and *tool* are difficult to identify in a MAS. Their definitions are precise but almost any element used to model a MAS can be them depending on the context. Besides, in the case of the *object* and the *tool*, it could be necessary to know the semantic of the involved *activity* to distinguish what element is transformed into the *outcome*, i.e. the *object*, and what supports the transformation, i.e. the *tool*. The *rules* and the *division of labour* define behavioural patterns for the MAS. Many methodologies define these behaviours assigning roles. The differences between both groups of patterns emerge from the *object*. The *division of labour* determines the organization of the *community* to transform the *object*, while *rules* are more generic. 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.

4. TRANSLATION PROCESS

The translation process can be seen in Figure 3. This process consists of two main activities: the management of the translation levels and the application of individual mappings.

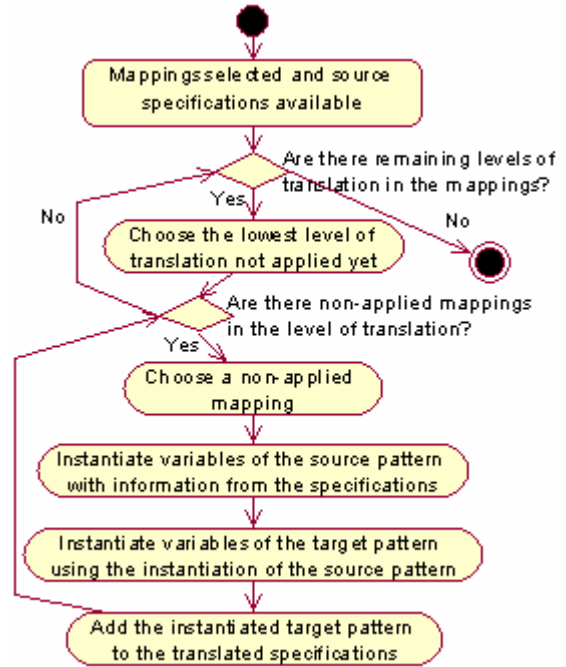


Figure 3. Translation process with mappings.

A translation level groups a set of mappings and has a sorting number. Translation levels are applied according to their sorting number from lowest to highest. Mappings in a translation level are applied sequentially, without any particular order, and can use all the concepts previously translated by other levels. These prerequisites of other concepts for the translation determine the sorting of the levels.

The translation process with the mappings in a specific level consists basically of three steps:

1. Identify the pattern of the source language, i.e. the *match pattern*, over the specifications and instantiate its variables with that information. The process begins looking for groups of elements with the same structure of the *match pattern* in the specifications. When identification happens, the process has to check that the values of the properties in both the pattern and the specification are compatible. If they are compatible, the identification creates a substitution function from the variables in the *match pattern* to the corresponding elements in the specifications.
2. Instantiate the pattern of the target language, i.e. the *target pattern*, with the substitution function obtained from the identification of the *match pattern*.
3. Add the instantiated *target pattern* to the translated specifications. This is the translation of the part of the original specifications that corresponds with the *source pattern* of the mapping.

A remark has to be done about the translation process. Translations between groups of concepts are not always one to one. A given group in the source language may have several corresponding groups in the target language. These groups can correspond to several views (i.e. instances of models) of the original information in the target language; but sometimes they are different possible interpretations of the same information. This ambiguity is inherent to the fact that the modelling primitives and their meaning are different in the source and target languages. However, the negative effects of this potential multiplicity of translations are minimized for the singularity of the original information. That is to say that the multiple interpretations are only information to work in the tool, but they do not exist as part of the specifications, where only the origin of the translation exists.

5. TRANSLATION TOOL

The application of the translation process in a development is automated by a tool called the *Activity Theory Assistant* (ATA). The general purpose of the ATA is to help MAS developers with AT-based techniques. Part of this support is the translation process presented in this paper. Other techniques implemented by the ATA include the support to elicit requirements [7] or to detect AT contradictions in the specifications [5]. The modules required for the translation can be seen in Figure 4.

