INGENIAS Development Kit (IDK): Tutorial and Manual

Facultad de Informática
Universidad Complutense de Madrid
C/ Juan del Rosal 8
28040 Madrid
España

http://grasia.fdi.ucm.es

TUTORIAL AND MANUAL

SUMMARY

THIS DOCUMENT PRESENTS THE INGENIAS DEVELOPMENT KIT (IDK), A SET OF TOOLS TO SUPPORT THE SPECIFICATION AND IMPLEMENTATION OF MULTI-AGENT SYSTEMS. IT IS ORIENTED TO RESEARCHERS LOOKING FOR A SPECIFICATION TOOL AND DEVELOPERS WANTING TO DOCUMENT AN AGENT ORIENTED DEVELOPMENT.

AS IT IS DISTRIBUTED, THE IDK CAN BE DIRECTLY USED TO SPECIFY, USING A GRAPHICAL EDITOR, MULTI-AGENT SYSTEMS. THERE ARE ALSO SOME MODULES TO PERFORM AUTOMATIC CODE GENERATION AND VALIDATION & VERIFICATION OF THE SPECIFICATIONS. AN ADDITIONAL FEATURE FOR THE DEVELOPER IS THE POSSIBILITY TO CREATE SUCH MODULES FOR THE IDK, E.G., TO BE ABLE TO GENERATE CODE FOR NEW TARGET PLATFORMS.

PROJECT: GRASIA!

VERSION: 0.1

DATE: 26/05/2004

AUTHORS: Jorge J. Gómez Sanz and Juan Pavón

INDEX

1. INTRODUCTION

This document introduces to the *INGENIAS Development Kit* (IDK) and the INGENIAS methodology. Based on the agent-oriented paradigm, IDK allows rapid application development, decoupling of specification and implementation, and verification of the specification according to the needs of the implementation.

The IDK has two parts:

- *INGENIAS Editor*. To generate specifications of a MAS using generic agent concepts. This editor bases on a definition of the elements that built a Multi-Agent System.
- *INGENIAS Code Generators*. It is a framework that facilitates traversing a specification and producing a single output. This framework has been integrated with the INGENIAS Editor so that developers can include new plugins in the IDE automatically. These plugins process directly existing diagrams in the tool through an API that is supplied to the plugin. A plugin can also generate code from information found in the diagrams. As we are original, we call these special plugins, *code generators*. A code generator is a piece of software that can traverse different diagrams and generate a piece of code that uses data from these diagrams. Code generators are a feasible solution to overcome the gap from specification to implementation.

INGENIAS methodology provides the concepts and theory to use the IDK. The IDK is the official support tool for the INGENIAS methodology. The INGENIAS methodology bases on the results from MESSAGE [Caire 2001] to build an ontology and semantics in form of a sets of meta-models. The use of these meta-models is organized according to a set of activities and a development process.

1.1. THE GRASIA! GROUP

grasia! is a research group in the Dep. of Computer Science and Programming of the Universidad Complutense Madrid (Spain) working in the development of software agent systems. Our aim is to implement intelligent applications for information processing in distributed environments (such as internet). We support added value services through personalization and semantic processing of information, improving information access and processing for both users and service providers.

We have a pragmatic view of the application of software agent technology, so we focus on engineering issues. This means that we intend to develop systems that work and are feasible in distributed environments. In order to achieve this objective we have defined a methodology and architecture for multi-agent system development and deployment, INGENIAS.This has been experimented in different application areas:

- Information recovery
- Intelligent interfaces
- Workflow management systems
- Web site personalization
- Virtual user communities
- Security in agent systems
- Ubiquitous computing

If you want to collaborate with our group, just write to jpavon@sip.ucm.es. Collaboration and contributions can be made in several ways:

- Providing new examples of specifications and/or modules for the IDK
- Contributing to enhance the INGENIAS specification with new aspects like ontologies, knowledge representation, and so on.

- Notifying and/or repairing bugs in the IDK.

1.2. SCOPE AND USE

This document has been written for dissemination purposes as well as to be used as reference manual.

The information contained in this document is property of the *grasia!* Research group, so if you use the information contained here or the IDK software, we would like you to include a reference to it in your papers. You can take the following as an example

[grasia 2004] Grasia! Research Group (2004). *INGENIAS Development Kit: Tutorial and Manual*. Technical Report. Facultad de Informática, Universidad Complutense de Madrid, Spain. Available at *http://ingenias.sourceforge.net*

And email us in order to update our referred paper lists. Please, remember that in the Academy world, being referred is what allows us to justify the quality of our work, and … it is free!!

1.3. CREDITS

To the members of the Grasia! Research group, specially to the authors of this paper (Jorge J. Gómez Sanz, and Juan Pavón). Also to the people that collaborated in the development of the IDK (Ruben Fuentes) and those who participated as beta-testers (Guillermo Jiménez, Juan Antonio Recio, Carlos Celorrio, Alberto Fernández, and many others).

This work has been developed in the project INGENIAS (TIC2002-04516-C03-03), which is funded by Spanish Ministry of Science and Technology.

1.4. GUIDE TO READ THE DOCUMENT

The document is structured in three parts. The first one presents, briefly, INGENIAS, its models, and how to represent concepts. The second presents the editor, how it works, and some hints and tricks to make your documenting work easier. The third part is the development part. It shows how to use the IDK framework to traverse diagrams and generate code or build a customized code generator.

2. THE IDK

This section presents, in order, install instructions, the editor, and code generation tools.

2.1. INSTALLING THE IDK

The IDK is officially distributed in http://ingenias.sourceforge.net in form of a zip file. You have to download it and unzip it into your favourite folder, say c:\foo or /home/luser/foo.

To run the IDK we recommend a **j2sdk 1.4.2** or higher. Lower versions do not have some enhancements that newer versions have, like regular expressions handling or clipboard transfer utilities.

To test the install, you may want to start the editor, the main component of the IDK. If you are in the *foo* dir, it can be run directly with

```
foo> java –jar lib/ingeniaseditor.jar
```

However we recommend to use the *ant* tool instead. The ant can be downloaded from *http://ant.apache.org*. This tool not only starts the editor, also it allows to recompile the editor, compile attached modules, run tests, and other interesting features for developers, specially.

Concrete ANT install instructions can be found at http://ant.apache.org/manual/installlist.html. The command in that case is

```
foo> ant runide
```

Where *ant* is the script that launches the ANT system. In order to make it work, you need to have installed ANT before. to install ANT go to http://ant.apache.org and follow its install instructions. Of course, this will not work if the *ant.bat* or *ant.sh* script is not in the *PATH* environment variable.

2.2. THE EDITOR

The main purpose of the editor is to generate a specification of a MAS. A specification here is a set of diagrams build up with in a concrete way, detailed by a set of meta-models, as in UML. The diagrams constitute a project, and they are organized using package-like constructs.

This editor saves these specifications using XML, so that other external tools can analize and produce other kind of outputs. Also, this editor provides access in runtime for installed plugins, and is able to upload plugins in run-time. These plugins are called in this document *modules*.

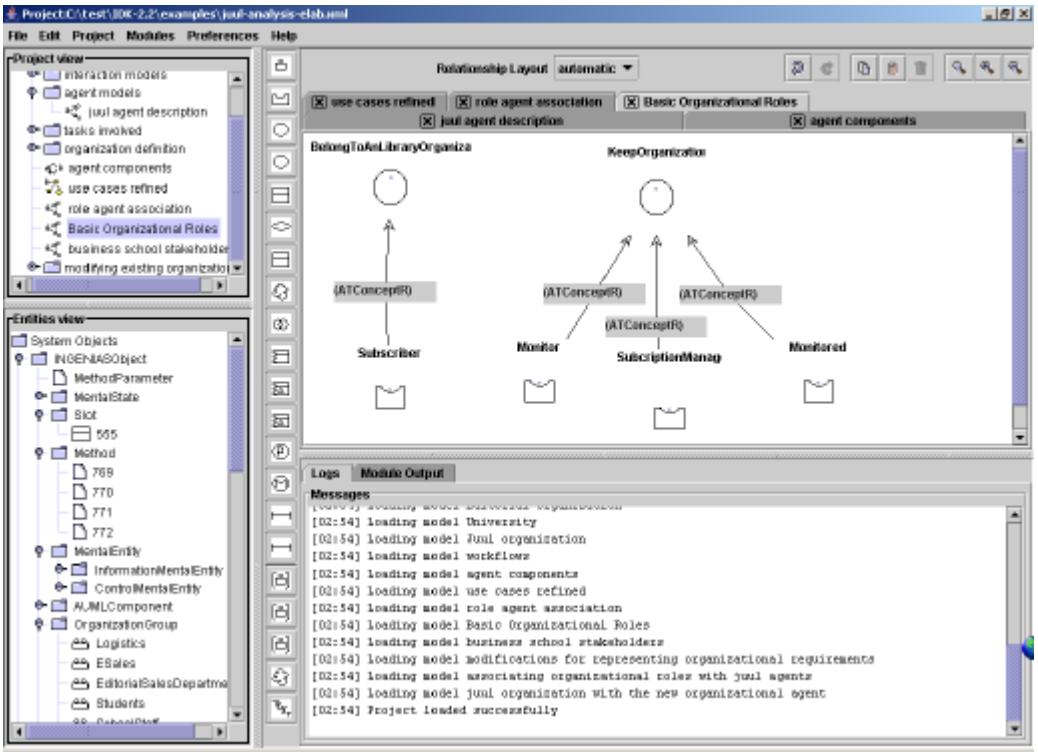


Figure 1. A screenshot of the editor

2.2.1 Parts of the editor and associated actions

Figure 2 shows labels for the different parts of the editor. The purpose of each section is the following:

- *Project view*. It shows a tree based view of your project. Folders represent packages and leaves of the tree (with special icons) represent diagrams. Frequent actions in this section are
  - **Drag and drop**. You can drag and drop a diagram into a package or a package into another package
  - **Open a diagram**. By double left clicking on the diagram
  - **Create a package**. To create a package, first select a package, then right click with your mouse and select *add package* in the pop up menu. Once selected, write down a name.
  - **Create a diagram**. To create a diagram, first select a package, then right click with your mouse and select the type of diagram in the pop up menu. Once selected, write down a name.
  - **Remove**. Select the diagram or package, and then right click with your mouse and select *remove package/diagram*
  - **Rename**. Select the diagram, and then right click with your mouse and select *rename*. Write down the new name, and then hit accept.
  - **Modify properties of a diagram**. Some diagrams have special properties. These are accesible through the pop-up menu triggered on right clicking on a diagram icon. Properties are modified the same way as object properties

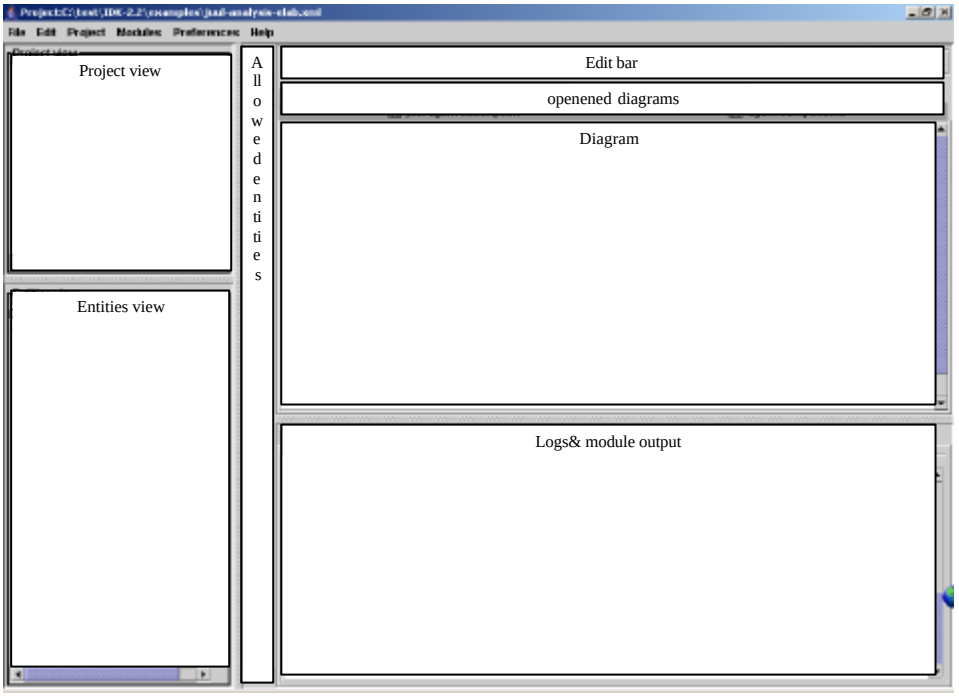


Figure 2. Parts of the editor

- *Entities view.* It contains a tree-like view of what types of entities do exist in the specification. The tree shows types as well as type instances. Type instances are distinguished by icons different of folders. Types are represented by folders. Frequent actions in this section are activated by selecting an icon (not a folder) and pressing the right button of your mouse. In the pop-up menu you can:
  - **Add the selected entity to the current diagram**. It creates a copy of the entity in the diagram, but only if the diagram can handle that specific type.
  - **Remove the entity**. It removes the entity from all diagrams and set all attributes of entities pointing at it to null.
  - **Edit properties**. It shows different properties associated to the selected entity.
- *Allowed entities.* This bar has buttons that create in the opened diagram instances of different type.
  - **What kind of entity is this?**. To know the kind of entity you can let your mouse over a button and wait some seconds.
  - **Inserting an entity**. Press one of the buttons to proceed. New entities will be allocated in the top-left of the *diagram* section.