The knowledge of the assistant about the languages and the correspondences involved in translations comes from XML files, as those seen in Section 3. The information concerning the languages allows the ATA to validate the specifications, checking that their contents are according to the definition of the language. Correspondences specify the translations themselves.

The *graph* package gives an implementation of the primitives of GOPRR [10]. Its components are instantiated for the internal representation of information in the ATA about languages, correspondences, and MAS specifications.

The *wrappers* package provides uniform APIs for the sources of MAS specifications. The *targets* package includes wrappers for the resources related to the input/output about the translation.

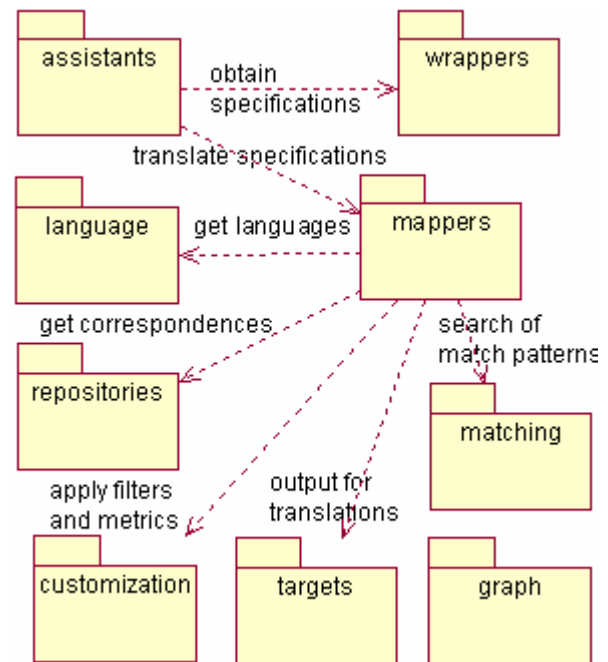


Figure 4. ATA architecture for translation.

Language gives the functionality to deal with the definition of the vocabularies, while the *repositories* package does it for the correspondences. They transform the XML descriptions to the internal structures of the ATA defined by *graph*.

Matching deals with the identification of *source patterns* in the specifications. Its results are used by the *mappers* package to translate specifications according to correspondences.

The *customization* package includes functionality to evaluate the tool. It generates statistics about the use of mappings, generated translations, or elements that the ATA could not translate.

The *assistants* module implements the functionality to coordinate all the previous packages. The core of the ATA belongs to this package.

The integration of the ATA in a specific agent-oriented development environment requires writing the corresponding components for the modules *wrappers* and *targets*. The component for *targets* is optional, since the ATA has default resources for output. As a proof of concept, the ATA is currently embedded as a plug-in of the *INGENIAS Development Kit* (IDK) (<http://ingenias.sourceforge.net>), which is the modelling and development tool of the INGENIAS methodology [15].

6. CASE STUDY: eCULTURE

To illustrate the integration architecture, this section shows a case study based on the *eCulture* system [3]. *eCulture* is a web-based broker of cultural information developed with the Tropos methodology. Tropos [3] describes a whole development process, from early requirements to implementation. This process does not fully address some issues, like the transition from the detailed design to the implementation. This transition is largely a manual process depending on developers' skills to map

concepts from Tropos specifications to the target platform. Even worse, as the process is manual, modifications in the original specifications have to be traced and updated to the final code, with the consequent negative impact on the maintainability.

Given the previous situation for Tropos, developers would prefer using the INGENIAS approach for this transition. INGENIAS [15] mainly bases the coding on templates and the IDK. The templates contain 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 specifications and generates the code for the target platform. Of course, developers have to refine the resulting code, as there is information that neither the specifications nor the templates contain. However, most of the work caused by changes in the specifications is performed by the IDK.

The problem for the previous approach is that the languages of Tropos and INGENIAS are different. Despite the availability of support tools (OME for Tropos and IDK for INGENIAS), none of them can interoperate with the other. A possible solution is the interconnection through the ATA. In order to perform the integration, the ATA needs two mappings: from Tropos to UML-AT and from UML-AT to INGENIAS. The mappings for INGENIAS and Tropos can be found at [6]. A summary of some mappings with INGENIAS appears in Table II. There is a discussion about how to build some of these mappings in the next paragraphs. To keep the explanation simple, this argumentation only considers correspondences from concept to concept, instead of correspondences between graphs, which are more precise semantically (see Section 3 for an explanation about this topic).

The development of the mappings begins with those concepts of the AT that are more restricted. These concepts are *subject*, *activity*, and *objective*.

A *subject* is any element of a MAS that behaves according to the Rationality Principle [12]. INGENIAS has two abstractions that satisfy this definition, *agent* and *role*. Both of them have related goals and tasks to accomplish them.

An *activity* is a practice that satisfies some goal for a *subject*. The concept of *task* in INGENIAS satisfies this definition.

Another possible *activity* in INGENIAS is the *workflow*, which comprehends a set of *tasks* and their related elements like *facts* and *applications*. A *workflow* does not have a satisfaction relationship with goals, but it has a consumption relationship with them. This consumption has to be interpreted as a relation of the *workflows* with the goals that they globally pursue. The reason is that INGENIAS allows including *tasks* without goals. If these *tasks* did not share the goals of their *workflows*, they would not satisfy any purpose, which is impossible according to the Rationality Principle. Given this interpretation, *workflows* are another kind of *activities*. An *interaction* is also an *activity*. It is defined as a sorted set of *tasks* that is executed pursuing a goal, either individual for the participants or global for a group.

An AT *objective* represents a state of the world that agents want to be true. INGENIAS has its own concept of *objective* for these states. Besides, they can also be graphically represented as *mental states* of agents. So, *mental states* are AT *objectives* too.

The concept of *community* is explicitly supported in INGENIAS with *groups* and *organizations*. These *communities* include

agents and *roles*, which share resources, and the *goals* that they pursue. These *agents* and *roles* usually participate in the same *workflows*. INGENIAS also has implicit *communities*, which are composed by *subjects* participating in the same *workflow* but that do not belong to a common *group* or *organization*.

Table II. Mappings between AT and INGENIAS.

AT	INGENIAS
Activity System	Task, Workflow, Interaction
Activity	
Subject	Agent, Role
Object	Resource, Application, Interaction, Interaction Unit, Fact, Belief, Mental State, Autonomous Entity Query
Outcome	
Objective	Objective, Mental State
Tool	Resource, Application, Task, Workflow, Interaction, Interaction Unit, Fact, Belief, Event, Mental State, Autonomous Entity Query
Community	Group, Organization
Rules	Mental State, Autonomous Entity Query, Interaction, Group, Fact, Belief
Division of Labour	
subject 1 → [decompose] → subject 2	- Organization Model group(subject 1) → [OHasMember] → agent(subject 2)
activity → [accomplished by] → subject	- Agent Model agent(subject) → [WFResponsible] → task(activity)
activity → [transform] → object	- Tasks and Goals Model task(activity) → [GTAffects] → fact(object)
object → [contribute positively] → subject	- Agent Model agent(subject) → [GTPursues] → goal ¹² agent(subject) → [WFResponsible] → task 1 - Tasks and Goals Model task 1 → [GTSatisfies] → goal 1 task 1 → [GTAffects] → fact(object)

The concepts of *object*, *outcome*, and *tool* are conceptually very close. Any element used by an *activity* is a *tool*. According to the previous translations for *activity*, a *tool* from INGENIAS can be 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*. The exceptions are *task*, *workflow*, *interaction*, *interaction unit*, and *event*, as they cannot be transformed or generated by an *activity*.