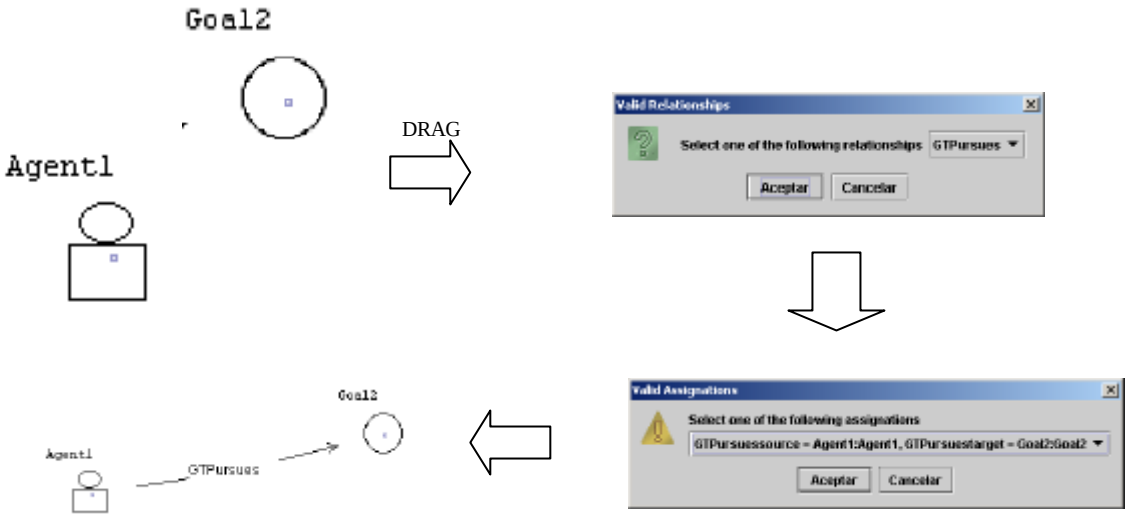


Figure 3. Steps to follow to create a relationships among two entities

- *Diagram.* Selected diagrams are shown in the section titled diagram. These diagrams appear in the *project view.* Frequent actions in these diagrams are:
  - **Changing the icon of an element.** Some elements have different associated views. You can select one of them by right clicking on an element and going to the *views* option. There, available views will be shown (see Figure 7).
  - **Connect two entities with a relationship**. Say that you have to entities, like an *agent* and a *goal* (see Figure 3). To connect one with another, first look for the little square in the middle of these figures. Then drag from the little square to the other entity. You will see that the target entity highlights when the relationship has found the destination. In that moment, release the mouse button. A new window will appear, indicating that the relationship is not allowed in this diagram or showing different possible valid relationships that could be defined. You have to select one. Afterwards, it asks how to configure the extremes of the relationship, since, sometimes, several assignments are valid. On finishing, a new relationship is created.
  - **Adding a new entity to an existing relationship**. Extension mechanism consists in considering the relationship as another entity and repeating the drag & drop process presented before. This time, only a relationship type is presented in the second step, and only possible valid types of extremes for the current new entity are considered.
  - **Edit an entity**. By double left-clicking on an entity. A new window will appear with data that can be edited in several ways.
    - **Text fields.** Just write whatever you want to. It should admit special latin characters (ISO-8859-1)
    - **Combo box fields.** These fields can admit only values defined in the associated list (see Figure 4). A value has to be selected.
    - **Diagram reference fields.** This field allows to refer to other diagrams (see Figure 6). The procedure consists in selecting in the combo the name of the diagram. The combo will show only existing diagrams of a preconfigured type. Once selected, press *Select one model*. This will make the *current value* label change. To jump to the selected diagram, you can press *show selected*.
    - **List box fields**. These fields are used to store references to entities already defined in some diagram or create new entities. They also can refer to collection of values or a single value.

- **Collection.** The list should appear initially in blank. By left-clicking in the list a pop-up menu will appear with four options

  o **Add existing.** A dialog window will appear with a combo box showing valid already defined entities that could be used. Select one and press *yes*

  o **Add new.** A dialog window will appear with a combo box showing valid types of entities that could be used. Select one and press *yes.* Another window will appear to fill in the data of the new entity

  o **Open selected.** It opens a window that shows the data of the selected entity. This window allows the same functionality to edit the data as presented here. So proceed recursively.

  o **Remove selected.** It removes the entity from the list but not from the main repository visible in the *Entities View*

- **Single value.** The list (see Figure 5) will show possible types that could be allowed in that part of the definition. You have to select one of them and press one of the available buttons. With *Create new*, you will create a new instance of the selected type and associate it directly with this field. The new instance will appear too in the *Entities View.* With *Select Existing*, you will associate this field with an existing entity. Existing entities of selected type will be shown in a dialog window in form of a combo box.

o **Insert an entity.** It is enabled when right clicking in this section. A pop-up menu will appear with the different valid entities type. This action is the same as pressing an entity button in the *allowed entities* bar.

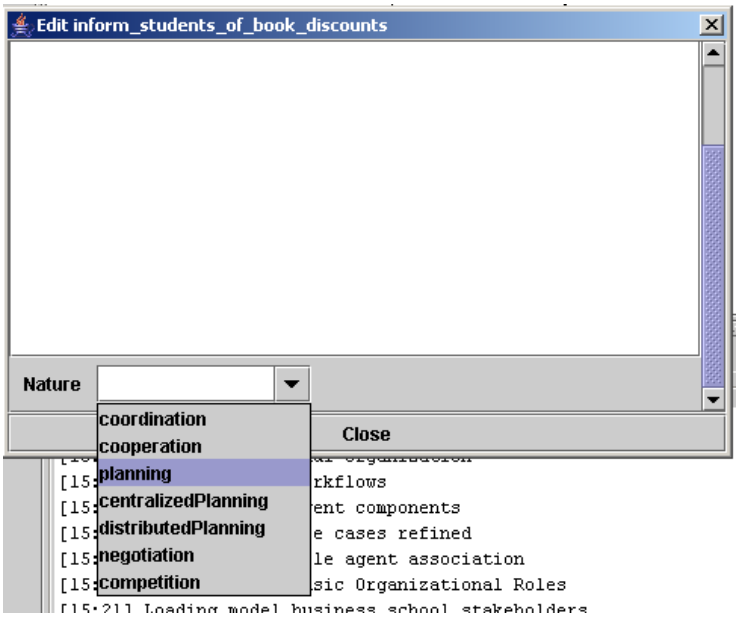


Figure 4. An editing window for an entity showing a combo box with fixed values

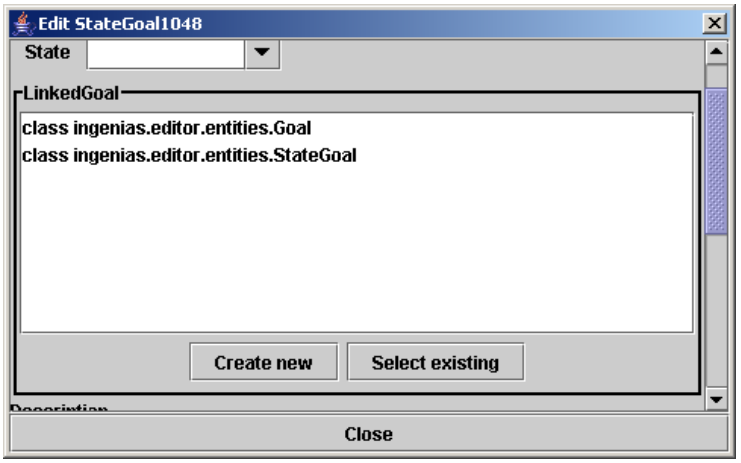


Figure 5. An editing window for an entity showing how to modify single value entity field.

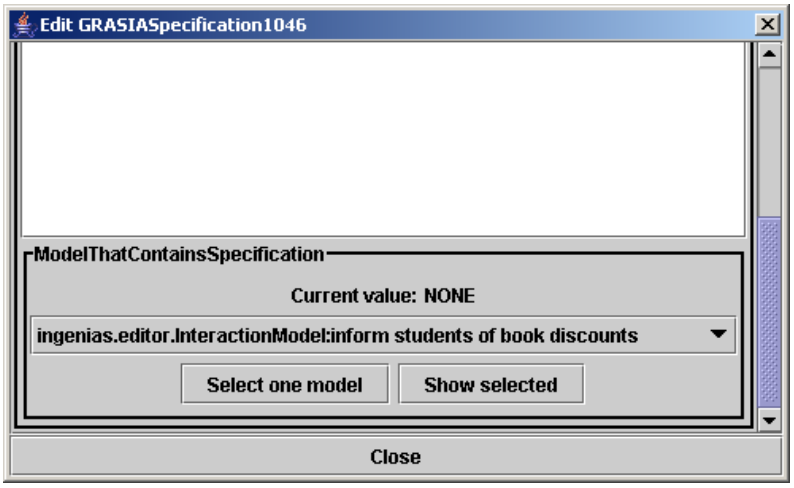


Figure 6. An editing window for an entity showing a field that refers to another diagram

Figure 7. Different views associated with a GRASIA Specification. First view is the *icon* view, and the second the *box* view

- *Logs & Module output*. This section shows messages from the editor or from different loaded modules.
  - **Clear the output/log**. This action is activated by right-clicking in this section. A pop-up menu will appear with a *clear* option. Select it.
- *Opened diagrams.* These are several labels with the title of opened, and not closed diagrams. Current diagram is highlighted in light grey. Other diagrams appear in dark grey.
  - **Close diagram.** Click with your mouse in the cross.
  - **Reopen diagram.** Click on one of the dark grey labels and the diagram will appear again.
- *Edit bar.* This bar offers options to
  - **Zoom/unzoom**. By pressing the and the buttons. The zoom will return to its normal state when pressing the button.
  - **Redo/undo actions**. By pressing the and buttons. Undo/redo actions should be limited to changes of positions of diagram components. It will not work to undelete entities, relationships, or unedit changes made to properties.
  - **Copy/paste/delete**. By pressing the , , or respectively. Relationships cannot be copied or pasted. If some are selected, the editor will unselect them automatically. Deleting an entity requires to delete first the relationships it participates into or the edges that connect the to-be-deleted entity to the relationship. In some cases, when the mandatory arity of the relationship would be violated, there is no other option but deleting the relationship before.
  - **Relationship layout**. It sets how relationships are lay out into the diagram. *Automatic* stands for *allocating the relationship in the middle of all participating entities*. *Manual* stands for *you are responsible for allocating the entity.*
- **Main menu.** This menu provides access to some key functionality.
  - **File menu**. This menu contains a list of recently loaded/saved files. By clicking on one of them, you will get them loaded. Also, you have the usual options here (load,save, save as) that do not deserve further explanation. Besides, the *new* option creates a new empty project. This can be useful to start from scratch.
  - **Edit menu**. It has some of the functionalities of the *edit bar* (copy,paste,delete,undo,redo). Also, it includes three new ones:
    - ***Select all*.** This option selects all elements in a diagram. It is useful to move all elements in the diagram. It is not good for deleting, since it may cause error dialog windows to appear.
    - ***Copy diagram to clipboard.*** It creates an image of the current diagram and stores it into the O.S. clipboard.

- ***Copy diagram to file.*** It creates an image of the current diagram and stores it into a file with two different formats *jpeg* and *png*. We advise to use png, since it ensures the diagram quality.

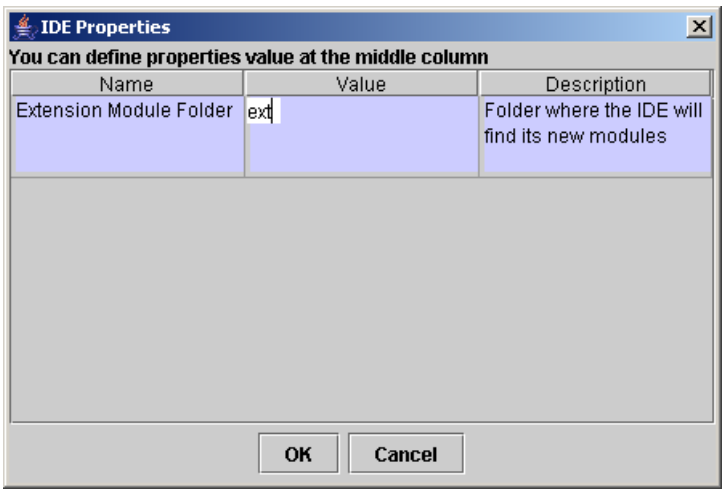


Figure 8. Properties of a project. You can edit only the *value* column of the window.

- ***Project menu*.** It permits to add diagrams or packages to a selected package exactly as done in the project view. Select the kind of diagram or the package and write down its name in the dialog window that will be shown. Also, this menu provides access to the *project properties window.* This window, see Figure 8, shows a table with three columns. The first is the name of the property, the second its value, and the third some text that describes the purpose of the property. Initially there is only one property, the one that defines where to look at new plugins or modules. However, whenever a new module is loaded, it can define new properties that will be added to this windows. Usually, this property refer to configurable execution parameters of the module. Note that you can only modify the second column, the one titled *values*.

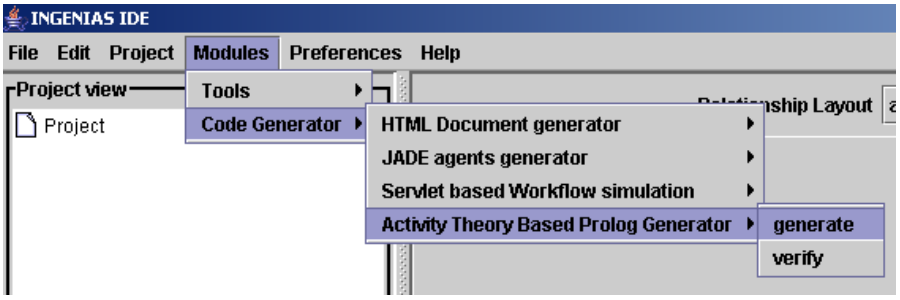


Figure 9. Module list available in the 2.2 version of the IDK

- **Modules menu.** This menu gives access to the different modules installed in the editor. Each module has an entry that can be allocated in the *tools* or *code generator* section (see Figure 9). The *tools* entry contains modules whose main purpose is not generating code but analizing specs to generate reports or detect inconsistencies, for instance. The *code generator* entry contains modules that able to generate code from diagrams. The concrete procedure will be explained later in this document. By now, it is enough to know that modules can both generate and verify a set of diagrams. This list of modules can be updated if a developer allocates a new module in the extension folder. Also, if the module has the same name as an existing one, the new version will replace the old one.
- **Help menu**. It provides access to a summarised version of this document (*tool manual* option), the credits, and the possibility of forcing a garbage collection to optimise memory usage. According to SUN specs, it seems that calls to the garbage collector do not imply an immediate garbage collection.

2.2.2 Working with the editor

Using the editor requires having read some documents about the INGENIAS methodology. For a quick guide we recommend [Pavón 2003] and/or reading the INGENIAS section of this document. The original work is available only in pdf and spanish at http://www.fdi.ucm.es/profesor/jpavon/doctorado/practicas/material/tesis.zip.

In general, the use of the editor is quite straightforward, but here are some advices

1. **Think in advance what diagrams you will need and create packages for them in the project.** E.g., if you need diagrams to describe how an agent performs a task A, create a folder labelled *specification of task A execution*, for instance (see Figure 10)

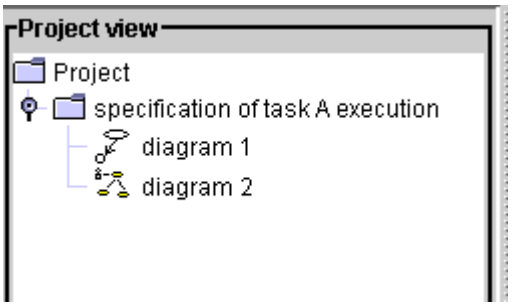


Figure 10. Organize your work with packages

2. **Use meaningful names.** This will help to trace diagrams and make the documentation more readable. Also, fill in the description fields of each entity and diagram when possible
3. **Don't be afraid if there are too many diagrams.** This is usual in any conventional development, why not in an agent oriented one? That is why we recommend start using packages since the beginning. Anyway, you can always create them later and use the drag&drop feature to rearrange them.

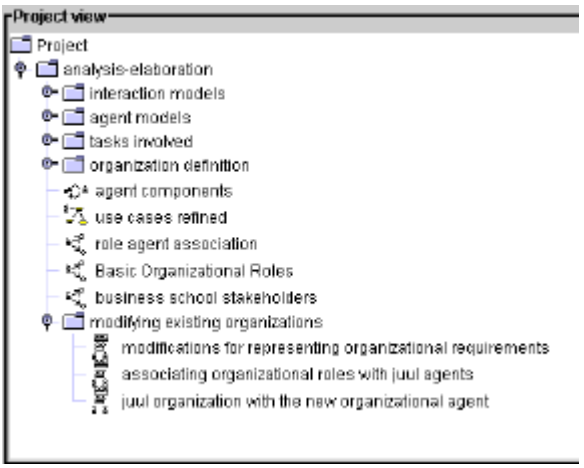


Figure 11. The number of diagrams can grow easily if you get into details. See how a package structure helps to manage them

4. **Copy diagrams to your clipboard.** Remember that the editor allows you to copy a diagram to your favourite text editor by copy&paste. The quality is rather good, but perhaps you prefer to save the image to a file, so that you can include the image in a latex file or similar.

3. AUML

The IDK Editor includes an alpha version of AUML Protocol diagrams. These diagrams are defined according to the last AUML draft that can be found in http://www.auml.org

3.1. CREATING A PROTOCOL DIAGRAM

AUML Protocol diagrams are created as other types of diagrams, by clicking on the *project* folder in the *project view* (see Figure 12) or the *Project* option in the main menu. In any of these two cases, you have to select *AUML Interaction Diagram*

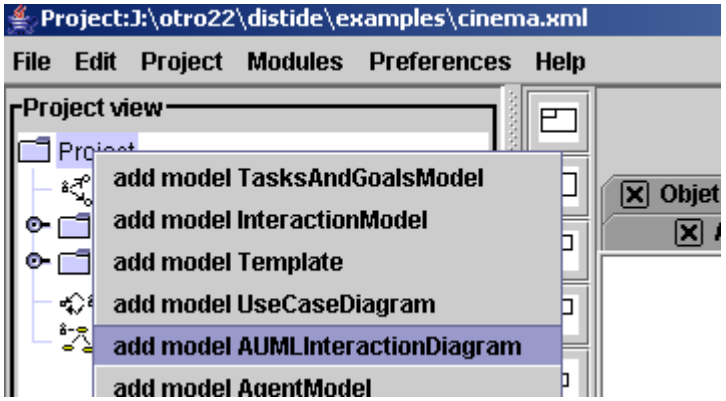


Figure 12. Selecting an AUML Protocol diagram type

The icon that represents a diagram of this type is the same as the one belonging to a common interaction

3.2. DEFINING PROTOCOLS

The protocol definition starts by pressing the *protocol* button in the diagram button bar. This corresponds to the first button as remarks Figure 13

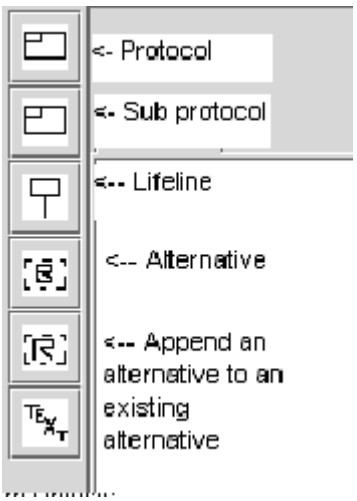


Figure 13. Buttons for defining the agent protocol

As a result, a protocol box is created. This Box shows the title of the protocol only (see Figure 14). You can edit the name of the protocol by modifying the id field. You will see how it changes in the diagram.

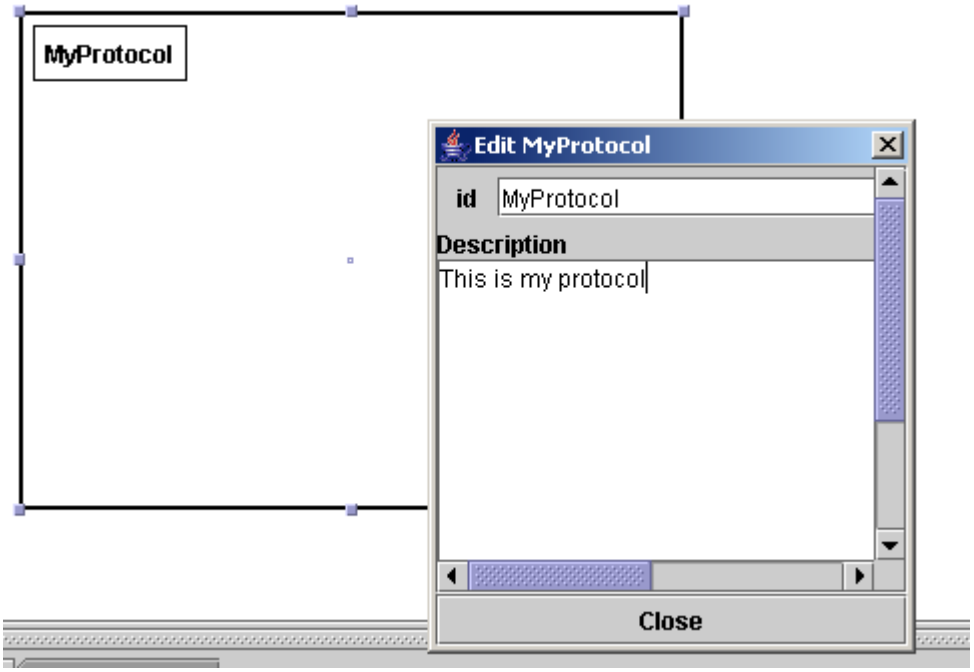


Figure 14. Protocol box renamed to *My protocol*

Once the protocol box is created, it is time to create some life lines. In this implementation, a lifeline is an entity that represents an object instance that is going to send messages to other objects.

To create a lifeline, a protocol box has to be selected first. Lifelines are created with one of the buttons that you can see at Figure 13. The result can be seen in Figure 15.

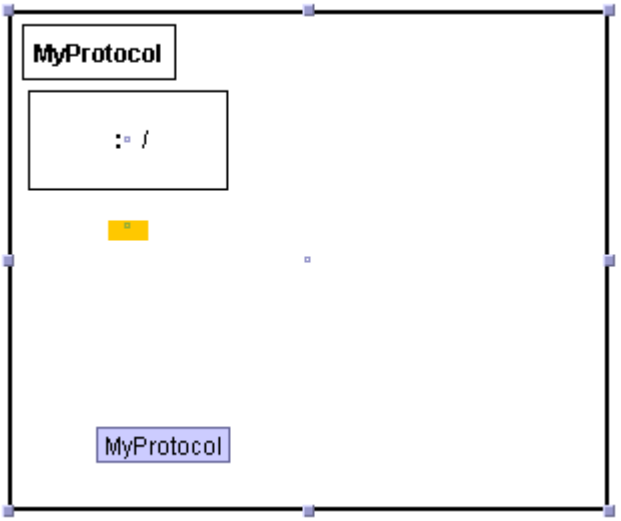


Figure 15. Protocol box with a lifeline

Lifelines do not show any special information until they are edited. Relevant data is (see Figure 16):

- **Name:** The name of the instance
- **Agent:** the type of agent that represents this instance. You can create a new kind of agent or select an existing type

- **Role:** The role that the agent instance is playing at the moment. You can create a new kind of role or select an existing one.

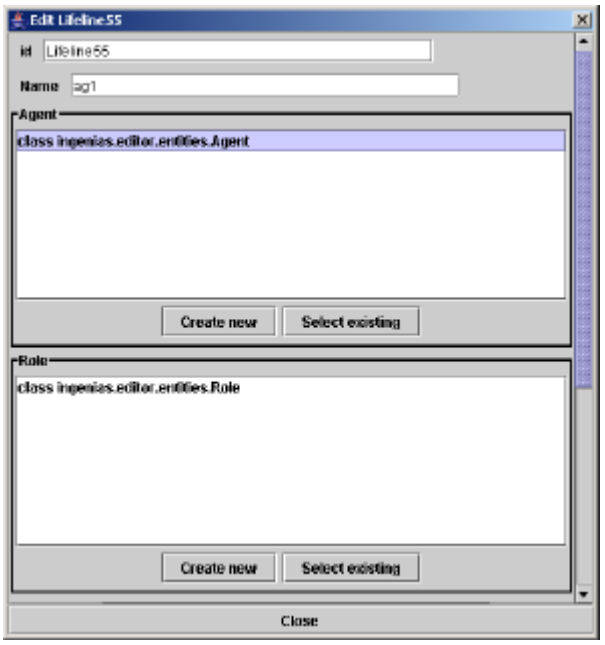


Figure 16. Data that personalizes a life line

After modifying the different fields, you will see that the representation of the lifeline incorporates the new data (see Figure 17).

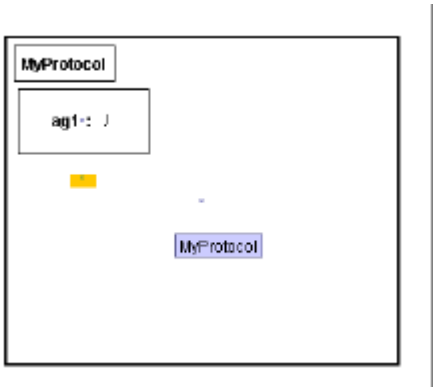


Figure 17. Modified lifeline after typing the name of the agent instance

Now, we will create another lifeline and start defining the messages among agents. The process is the same as before, but this time we, previously, moved the first lifeline to the right so that there is enough space for the new one. Please, remember to select the protocol box before creating the lifeline.

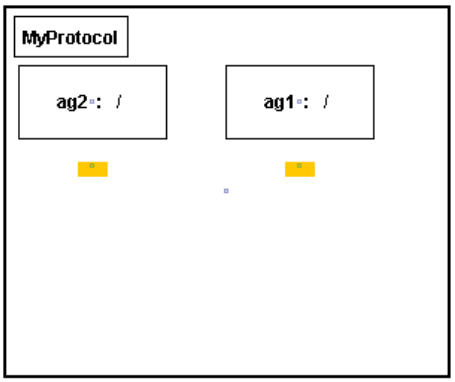


Figure 18. Two lifelines in a protocol box

3.3. CONNECTING TWO LIFELINES

Connecting two lifelines is the first step for defining a message passing, an alternative definition, or a protocol instantiation. To connect, notice the little square drawn in the orange rectangle. By dragging from one to the equivalent on the other lifeline, we get something like the following.

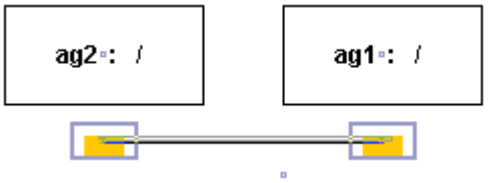


Figure 19. A connection between two lifelines

The process is the same as when connecting two entities as shown in section 2.2.2. So the next step is selecting the kind of association between the two lifelines. At the moment there are three.

Figure 20. Dialog window for determining the type of connection between two lifelines

What happens on selecting one of these options will be explained in the following sections.

3.4. SENDING A SIMPLE MESSAGE

On sending a message, the result is the one shown in Figure 21. It shows an arrow from one lifeline to another and a square in the middle that will hold message data. Notice that after creating the arrow, the orange squares have grown.

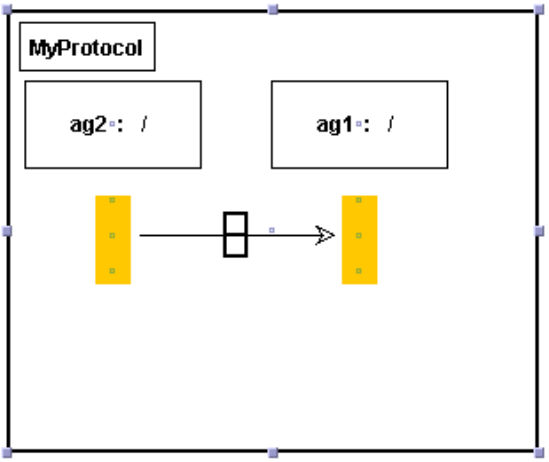


Figure 21. Result of selecting “AUML Send simple”

The data that can be configured in the message is:

- **Label.** This label will be shown in the diagram in the top part of the message
- **Speech act.** It is a combo with predefined names of valid FIPA ACL speech acts

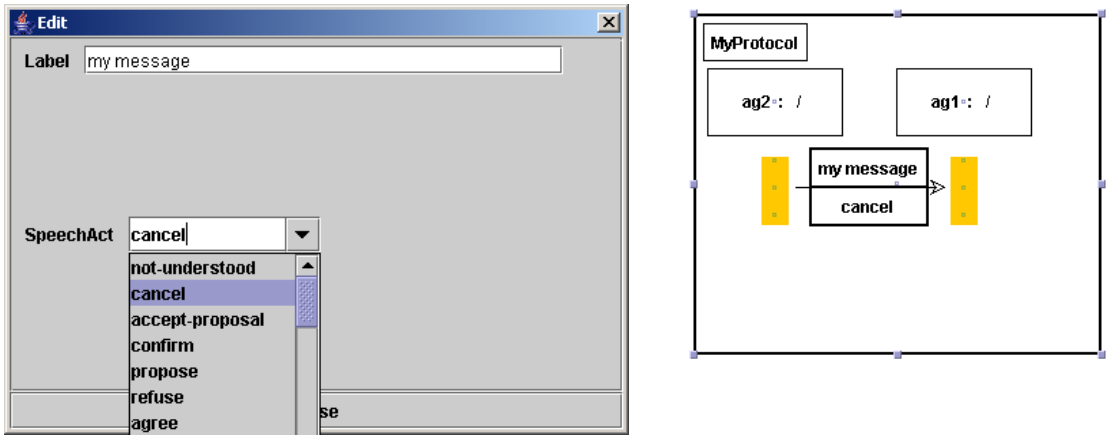


Figure 22. Data for a message (left) and resulting drawing (right)

You can repeat the process as many times as you need.

3.5. CREATING AN ALTERNATIVE

You create an alternative when you select AUMLSelection in Figure 20. As a result, the editor creates an alternative box between the two lifelines (see Figure 23).