The *rules* and the *division of labour* can be specified in INGENIAS with *mental states*, *autonomous entity queries*, *interactions*, *groups*, *facts*, and *beliefs*. All of them describe

² Elements in italics are new ones introduced by the mappings.

behavioural patterns for the agents and are therefore suitable to establish the norms of the system.

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.

This discussion is just an example of the kind of reasoning needed to develop the mappings. More complex patterns (as those about relationships in Table II) would improve the semantic accuracy of the resulting translations.

Once having the mappings, the ATA can perform the translation. For this case study, the actor of Tropos in Figure 5 is translated to INGENIAS. The translation from UML-AT to INGENIAS is going to be discussed with more details; for Tropos, Figure 6 shows the translation to UML-AT using the mappings in [6].

The translation of Figure 6 to UML-AT is, according to mappings for relationships in Table II, as follows. The *match pattern* (i.e. the one on the left column of Table II) is searched over the specifications. For the *decompose* relationship, three matches are found, those corresponding to *Info Broker* with *Results Synthesizer*, *Info Searcher*, and *Area Classifier*. The

solution pattern (i.e. the one on the right column of Table II) proposes to insert as translations for the *decompose* relations new *OHasMember* relations in the Organization Model. These relations connect the *Info Broker* as a *group* with the other *subjects* as INGENIAS *agents*.

The translation of the relations *accomplished by* and *transform* in Figure 6 is very similar to the one for *decompose*. It uses the second mapping for relationships in Table II.

The mapping for *contribute positively* deserves some remarks. Source and target languages of mappings usually have different semantics. For this reason, it is not always possible to find a direct translation between concepts of both languages. In the case of the *contribute* relations in AT, they can connect *artifacts* (i.e. whatever modelling element of AT), what cannot be represented in INGENIAS. A similar semantic can be obtained with the introduction of two new entities (i.e. *task 1* and *goal 1*) in the solution pattern. These entities are not saved in the specifications; they are just temporarily introduced for the modelling activity.

The result of the application of these three mappings for relationships over the specifications from Figure 6 can be seen in Figure 7. The result is split in different models of an INGENIAS specification.

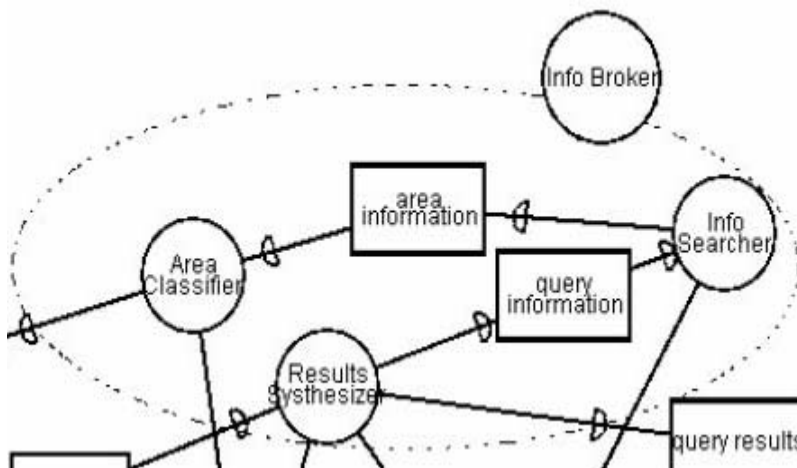


Figure 5 An actor and its components with Tropos.

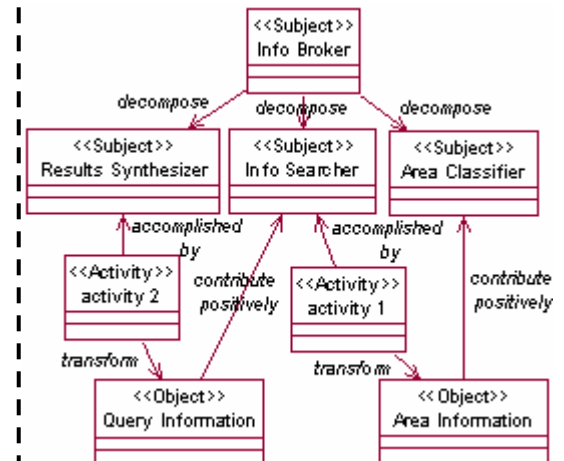


Figure 6 Translation of Figure 5 to UML-AT.

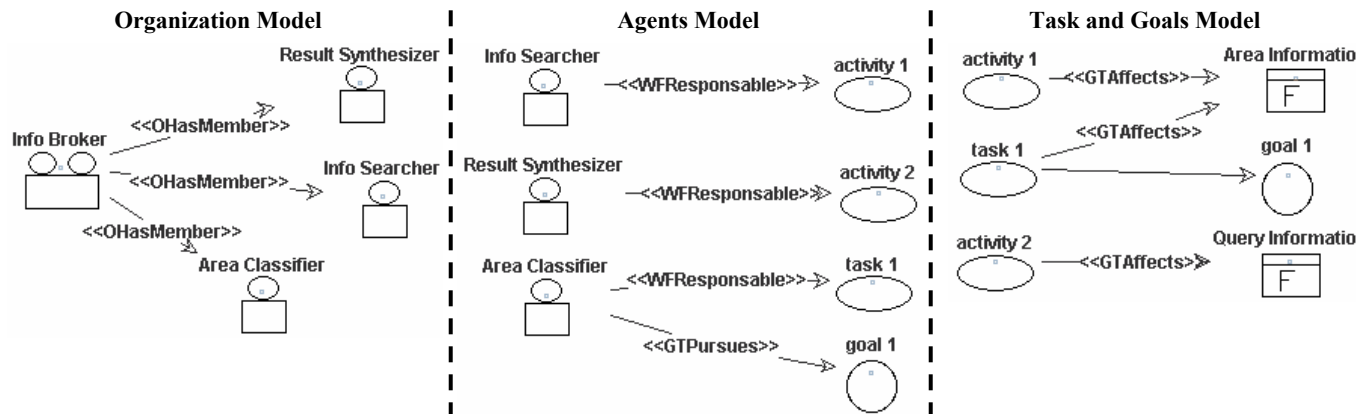


Figure 7. Translation of Figure 6 to INGENIAS.

7. CONCLUSIONS AND FUTURE WORK

This paper presents a framework for the integration of different agent-oriented methods in MAS development. It allows the use of the best-suited method to specify every aspect of a MAS.

The integration is based on a translation process with mappings. A mapping indicates how a structure in a source language can be represented by a structure in another language with the same meaning, or at least a close one. Mappings use as an intermediate language UML-AT. This language grounded in AT is rich enough to support most intentional and social concepts present in agent-oriented modelling languages.

The main advantages of the proposed integration architecture over others are: 1) it is independent of any agent-oriented methodology, as it is based on the neutral language UML-AT; 2) the pattern matching process gives simple explanations about the correspondences applied in the translation; 3) the addition of new methodologies just requires the specification of their mappings; 4) mappings with UML-AT are a suitable resource to discover and analyze missing features in agent-oriented languages; 5) the architecture has automated support tools.

The specification of the mappings is the hardest task of those related with the application of the architecture. It demands a deep analysis of the semantics of the involved concepts. Besides, the possibility of introducing new elements in the *solution pattern* deserves a careful consideration as it should avoid the appearance of unnecessary additional information.

Future work includes two main lines of research. The first one is the creation of mappings for new methodologies, which will enrich UML-AT with new concepts. The second one is the study of agent-oriented methodologies from the AT perspective. This study aims to improve these methods and their integrated use for development, and to get a better understanding of the foundations of the agent paradigm in Social Sciences.

8. ACKNOWLEDGMENTS

This work has been funded by the Spanish Council for Science and Technology under grants TIC2002-04516-C03-03 and TIN2005-08501-C03-01.