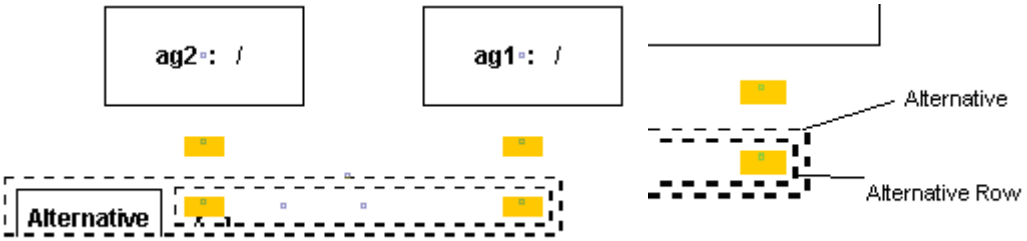


Figure 23. Alternative between ag1 and ag2

Figure 23 shows an alternative and an alternative row. The first one is just a container for the different possibilities that may consider a message selection in the middle of a protocol. Each one of these possibilities is an alternative row. Alternative rows can be edited and their attributes modified. In this case, the attribute is the condition that this alternative row represents

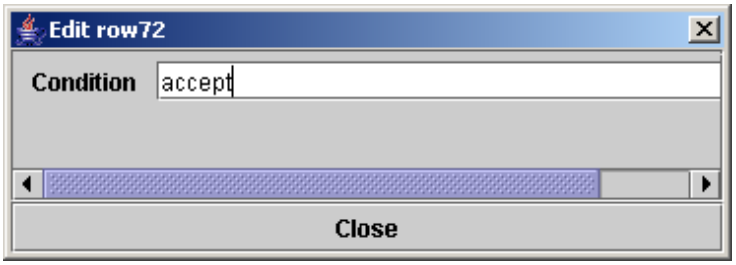


Figure 24. Defining a condition for an alternative row

You can draw messages from one alternative lifeline to another, just as you did in the previous section (see Figure 25)

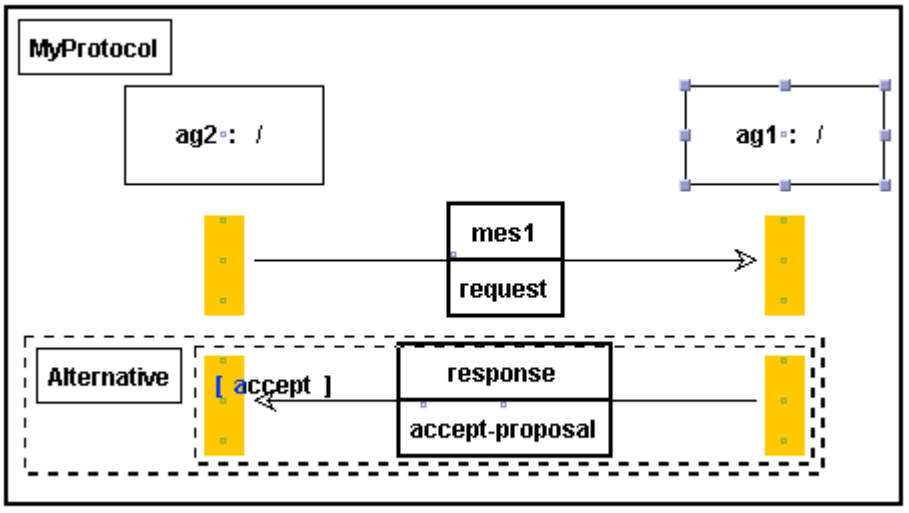


Figure 25. Drawing messages between two lifelines with an existing alternative

To add a new Alternative Row, we select the alternative again and press the alternative row button

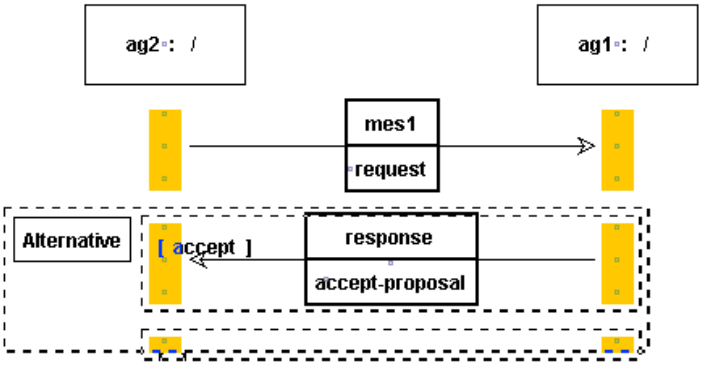


Figure 26. Defining another alternative

Again, we define a condition for the alternative row and define the a message as well. The result is shown in Figure 27.

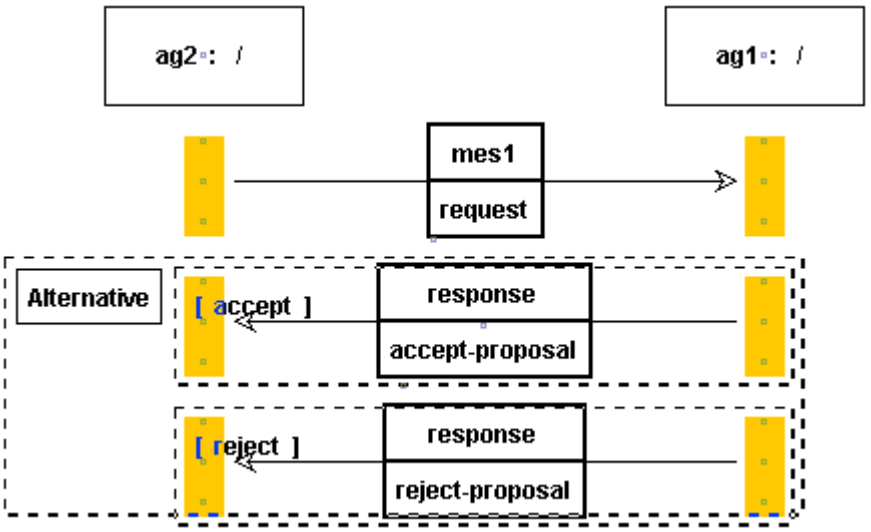


Figure 27. Alternative with two posible responses from ag1 to a request sent by ag2

3.6. CREATING A SUB PROTOCOL

A sub-protocol refers to another protocol in order to make use of it. In the example, Figure 28, we create another protocol in a different diagram and intend to reuse a previously defined diagram, in this case, the one defined in Figure 27. We start the definition of the subprotocol as in the Figure 20 and select *AUMLUseProtocol*.

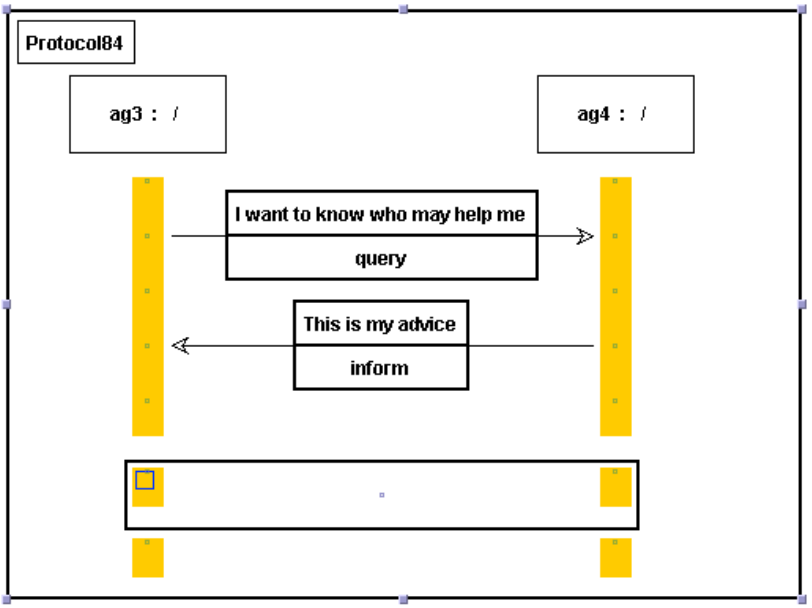


Figure 28. Subprotocol creation

Figure 28 shows a subprotocol. It is a rectangle that needs to be instantiated. To instantiate it, we select it and try to edit its containment (right click → select edit or double click).

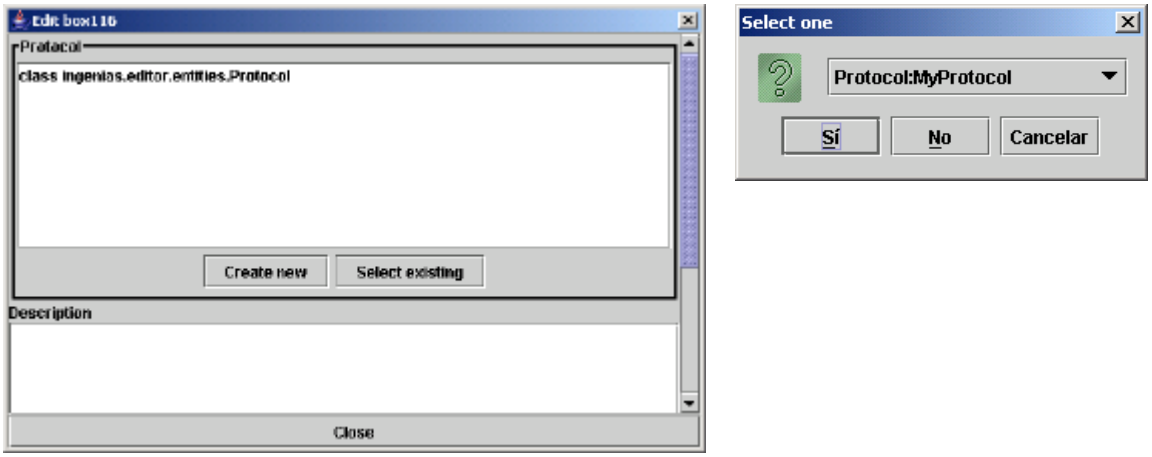


Figure 29. Edit dialog to fulfill the subprotocol

Figure 29 shows the edit dialog. There, we have to choose either *create new* or *select existing*. In this case, as we want to reuse an existing protocol, we chose *select existing* and then *My protocol*, which is the one we defined in in Figure 27. The result is shown in Figure 30.

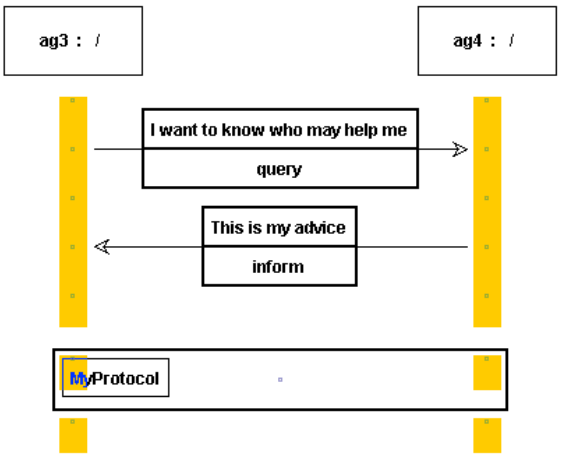


Figure 30. Initialized subprotocol instance

3.7. TO BE DONE

There are several aspects that have not been completed in this version (that's why it is an alpha). In concrete, we could highlight:

- **Layout problems**. Alternatives and Subprotocols are always appended at the bottom of the life lines. This is correct for alternatives, but not for subprotocols. Besides, the gap between orange squares is something we want to remove.
- **Translate other AUML protocol diagrams primitives.** So far, only alternatives, basic messages, and subprotocols have been translated.
- **Semantics of AUML primitives.** Throughout the implementation of the diagrams, many questions have arised that were not considered in the FIPA AUML Draft diagrams. For instance, what kinds of connections were allowed. Is it correct to nest a subprotocol in an alternative? Does the lifeline end in an alternative (can we define messages after an alternative)? What if I add another lifeline to an existing alternative and the new lifeline already has an alternative defined? These an another questions makes us think that there are still missing aspects in AUML that need to be further detailed.

-

4. MODULES

Modules (see Figure 31) are programs that process specifications and produces one or many of the following outputs:

- **Programs.** There is an infrastructure that facilitates the generation of sources. The infrastructure bases on templates defined with XML. These templates are filled in with information extracted from the diagrams.
- **Reports.** Diagrams can also be analysed to check, for instance, if they have certain elements; if special semantics, defined by the developer, are being respected; to collect stats of usage of different elements.
- **Modifications on current diagrams.** Though this feature is in beta stage, a module could insert and/or modify entities in the diagrams, or insert/delete diagrams. This feature is useful to define personal assitants that interact with the tool.

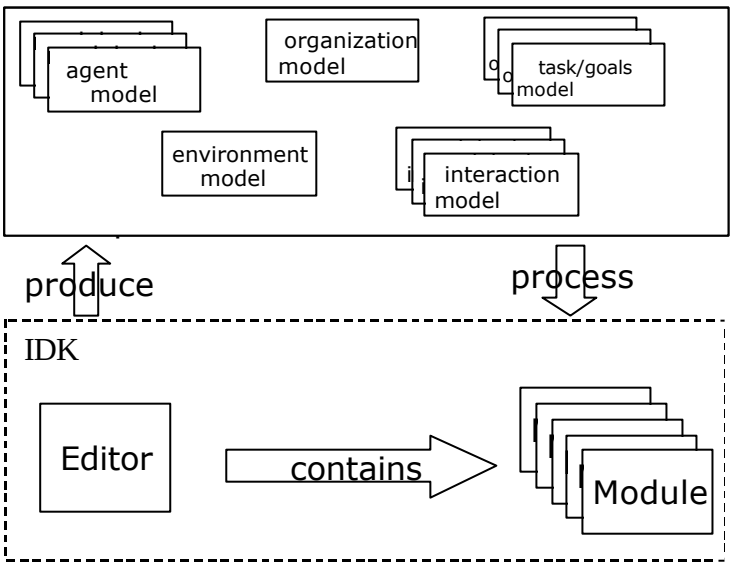


Figure 31. Relationship between the IDK, models and components

Modules are built in the top of a framework made to traverse specifications, extract information from those specifications, dump the extracted information into templates. Developing a module implies following several steps:

1. **Producing a prototype.** A developer would centre into one or more features, easy to implement if possible. These features would be expressed with diagrams that an analyst would have produced.
    a. **Tools:** a conventional development environment.
    b. **Output:** a prototype that realizes a small part of the specification
2. **Marking up the prototype code.** Parts of the prototype should match against parts of the specification. As a result, a developer identifies the possible mappings from the specification to the prototype code.
    a. **Tools:** an XML editor or a text editor
    b. **Output:** prototype code marked up with tags. The marked-up pieces of source code are known as templates.
3. **Generate/modify a module.** The module will traverse the specification and obtain the information required by the prototype. This can be done with a conventional Java development environment.
    a. **Tools:** a conventional J2SDK development environment

b. **Output:** one or more Java classes that extend BasicToolImp or BasicCodeGeneratorImp classes. Other classes may be created as well.

4. **Deploy the module.** Java classes and templates are put together into a jar file. This jar file is created in a specific folder where the IDK Editor can load it.
   a. **Tools:** J2SDK and ant. The J2SDK jar tool generates the jar, and the ant tool executes the appropriate ant task that perform the compilation and copy of the source and binary files.
   b. Output: a *jar* file that has the module code and the templates obtained from the prototype
5. **Test the module.** Testing tasks are launched from the IDK Editor. By executing the module over the specification, the developer can check if the diagram is traversed properly and if all templates have been filled in as they should. Also, as templates demand concrete information, it may be possible that this is not present or that it is not expressed as it should. Therefore, it may turn out that the specification was not correct.
   a. **Tools:** the IDK editor mainly. Other tools may be needed when testing the output code (appropriate compilers and runtime environments).
   b. **Output:** problems with the code generated by the module, problems with the traversal of the specification, or problems with the specification
6. **Debug.** If something goes wrong, debug the prototype and go to
   a. *Step 2.* If there is new code that was not marked up before
   b. *Step 3.* If the failure was in the module and the data traversal
   c. *Step 4.* If the was a failure in the prototype and could be solved without marking up again the code
7. **Start over.** When the module is finished, it can translate diagram specifications into code. However, it performs this tasks with a reduced set of the diagram specification. The next step would be to take the code generated by the module and change it so that it can satisfy other parts of the specification. Therefore, we would go back to step 1.

Modules produce code using a template based approach. Figure 32 shows an illustrative example. A developer defines a template of JESS rule and extracts data from a specification to generate the rest of rules. Rules need a condition, action, and a name. This data is expressed using a concrete structure that will be presented later. As a result, we get two different rules instantiated from the same template.

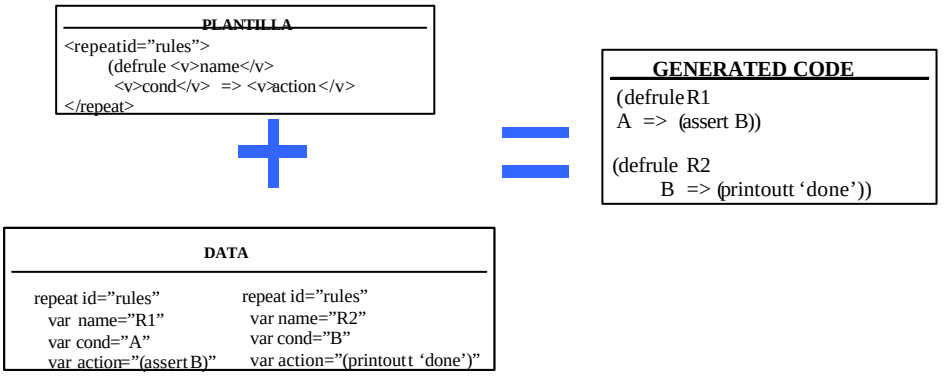


Figure 32. An example of code generation for a set of JESS rules

According to this description, the reader may infer that we assume:

- There are parts of the specification that are very similar among themselves.

- There are parts of the code that are very similar among themselves

One may be a consequence of the other, since code is supposed to satisfy a specification, and if there are parts of the specification that are repeated, therefore, there should be parts of the code that repeat as well.

A module can be of two different types: a code generator or a specification processor. To create a module of the first type, you have to extend the ingenias.editor.extension. BasicCodeGeneratorImp class. If you try to create a module of the second type, you have to extend the ingenias.editor.extension. BasicToolImp class.

Both classes define abstract methods that have to be redefined into their inheriting classes. Also, both classes initialise internal variables that give access to the internal data structure of the IDK. Once created an inheritor of these classes, developers will realize that most of the work is done, and that only traversal specification and template creation needs to be done.

In the rest of the document, there are further instructions for some of these taks. Following, there are instructions for defining a graph traversal (section 4.1). Then, we present how to create templates from prototypes (section 4.2). Afterwards, we introduce the facilities to create the data needed to fill in the templates (section 4.3). Finally, section 4.4 explains how to deploy a module.

4.1. TRAVERSING THE SPECIFICATION

A module views the internal data of the IDK Editor as the data structure of the Figure 33. This data structure is an interpretation of the GOPRR model (please, visit section 6.2 for more details). In this representation, a *graph* is a diagram, a *relationship* an edge from an object to another object, and an *entity* an object.

A relationship here is n-ary. This means that there may be many extremes, not only a source or a target. This capability is useful for representing agent concepts relationships, since most of them are of this kind.

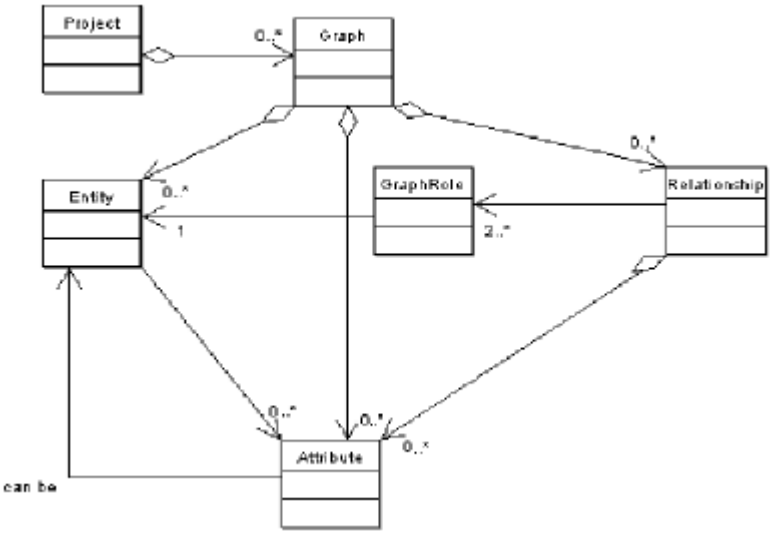


Figure 33. Logical view of the data stored in the IDK

The access to this structure is controlled by a set of interfaces shown in Figure 34. These interfaces take advantage of the commonalities of the elements of Figure 33, such as that all elements have properties (graphs, relationships, and entities).

In the IDK, a developer can obtain instances of *Graph*, *GraphRelationship*, and *GraphEntity* by using a singleton pattern [Gamma 1995]. A class that implements this pattern in the IDK is the *ingenias.generator.browser.BrowserImp* and the method to invoke to get a valid instance of this class is *getInstance*( ).

```
Browser browser=BrowserImp.getInstance();
```

This kind of instantiation is valid only when the module is executed inside the IDK Editor. If you plan to execute outside the editor directly over an specification file generated by the IDK Editor, you should put into your main method the following:

```
File file;
....
ingenias.editor.Log.initInstance(new java.io.PrintWriter(System.err));
ingenias.generator.browser.BrowserImp.initialise(file);
Browser browser=BrowserImp.getInstance();
```

Traversing the specification means to obtain a graph traversal algorithm that goes through elements of the specification and:

1. Ensure that all requested elements are present. A traversal intends to find certain elements and, from them, go to other elements in the diagrams.
2. Extract information from the requested elements. Information extraction is a matter of invoking specific methods of *GraphEntity*, *Graph*, and *GraphRelationship*.

Initially, the developer has a list of existing graphs or a list of existing entities in all graphs. At this point, it is important to clarify that some objects may be already present in different diagrams: we allow repetitions. From this initial graphs or entities list, the developer articulates the traversal. It can be as simple as "*In each diagram, look for instances of the relationship X, and tell me what elements it connects*" or as complex as a traversal towards JADE based code generation.

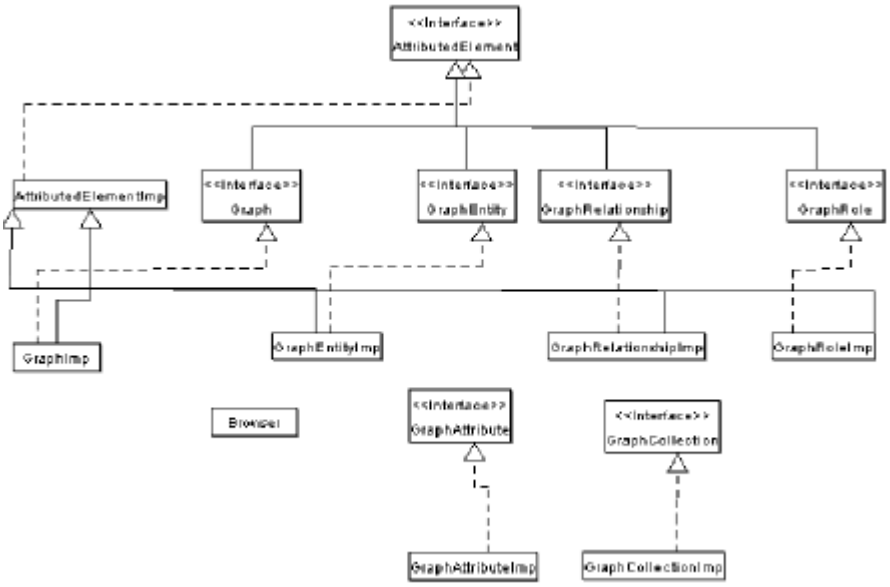


Figure 34. Interfaces provided to access the data stored as **¡Error! No se encuentra el origen de la referencia.** describes

A silly example of how to traverse existing diagrams and printing out their names is the following:

```
    // browser has been previously initialised
    Graph[] gs = browser.getGraphs();
    StringBuffer result = new StringBuffer();
    for (int k = 0; k < gs.length; k++) {
      Graph g = gs[k];
```

```
      result = result.append( "\n###### Diagram " + g.getName()+
                      " ######\n");
      result.append(this.generaInformeDiagrama(g)+"\n");
    }
    System.out.println(result);
```

We use StringBuffer because the concatenation of Strings is very unefficient and it may lead to memory exhausted errors.

4.2. MARKING UP THE PROTOTYPE

The code of the prototype is marked-up according to the DTD shown in Figure 35. This DTD determines that any piece of source code is a XML document. Therefore, templates can be written in any language, provided that the source code can be later marked-up.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT file (#PCDATA | v)*>
<!ATTLIST file  overwrite (yes|no) #REQUIRED>
<!ELEMENT program (#PCDATA|repeat|saveto|v)*>
<!ELEMENT repeat (#PCDATA | saveto | v | repeat)*>
<!ATTLIST repeat id CDATA #REQUIRED>
<!ELEMENT saveto (file, text)>
<!ELEMENT text (#PCDATA | repeat | v | saveto)*>
<!ELEMENT v (#PCDATA)>
```

Figure 35. DTD for extracted information

Tags from Figure 35 have concrete semantics:

- **program**. It is the main tag of the document. It requires no special semantics.
- **repeat**. It means that the text among *repeat* tags has to be copied and pasted again to have a duplicate. The duplicate is parsed following looking for variable instantiation or other meaningful tags.
- **v**. It represents a variable. Its matching tag encloses a piece of text that has to be replaced. The text itself is considered as an id. This id permits to distinguish what data corresponds to this variable.
- **saveto.** This tag orders to save the contained text into a file. The file name and the text are enclosed into specific tags
    - **file**. It is the name of the file. It can contain other tags as well.
    - **text.** It is the text to be saved. It can contain other tags as well.

When writing templates, soon it becomes clear that is not easy to code programs as XML since < and > symbols, which appear frequently, have to be codified as < and > (as demands XML). Doing this for every symbol is a time consuming task. To save effort, we tend to express XML tags in our templates using the at symbol (@) instead of the < and >. This way, instead of writing:

```
<program>
 if (a &lt; b)
  cout &lt;&lt; “hello”
</program>
```

We would write:

```
@program@
 if (a < b)
  cout << “hello”
@/program@
```

Of course, other uses of @ symbol would be forbidden in the code. Both formats can be used to generate code, though specific methods should be invoked in each case.
So that this decision does not affect the rest of the framework, we have prepared some tools that translate one format to another. The command that performs the translation is the following:

```
java –cp “lib\ingeniaseditor.jar" ingenias.generator.util.Conversor [-a2t|-t2a] my_template
```

The a2t means transforming an @ format to the conventional XML format. The t2a is the opposite. This utility also transforms <, >, &, ‘, “ symbols to their equivalents in XML

4.3. GENERATING THE CODE

The code generation facilities takes as input a template that satisfies the DTD from Figure 35 and data to full fill the template *v* and *repeat* tags. The data that feeds the code generator has the structure shown in Figure 36. We name this data *sequences* due to some data structures we used in the past for this purpose. Right now, there are Java classes that implements this structure and provide adequate translation mechanisms for XML.

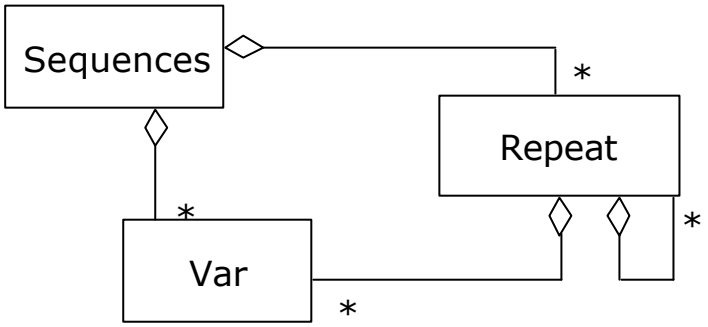


Figure 36. UML Description of the data structure

As Figure 36 remarks, there can be several *repeat* instances and *v* instances. Each one is created using an id, in the case of the *repeat*, and an id and data, in the case of the *var*. The id of *repeat* and *var* is used to distinguish among the different *v* and *repeat* tags that may exist along the template. Data is supplied as an unordered sequence of repeat or var structures and the effect is a replacement of the template by concrete data, in the case of var structures, or duplicates of existing data, in the case of *repeat* structures.

But, what data should I extract and what structure it should have? We answer partially that question with an utility that parses a template and returns text representing how the data structure should look like. The utility is started from command line in the install folder of the IDK

```
java
–cp “lib\ingeniaseditor.jar; lib\xerces_2_3_0.jar; lib\xercesImpl.jar; lib\xerces-
J_1.4.0.jar" ingenias.generator.util.ObtainInstantiationStructure
template_filename
```

As an example of the kind of output, Figure 37 shows the data structure needed to instantiate a template of the html code generator module included in the IDK distribution. The template corresponds to an *index.htm* file.

```
@program xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../plantilla.xsd"@
       @saveto@
              @file overwrite="yes"@
                     @v@output@/v@  /index.html  @/file@
              @text@
<HTML>
<BODY bgcolor="#FFFFFF">

<p><img src="../images/logograsia.jpg" width="151" height="71"> </p>
<p><font size="5">Specification Diagrams are:</font>
  <BR>
</p>
<ul>

 @repeat id="paquete"@
       <li><font size="4"><b>    @v@name@/v@  </b></font> </li>
       <ul>
       @repeat id="graph"@
            <li>Diagram name: <A HREF="     @v@name@/v@  .html"> @v@name@/v@  </A> type :
<font color="#000099">    @v@tipo@/v@   </font>
            </ li>

       @/repeat@
       </ul>
       <br>
  @/repeat@

<p><font size="3">Document generated automatically with the Ingenias Development
Kit <font color="#993333">
  IDK 2.1</font></font></p>

</BODY>
 </HTML>

@/text@
       @/saveto@
@/program@
```

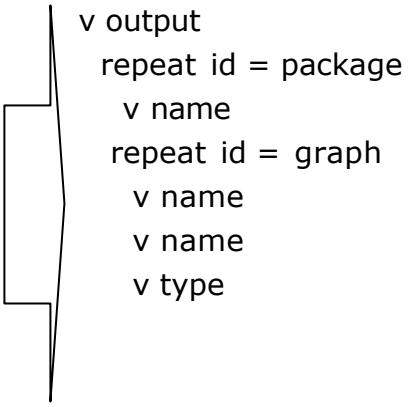


Figure 37. Information structure extracted from the template

If we tried to create a similar Java structure with the classes from Figure 36, it would look like the following

```
Sequences seq=new Sequences();
Repeat r1=new Repeat(“package1”);
Repeat r2=new Repeat(“package2”);
seq.add( r1 ); seq.add( r2 );
r1.addVar(new Var(“name”,”mipackage1”));
r1.addVar(new Var(“type”,”agent diagram”));
r2.addVar(new Var(“name”,” mipackage 2”));
r1.addVar(new Var(“type”,”interaction diagram”));
.....
```

And to launch the code generator, the following code should be executed. The input stream is a stream whose source is a file containing the template. The sequence structure is transformed into a string, whose toString method is overloaded to generate a XML structure.