9. REFERENCES

- [1] Beydoun, G., Gonzalez-Perez, C., Low, G., and Henderson-Sellers, B. Synthesis of a Generic MAS Metamodel. In *Proc. 4th Int. Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS'05)* (St. Louis, MO, USA, May 15–16, 2005).
- [2] Brazier, F. M. T., Dunin-Keplicz, B. M., Jennings, N. R., and Treur, J. DESIRE: Modelling Multi-Agent Systems in a Compositional Formal Framework. *International Journal of Cooperative Information Systems* 6, 1 (1997), 67–94.
- [3] Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., and Perini, A. A Knowledge Level Software Engineering Methodology for Agent Oriented Programming. In *Proc. 5th Int. Conf. on Autonomous Agents (AGENTS'01)*, ACM Press, 2001, 648–655.
- [4] Dardenne, A., van Lamsweerde, A., and Fickas, S. Goal-directed Requirements Acquisition. *Science of Computer Programming*, 20 (1993), 3–50.
- [5] Fuentes, R., Gómez-Sanz, J. J., Pavón, J. Activity Theory for the Analysis and Design of Multi-Agent Systems. *Agent-Oriented Software Engineering IV, 4th International Workshop on Agent-Oriented Software Engineering (AOSE-2003), Lecture Notes in Computer Science*, 2935, Springer-Verlag, 2003, 110–122.
- [6] Fuentes-Fernández, R. *Activity Theory for the Development of Multi-agent Systems* (in Spanish: *Teoría de Actividad para el Desarrollo de Sistemas Multi-Agente*). Ph. D. Thesis, Universidad Complutense, Madrid, Spain, 2004.
- [7] Fuentes-Fernández, R., Gómez-Sanz, J. J., Pavón, J. Requirements Elicitation for Agent-Based Applications. In *Proc. 6th Int. Workshop on Agent-Oriented Software Engineering (AOSE-2005)* (Utrecht, The Netherlands, July 25–26, 2005).
- [8] Iglesias, C. A., Garijo, M., Gonzalez, J. C., and Velasco, J. R. Analysis and design of multiagent systems using MAS-CommonKADS. *Intelligent Agents IV, Lecture Notes in Artificial Intelligence*, 1365, Springer Verlag, 1998, 313–326.
- [9] Kinny, D., Georgeff, M., Rao, A. A methodology and modelling technique for systems of BDI agents. *Agents Breaking Away: Proceedings of the 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-96), Lecture Notes in Artificial Intelligence*, 1038, Springer-Verlag, 1996, 56–71.
- [10] Lyytinen, K. S., and Rossi, M. METAEDIT+ – A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment. *Advances Information System Engineering, 8th International Conference (CAiSE'96), Lecture Notes in Computer Science*, 1080, Springer-Verlag, 1999, 1–21.
- [11] Malsch, T. Naming the Unnamable: Socionics or the Sociological Turn of/to Distributed Artificial Intelligence. *Autonomous Agents and Multi-Agent Systems* 4 (3), 155–186. 2001.
- [12] Newell, A. The knowledge level. *Artificial Intelligence*, 18, 87–127, 1982.
- [13] OMG. Unified Modelling Language Specification. Version 1.5 [Online]. Available: <http://www.omg.org>. 2003.
- [14] Omicini, A., Ricci, A., Viroli, M., Castelfranchi, C., and Tummolini, L. Coordination artifacts: Environment-based coordination for intelligent agents. In *Proc. 3rd Int. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS-2004)* (New York, USA, July 19–23, 2004), ACM Press, vol. 1, 286–293.
- [15] Pavón J., Gómez-Sanz J., and Fuentes, R. The INGENIAS Methodology and Tools. In Henderson-Sellers, B., and Giorgini, P., editors: *Agent-Oriented Methodologies*. Idea Group Publishing, chapter IX, pp. 236–276, 2005.
- [16] Vygotsky, L. S. *Mind and Society*. Harvard University Press, Cambridge, MA, USA, 1978.