```
Sequence seq;
InputStream is;
...
ingenias.generator.interpreter.Codegen.applyArroba(seq.toString(),is);
```

The interpreter will analyze the template and will produce the output code. If you used *saveto* tags into your templates, the interpreter will save the results to the files that you defined. If not, the output will be the standard one.

4.4. DEPLOYING

A module has templates, if it is a code generator one, and classes that extend the BasicToolImp or the BasicCodeGeneratorImp. All of them tend to be in folder created by the developer.

To deploy the module, it is required that templates are allocated in a folder named *templates* in the root of the folder structure where module sources are. The deployment consists in:

1. Compiling the sources of the module into a separate folder, which we will call *binary folder*.
2. Copying the template folder into the binary folder
3. Invoke the *jar* utility to compact the module binaries and the templates
4. Move the resulting *jar* to the deployment folder of the IDK, by default it is a folder named *ext* allocated in the IDK install folder

At the end of the process, if the IDK Editor is running, the message panel should show a message indicating that a new module has been added (see Figure 38). Each time you deploy a module the IDK Editor will load it automatically and replace internal references to it with the new version.

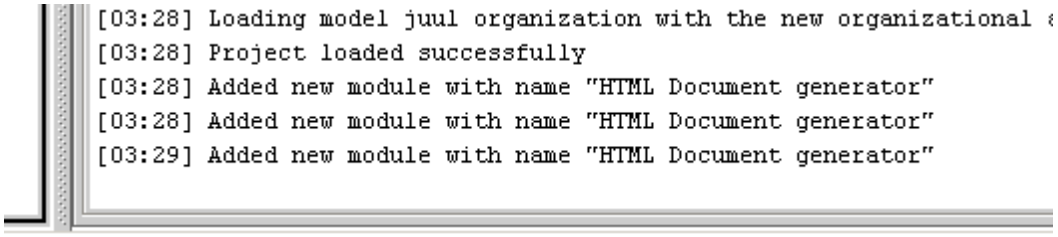


Figure 38. Messages that confirm the module load

These tasks can be automatized if the developer uses the *ant*. In the build.xml file, the developer can find examples of how these tasks look like. For instance, the *modhtml* tasks, listed following (bolder and italics represent comments inserted to facilitate understanding):

```
  Change the location attribute to the path to your module source
folder
  <property name="modhtmldoc" location="modules/srchtmldoc" />

      ....

  <target name="modhtmldoc">

   Here  the folder structure is created
    <delete dir="${temp}" />

    <mkdir dir="${temp}/templates" />

    <depend srcdir="${modhtmldoc.dir}" destdir="${temp.dir}"

    cache="depcache">

      <include name="**/*.java" />

    </depend>

  Now, module sources are compiled

    <javac compiler="modern" depend="true" destdir="${temp}"
debug="true">

      <src path="${modhtmldoc}" />
```

```
    <classpath>
      <pathelement path="${classpath}" />
      <pathelement path="${build}" />
      <fileset dir="lib">
        <include name="**/*.jar" />
      </fileset>
    </classpath>
  </javac>
  After compilation, templates are copied to the binaries folder
  <copy todir="${temp}/templates">
    <fileset dir="${modhtmldoc}/templates"></fileset>
  </copy>
 A jar is created with the name of the module
  <jar jarfile="${modhtmldoc}/modhtmldoc.jar"
  basedir="${temp}" />
  <delete file="${moddeploy}/modhtmldoc.jar" />
 The resulting jar is moved to the deploy folder. Change the name of
the module to avoid collision with other jars
  <move file="${modhtmldoc}/modhtmldoc.jar"
  toDir="${moddeploy}" />
 </target>
```

Only by changing the *modhtmldoc* property, you could get a personalized task to compile and deploy your module. The task would be started with

```
ant modhtmldoc
```

Of course, we recommend not reusing completely the ant task code, copying and pasting the *modhtmldoc* taks into your build.xml file, and modifying the copy trying to personalize if possible, specially changing modhtmldoc with other more appropriate names. For more information about how ant works, we strongly recommend reading the ant manual, which is available at ***http://ant.apache.org***

4.5. MODULE EXAMPLES AVAILABLE WITH IDK DISTRIBUTION

IDK has proven its ideas with several modules that

4.5.1 HTML module

TODO

4.5.1.1 Output of the module

TODO

4.5.1.2 Requirements

TODO

4.5.1.3 Configuration

TODO

4.5.1.4 User manual

TODO

4.5.1.5 Development issues

TODO

4.5.2 JADE module

The case study consists in translating INGENIAS interaction specifications into executable code using JADE Agents communication infrastructures. There are a set of tests that consider different situations that may arise. Later on, this code generator is applied to the Juul Boklander specification to check if it behaves properly.

4.5.2.1 Output of the module

In this case, the target platform is the JADE Platform. The code produced is a set of JADE agents with customized behaviors that implement the protocols described with INGENIAS. A Main class is also generated with the responsibility of launching the generated agents.

4.5.2.2 Requirements

J2SDK 1.4.2. There is a JADE distribution included in the IDK together with its distribution license (LGPL)

4.5.2.3 Configuration

This platform does not need special configuration. JADE libraries are already included in the distribution. First, you have to compile the jade module, though it is already compiled in the IDK distribution. To compile the JADE module from the IDK distribution folder, type:

```
ant modjade
```

Once compiled, start the IDK editor and load an specification like *examples/cinema.xml* (written by Carlos Celorrio and translated to English by Jorge Gómez)

The specification for JADE platform requires the definition of the following diagrams:

- Define the protocol using an Interaction Diagram and defining inside messages exchange and message ordering.

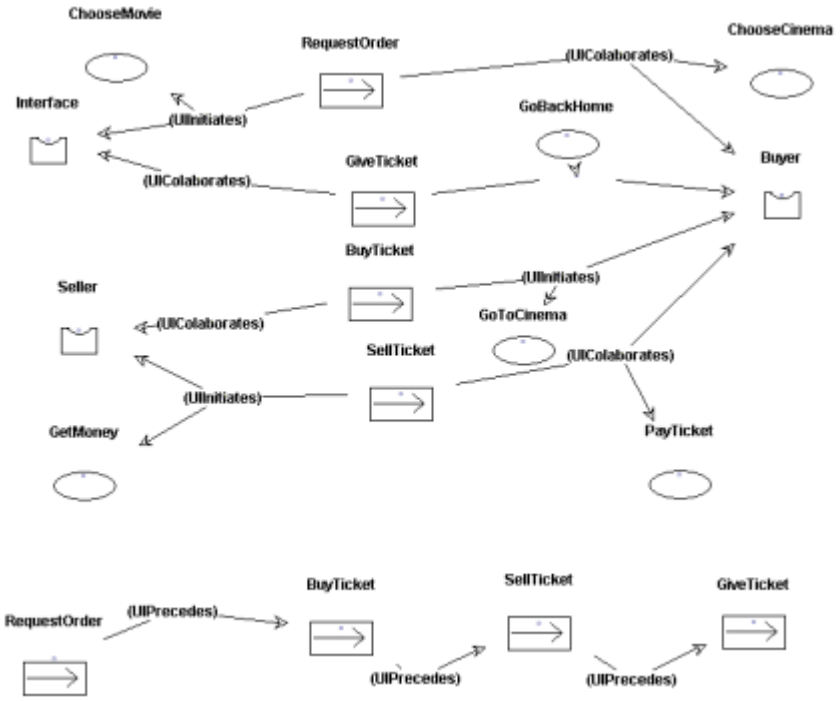


Figure 39. Interaction Protocol definition. Note that there is a separate definition of the messages to be sent (UIInitiates and UIColaborates) and the message order (IUPrecedes)

- Interaction diagrams where Interactions and agents are related (IInitiates and Icolaborates), see Figure 40. It is important also to link a GRASIA Specification entity to the interaction. In this GRASIA Specification, you will have to select the interaction diagram previously defined. Select it in the combo and press *select one* for this.

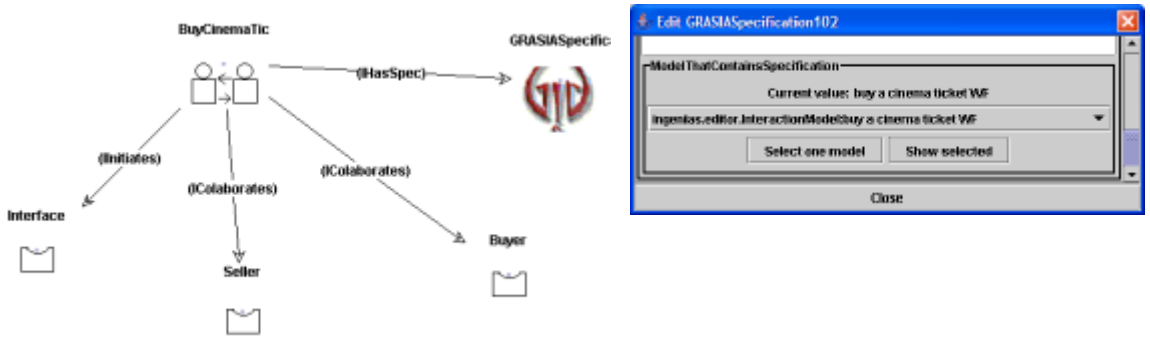


Figure 40. Interaction diagram defining the Interaction entity, interaction participants, and the specification entity, a GRASIA Specification in this case

- An organization diagram where agents are associated to the roles defined in the interaction

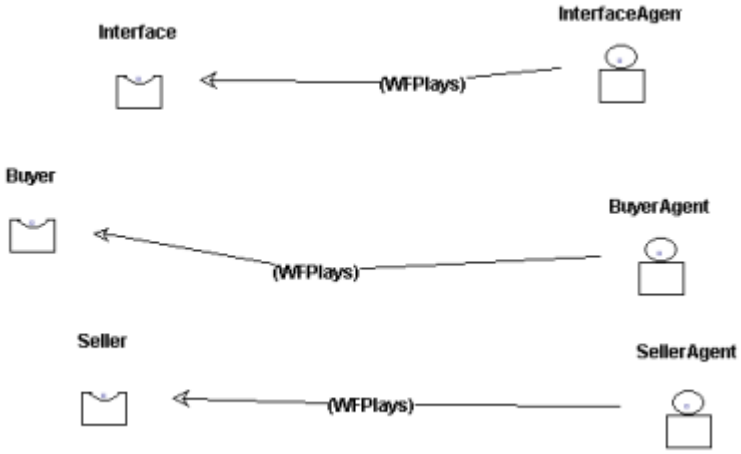


Figure 41. Assocation of roles participating into the Interaction with their agents

Once defined these diagrams, you can go to the module code generation option and select *generate code* as says Figure 42.

Figure 42. Code generation option for JADE platform

This will produce the source code to run the JADE agents. The next step is to compile the jade code:

```
ant compjademas
```

This will produce the required binaries to run the simulation. Then, to run the simulation, open two consoles in the same IDK distribution folder.

In the first one, type

```
ant runjade
```

This will start a JADE platform instance. Once launched, you will see a GUI, you have to type in the second console:

```
ant runjademas
```

This will open the gui like the one shown in Figure 43.

4.5.2.4 User manual

From the GUI shown in Figure 43, you can start protocols by clicking the buttons on the left part. On the right side you can check state machines allocated in the agents that are participating in the conversations. Each labelled square denote a conversation. Each white square inside the labeled square, represents a state machine instance. Above each white square, there is a label indicating which agent owns the state machine. Texts in red represent current states in each state machine.

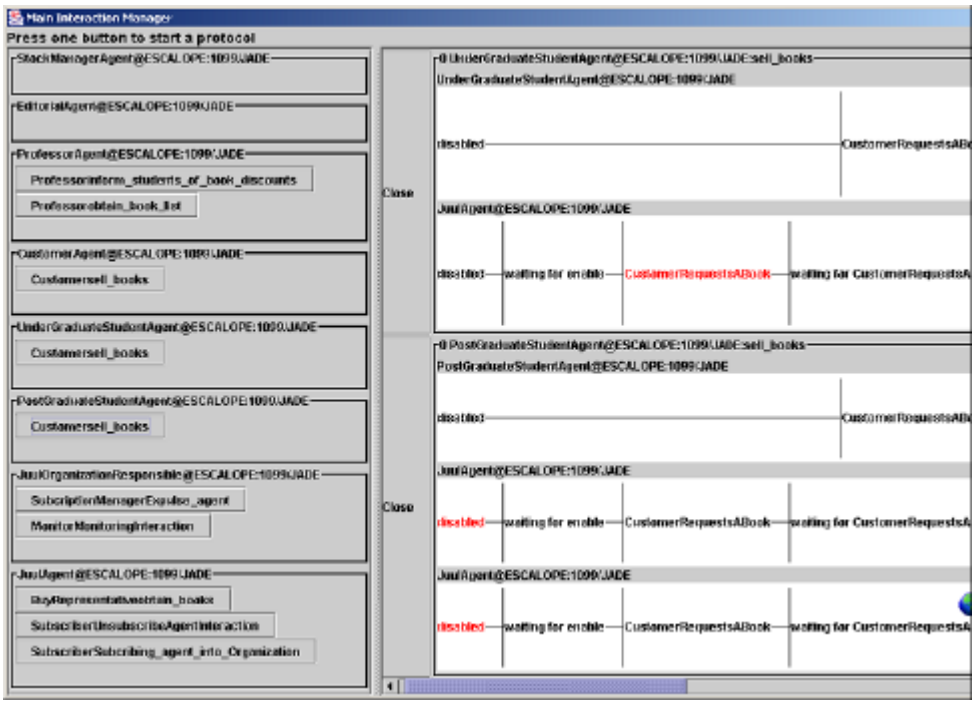


Figure 43. Snapshot of the main panel that starts individual agent protocols

You can hit any button as many times as you want. The agent will distinguish the different conversation instances of the same or different type.

During the execution, new state machines may appear and make the GUI size change. Size changes are not handled properly and you may have to resize a little the window to start the layout algorithm and size the frame properly.

Size changes happen because generated agents start protocol state machines under demand. So, you may start with one white panel and see how new panels are created without pressing buttons. The GUI does not close state machine protocol panels when one state machine finishes. To clean a state machine, press *close* button allocated right on the left of the affected state machine panel.

4.5.2.5 Development issues

A current drawback in the implementation is that alternatives in the protocol are not correctly implemented. Right now, if there is a branch in the protocol where the agent has to decide what to do next, by default it is coded that the agent will choose always the first option. However, code implementing other options is generated. Only the decision procedure is missing.

Each agent starts a new conversation only when requested. You can check by running the introspector of JADE and check how many behaviors exist. Also, you can run a sniffer and see what messages are passed.

An agent in this implementation can have many conversations at the same time. The conversation managemente facilities are builtin in the JADEAgent class and its inheritors.

What to do with this code? The system generates state machines that you can reuse for your system. To allocate the code to execute in each state, look at the if-thens inside each state machine specialization.

There are two templates in this module:

- **agent.xml.** This templates defines a JADEAgent. Every agent defined in the specification is a JADEAgent and knows as many protocols as the specification says. A JADE Agent contains three main behaviors that are responsible for deciding what protocol to enact according to non-processed messages.

- **ideaprot.xml.** This template defines the state machines that the code generator instantiates according to the role of each agent in a interaction. Each state machine has a controller that determines what is the next state and what message has to be sent.

The most complex part is the traversal of the diagrams to obtain the information required for the specification.

There are two other versions of this module:

- **JADE Leap module.** It is a variation of this module adapted to the JADE Leap platform
- **JADE Organization.** This one is a more complete version of the module where agent decision procedures are implemented and there are GUIs to inspect individual agent mental states, task schedulers, and other interesting features. However, it is not finished yet.

4.5.3 JADE Leap module

Developed by Carlos Celorrio (ccelorrio@telefonica.net) for his Ph.D. degree. He adapted the JADE module so that it could work with JADE Leap into a PDA. Main changes attained the conversion from J2SDK 1.4 to JDK 1.1, modifications of the templates to support task code execution on sending a message, and reconfiguration of the JADE module GUIs.

4.5.3.1 Output of the module

Agents that can run on a JADE Leap platform. A Main class that launches requested agents.

4.5.3.2 Requirements

JDK 1.1.8 (for JADE Leap to run) and J2SDK 1.4.2 (for IDK to run) installed . The JADE Leap platform installed.

4.5.3.3 Configuration

In order to run properly the ant task that compiles the code generated by the JADE Leap module, you must edit the leapBuild.xml file and edit the 'boot-java-1.1-classes' ant property so that it points to the *lib/classes.zip* allocated in the distribution folder for JDK1.1.8.

The Demo runs in a PC and a PDA. We assume that the PDA has already a JADE Leap distribution properly installed (please, consult the JADE Leap manual). The PC is a Windows based one, though could be any a unix based as well. The PDA in our case is a Sharp Zaurus

The steps to perform in the PC are:

- Open a command console and go to the IDK distribution folder
- Deploy the module with

```
ant modjadeleap
```

- Go to the IDK Editor with

```
ant runide
```

- Load an specification, for instance the ***cine.xml*** specification allocated in the examples folder of the IDK distribution.
- Go to modules → code generators → JADE LEAP agents generator → generate

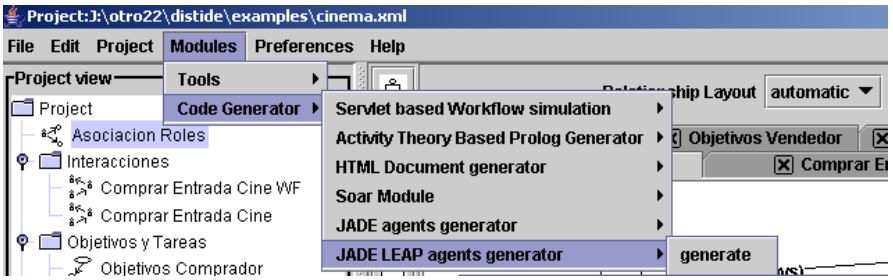


Figure 44. Code generation option for JADE Leap module

- Check that the IDK Editor says that some files were created and written

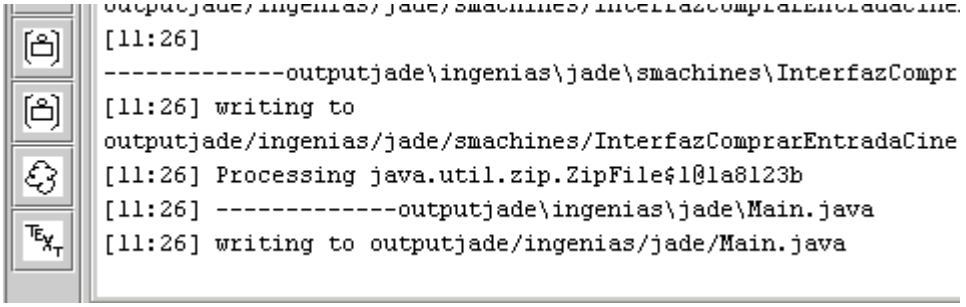


Figure 45. Produced messages on generating code for JADE Leap

- Close the IDK Editor
- Edit the *build.xml* file and set the property *boot-java-1.1-classes* to the *classes.zip* file allocated in the JDK1.1.8 distribution
- Compile the generated code with

```
ant compjadeleappjava
```

- Upload the generated jar ***genlib/demoPjava.jar*** into the PDA. This step depends on the software that allows to transfer files from the PC and the PDA. Sometimes, it is a desktop application that connects directly with the PDA, others, it is a matter of ftp-ing the file to the PDA or dragging and dropping it into a special *My PC* drive
- Start Jade Leap Main Container with

```
ant  runjadeleap
```

In the PDA (once the Main Jade Leap is running and the code of the agents code (the previously mentioned jar file) is uploaded into the PDA:

- Start Jade Leap Agents within the PDA. In the ZAURUS, this is done with the JVM implementation named *evm*

```
evm –cp demoPjava.jar:JadeLeap.jar ingenias.jade.Main –container –host 192.168.129.1
```

The "-host" parameter may be changed because it's the ip location of the Jade Leap Main Container (that must be running in the PC). Zaurus USB Lan uses "192.168.129.1" to identify the PC.

- Agent's GUIs should appear in the PDA, like the one from Figure 46

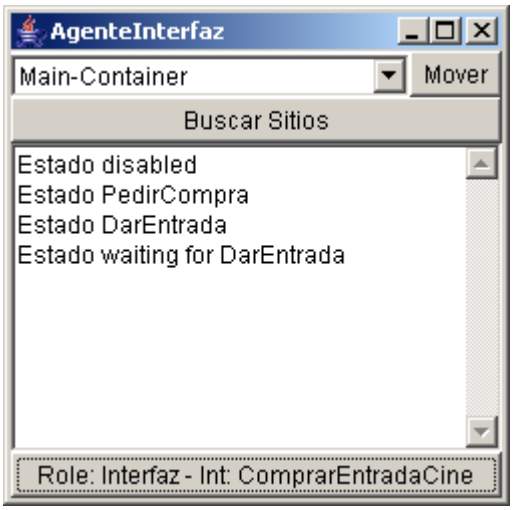


Figure 46. GUI representing an agent

- Move Vendor Agent to the PC by selecting 'Main-Container' in its GUI location selector and clicking 'Move'.
- To start the interaction click on the Interaction Button placed in the bottom of the Interface Agent's GUI.

4.5.3.4 User manual

When the agents are started as described above, you'll see a GUI window for each agent that is running. The agent GUI has three different zones:

- In the top of the window there is a selector to choose among the locations the mobile agent can go. Also, there is a button named 'Move' to go to the selected location.

- In the middle of the window there is a big text panel where it will be displayed the states that each agent transits.

- Every agent who does start an interaction -the interaction initiator- will have a button in the bottom of the window. When that button is clicked, the interaction will initiate. There will be a button for each interaction and it is possible for an agent to have more than one button.

The state machine that governs the behaviour of the agents is the same as in the Jade Module

4.5.3.5 Development issues

This module is based on the Ingenias' Jade Module. It has been adapted to compile with a JDK1.1 distribution, so that the code is suitable to be executed on every PC as well as on any PDA which complies with the Personal Java Profile.

The Swing GUI has been replaced with an AWT GUI, for the same reasons. The state of the agents is no more presented graphically in a single window, but in a textual way on the agent's GUI window.

On enabling mobility so that the agents could jump from one container to another, there were some problem serializing the agent's GUI. Because of this, it was decided to make it transient –not serializabled- and it is disposed and reconstructed every time the agent moves to a different location, so it is important that the JADEAgentGUI classes are in the classpath in the target move location. Nevertheless, the rest of the agent's classes, including its Jade behaviours and state machines, are correctly serialized and sent to the target machine.

4.5.4 JADE Organization module

Under development

4.5.4.1 Output of the module

Agents that can run on a JADE Leap platform. A Main class that launches requested agents.

4.5.4.2 Requirements

JDK 1.1.8 installed. The JADE Leap platform installed

4.5.4.3 Configuration

TODO

4.5.4.4 User manual

TODO

4.5.4.5 Development issues

TODO

4.5.5 SOAR module

Developed by Juan Antonio Recio (jareciog@ucmail.ucm.es) for his Ph.D. degree.

4.5.5.1 Output of the module

SOAR rule specification files detailing the behavior of a tank

4.5.5.2 Requirements

Having installed the soar distribution in your system. You can download SOAR 8.5.1 from

http://sitemaker.umich.edu/soar/soar_software_downloads

To run SOAR demos, you will need also the TCL/TK. For windows, the binary with the distribution can be downloaded from

http://prdownloads.sourceforge.net/tcl/tcl805.exe

You will need also the J2SDK 1.4.2

4.5.5.3 Configuration

To build the SOAR module, you should type, in a command console

```
ant modsoar
```

Then, after launching the editor with

```
ant runide
```

Now it is time to setup where you want your code to be allocated. SOAR Tank demo requires that customized tanks are allocated at C:\Program Files\Soar\Soar-Suite-8.5.1\tanksoar-3.0.7\agents. Therefore we set the `outputFolder property to` *C:\Soar\Soar-Suite-8.5.1\tanksoar-3.0.7\agents\soarAgent* as shows Figure 47. The project properties dialog window is opened with the option shown in Figure 8.

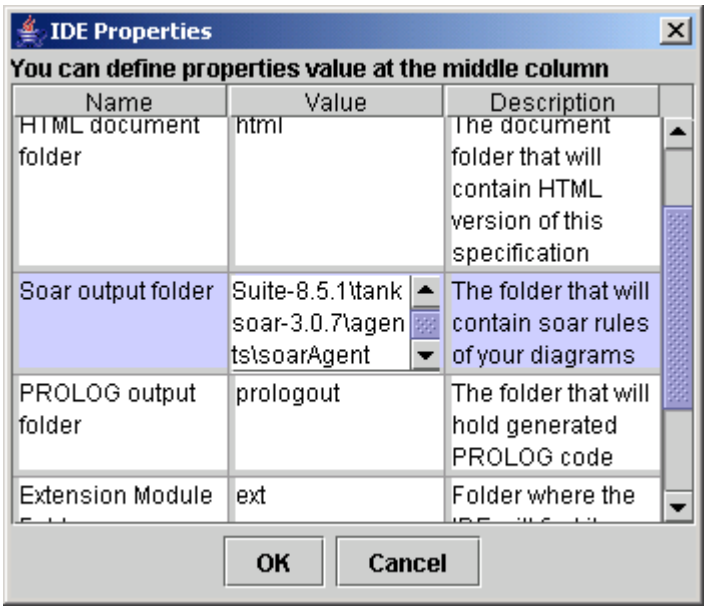


Figure 47. Project properties to modify the Soar output folder

Now, you should open the demo specification file, which is the *tanksoar.xml* file, allocated in the *examples* folder. Then, selecting the *generate* option of the soar module as shows the Figure 48 to generate code

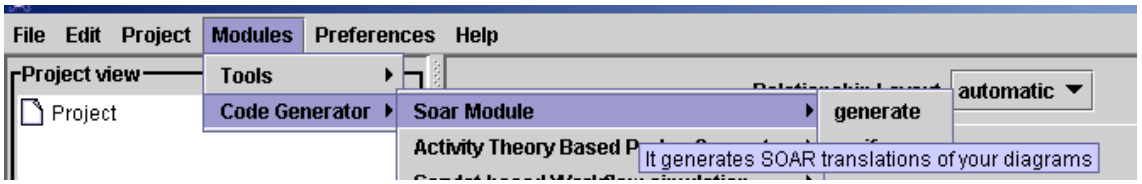


Figure 48. Code generation option for SOAR

The message panel should show something like the output of the Figure 49

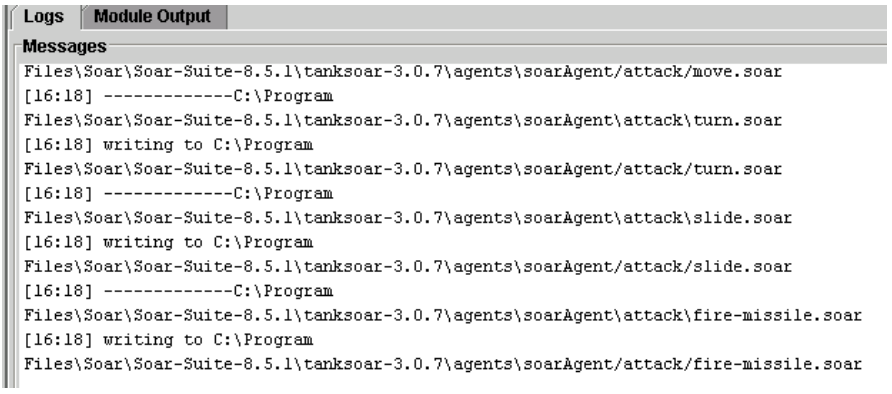


Figure 49. Output after selecting code generation option

Now you should open the SOAR by choosing Tank Soar from the Windows Start menu or starting the corresponding batch or shell file in other OS

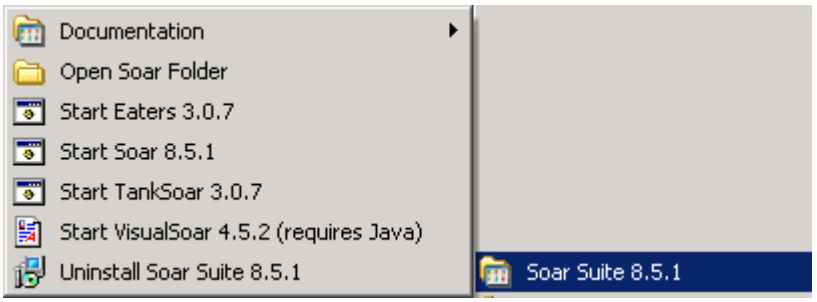


Figure 50. Starting the Tank SOAR demo

4.5.5.4 User manual

TODO

4.5.5.5 Development issues

TODO

4.5.6 Servlet based workflow simulator module

This servlet was created as a demonstration module for the juul bookhandle case study shown in next section.

4.5.6.1 Output of the module

This module produces a **servlet** that implements the execution workflow of tasks defined with the IDK. The module can be run withtin the IDK Editor or directly in a console.

4.5.6.2 Requirements

This implementation requires an application server that supports servlet execution. In this case, we used the resin application server (http://www.caucho.com). It is very easy to configure and launch, in any case we provide further instructions for this case study configuration.

The IDK includes the resin distribution and its associated licenses.

4.5.6.3 Configuration

Open the build.xml file included in the IDK distribution and change the property *resinfolder* so that it points at the home folder where the resin server is installed

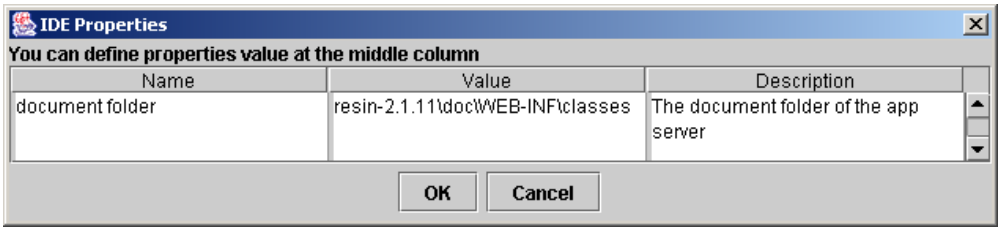
<!-- Change this property to the folder where resin is installed --> <property name="resinfolder" location="resin-2.1.11"/>

To compile the module, use the following command (windows)

```
ant modservlet
```

Once it is compiled, launch the editor, load the specification file (juul.xml in this case) and:

1. Open project properties (Project-> properties), and look for and set the property named "document folder". Set it to the *doc/WEB-INF/classes* folder under the resin home folder



2. go to the main menu selecting *Modules -> servlet based workflow simulation -> generate*

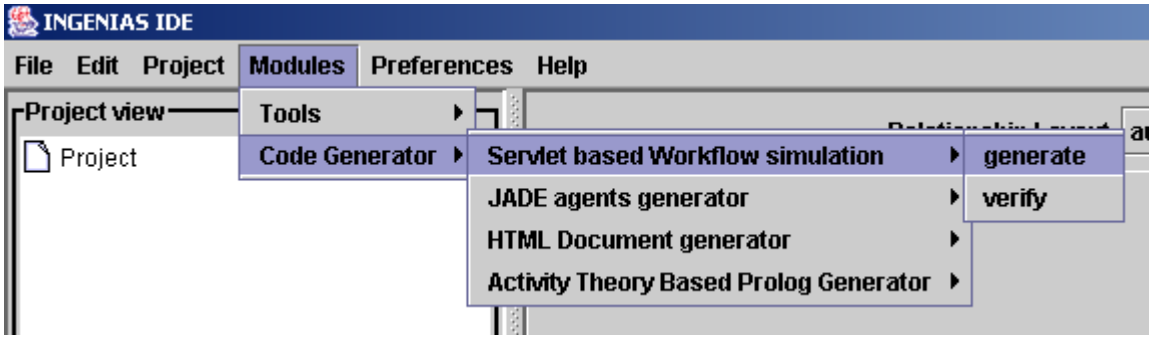


Figure 51. Invocation of the servlet generator

If everything goes as expected, it will start generating code of the servlet, note that the module output will show where it is writing files. If some part of the specification is not correct, it will show red messages showing the failures and what to do to solve them.

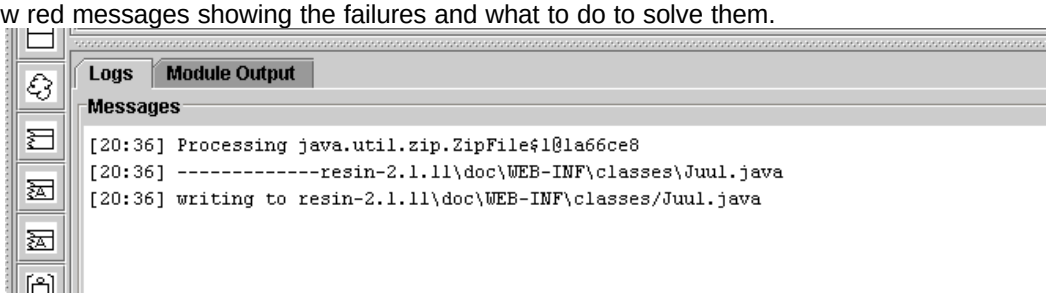


Figure 52. Confirmation message shown in the IDK Editor

The IDK distribution has a preconfigured resin application server. This server has a special file, named resin.conf, that is read by the app server to know what ports to use, where are the www document root, etc. To run the generated servlet, in the IDK we added the following lines to the resin.conf file (NO NEED TO DO THIS):

```
<web-app id='juul'>
<context-param info='An application information string'/>
<servlet servlet-name='juul' servlet-class='Juul'>
<init-param info='A servlet information string'/>
</servlet>
</web-app>
```

To start the web server, a specialised task has been defined in the build.xml included in IDK. To run this task from command line console the command should be (windows & unix)

```
ant launchhttpserver
```

It should output something like

```
C\foo\>ant launchhttpserver

Buildfile: build.xml

launchhttpserver:
[java]    Resin    2.1.11    (built    Mon    Sep    8    09:36:19    PDT    2003)
[java] Copyright(c) 1998-2003 Caucho Technology. All rights reserved.

[java]   Starting   Resin   on   Wed,   18   Feb   2004   19:37:44   +0100   (CET)
[java]       [2004-02-18       19:37:46.892]       initializing       application
http://localhost:8080/
[java]       [2004-02-18       19:37:46.892]       initializing       application
http://localhost:8080/java_tut
[java]       [2004-02-18       19:37:46.892]       initializing       application
http://localhost:8080/juul
[java]       [2004-02-18       19:37:46.892]       initializing       application
http://localhost:8080/examples/tags
[java]       [2004-02-18       19:37:46.892]       initializing       application
http://localhost:8080/examples/tictactoe
[java]       [2004-02-18       19:37:46.892]       initializing       application
http://localhost:8080/examples/navigation
[java]       [2004-02-18       19:37:46.972]       initializing       application
http://localhost:8080/examples/xsl
[java]       [2004-02-18       19:37:47.363]       initializing       application
http://localhost:8080/examples/templates
[java]       [2004-02-18       19:37:47.363]       initializing       application
http://localhost:8080/examples/login
[java]              http              listening              to              *:8080
[java] srun listening to 127.0.0.1:6802
```

Once launched, the server will be accesible from this URL

http://localhost:8080/servlet/Juul

4.5.6.4 User manual

Initially, only tasks that do not get inputs from other tasks will be shown as form buttons. By pressing on one of them, the servlet will check what tasks are enacted and will add new buttons to the form. Recently enacted tasks will be highlighted with an orange frame. Besides, the form will tell you what happened when you pressed the button the last time.

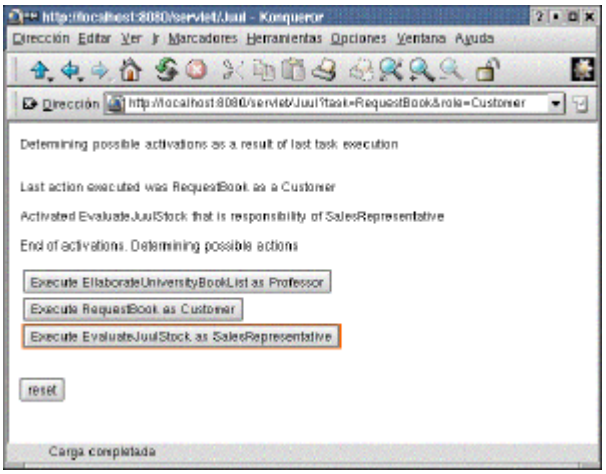


Figure 53. Navigator window showing selected tasks

Figure 53 shows a snapshot of the servlet. Each button represents a task that can be activated. It also has text presenting who is supposed to execute the task.

Be advised that some tasks enact other workflows. So it would not be surprising that when you thought that the workflow had finished, it turns out that the browser keeps on showing you new buttons.

4.5.6.5 Development issues

In this case, the target platform is a J2E Servlet. This servlet simulates a workflow execution by showing the user what tasks are available on executing a concrete one. Each task is associated to a concrete role, so the user has to be informed about who is supposed to be executing each task.

This is a code generation module, so we define the new class as an inheritor of *ingenias.editor.extension.BasicCodeGeneratorImp*. Taking the template as starting point, the new module had to generate a data structure that fitted into the repeat-var structure that was defined in the previous section. To do so, the program had to traverse the diagrams.

On traversing the diagrams, we realised some problems that did not appear in the initial prototype:

- **Determine which task goes first.** The program had to infer which tasks were the first that enabled other tasks. This was deduced by locating those tasks that did not take inputs from any task. As a consequence, there can be more than one initial task.
- **What if a task enacts two tasks?.** In this case, the program had to take into account that there can be multiple states in a determined moment. Information flow could select any path in current structure.

The source code for the servlet generator is here. To execute it, you will need the INGENIAS distribution. This code can be executed standalone or from within the IDK.

Diagram Entity	**Expected behavior and constraints**	**Target Platform representation**
Task	When a task is executed, it enacts other tasks in the same or other workflows. Every task has to have a role associated with relationships *WFResponsible*	Form buttons. Data structures in the servlet Strings
Role	The executor of a task. A role has to have at least one task assigned.	Just a string
WFConnects	It is a relationship that induces the workflow structure. This relationship means that the ouput of a task is connected to the inputs of another	if-then-else to codify state transition
WFResponsible	Associates roles and tasks	Implicit

Table 1. Mapping description to generate Servlets

5. CASE STUDIES

The IDK currently incorporates several cases of study. In this section we introduce briefly each one of them and what modules could be applied. In principle, any module can be applied to any specification. However, some of them are specially adapted to specific configurations.

5.1. COLLABORATIVE INFORMATION FILTERING

This case study describes how interact societies compound of two types of agents: Personal Agents and Community Agents. Personal Agents represent the interests of individuals in the system. Their purpose is to provide useful information to their users. Community Agents, on the other side, correspond to agents willing to group together Personal Agents with common interests and keep track of the evolution of the interest of the community.

This case study was performed following the INGENIAS methodology and recommended design steps. The documentation can be also reviewed in the INGENIAS web site (http://grasia.fdi.ucm.es)

The file containing the specification for the IDK is *colaborative filtering.xml*

5.1.1 Modules

This is a modelling case study only, but you can try to apply the following modules

- Servlet Based Workflow simulation. Described in section 4.5.6.
- JADE module. Described in 4.5.2.
- JADE Leap module. Described in 4.5.3.

5.2. BOOKSELLERS

We chose as case study the Juul Boklander case study written by Espen Andersen from http://www.aspen.com. The case study is a bookseller company that sells books to university students. The company has an agreement with professors and students to obtain a list of books used in their courses and sell them at special prices. The bookseller is considered an organization in which there are departments in charge of sales and departments in charge of logistics. Sales can be conventional or through internet. In this case,the goal is how to define electronic sales.

This case study was performed following the INGENIAS methodology and recommended design steps. Different steps are available:

- *juul-analysis-elab.xml*
- *juul-analysis-init.xml*
- *juul-design-elab.xml*

5.2.1 Modules

This case study provides several suitable files for the following modules:

- **Servlet Based Workflow simulation.** You can use as input for this module the following files:
  - *juul-design-elab-servlet.step1.xm*: example with failures
  - *juul-design-elab-servlet.step2.xm*: example with failures
  - *juul-design-elab-servlet.step3.xml: example operational.*
- **JADE module**. You can use as input for this module the following files

    - juul-design-elab-jade.step1.xml. example with failures
    - juul-design-elab-jade.step2.xml. example with failures
    - juul-design-elab-jade.step3.xml. *example operational.*
- **JADE Leap module.** You can use as input for this module the following files:
    - juul-interaction.xml. *example operational.*

5.3. QUAKE

This example was developed by: Guillermo Jimenez (gui_jim@hotmail.com). This is a modelling only case study where INGENIAS specification language shows its ability to deal with a complex environment such as the QUAKE game.

The example is based on a demo video of an implementation of quake bots made with PROLOG. The video is in Spanish and it is available at http://grasia.fdi.ucm.es/video

Current specification is on file *quake.xml*

5.3.1 Modules

This is a modelling case study only, but you can try to apply the following modules

- Servlet Based Workflow simulation. Described in section 4.5.6.
- JADE module. Described in 4.5.2.
- JADE Leap module. Described in 4.5.3.

5.4. ROBOCODE

This case study is not finished yet. The purpose of this case study is to translate ROBOCODE primitives to the IDK front end, and leave coding tasks apart. ROBOCODE is a game developed by IBM for educational purposes. In the game, you code the behaviour of a tank or a set of tanks in order to win battles.

This case study comes with an example of what would be the expected results of each stage of the INGENIAS methodology:

- *robocode-inception.xml*
- *robo-elaboration.xml*

5.4.1 Modules

- Robocode module. Described in section XXXX

5.5. SOAR

The case study developed by Juan Antonio Recio (jareciog@ucmail.ucm.es) for his Ph. D. degree. This case study consisted in adapting the Tank SOAR demo to INGENIAS. Adaptation process transformed the rule based behaviour that SOAR uses to diagrams that could be represented in INGENIAS. Basically, it expresses all a tank needs to know with two kind of diagrams: task and goal, and agent diagrams.

The specification file for soar is *tanksoar.xml*

5.5.1 Modules

- **Soar module.** This module has been described in Section 4.5.5.

6. INGENIAS

This section presents the INGENIAS methodology. INGENIAS have been developed taking MESSAGE [Caire 2001] as starting point. INGENIAS improves MESSAGE in several aspects:

- **Integration of design views of the system.** INGENIAS links concepts of different diagrams and allows mutual references.
- **Integration of research results.** Each meta-model has been elaborated attending to current research results in different areas like coordination, reasoning or workflows.
- **Integration of software development lifecycle.** The coupling of the UML Unified Process and INGENIAS is stronger now. We have defined concrete activities as well as how they would be distributed along the development cycle.
- **Support tools.** MESSAGE based in a commercial meta-case tool, named METAEDIT+. INGENIAS has its own java based tool, which is more portable and configurable.
- **Implementation concerns.** MESSAGE did not research how specifications could help in the implementation. INGENIAS is supported by a framework that allows to translate its diagrams into whatever programming language or text document.

At the moment, the full documentation of INGENIAS is available only in spanish in a ph.d. Any help in translating it to english will be welcome. Until a full version of INGENIAS is available in english, we provide a small summary of what is INGENIAS.

6.1. INTRODUCING INGENIAS

The development of Multi-Agent Systems (MAS) brings up new issues with respect to conventional software engineering practices, as it requires the integration of different concepts from the distributed artificial intelligence field, such as autonomy, agent mental state modelling, agent interactions and organization, or the definition of objectives and tasks assigned to agents in a MAS.

The purpose of INGENIAS is the definition of a methodology for the development of MAS, by integrating results from research in the area of agent technology with a well-established software development process, which in this case is the Rational Unified Process (RUP). This methodology is based on the definition of a set of meta-models that describe the elements that form a MAS from several viewpoints, and that allow to define a specification language for MAS. The specification of a MAS is structured in five viewpoints: the definition, control and management of each agent mental state, the agent interactions, the MAS organization, the environment, and the tasks and objectives assigned to each agent.

The integration of this MAS specification language with engineering practices is achieved by the definition of a set of activities that guide the engineering in the analysis and design phases, with the statement of the results that have to be produced from each activity. This process is supported by a set of tools, which are generated from the meta-models specification by means of a meta-modelling tool (MetaEdit+). MAS modelling is facilitated by a graphical editor and validation tool. As complement to this tool, there is a generic process for parameterization and instantiation of MAS frameworks, given a concrete MAS specification. The usability of this language and associated tools, and its integration with software engineering practices, have been validated in several examples from different domains, such as PC management, stock market, word-processor assistant, and specially the application to a collaborative filtering information system.

6.2. META-MODELLING

Though there may be previous interpretations of what meta-modeling is, in this document we attend to the definition provided in the Meta Object Facilities (MOF) [OMG 2000] specification of UML. This definition states that there are several levels in the definition of a language. In fact, it defines four levels where different language grammars are defined and each level defines the

grammar to be used in the next level. This process could be understood as a backwards stepwise abstraction from the information level. The process ends at the M1 level, which so far has proven to be enough to UML.

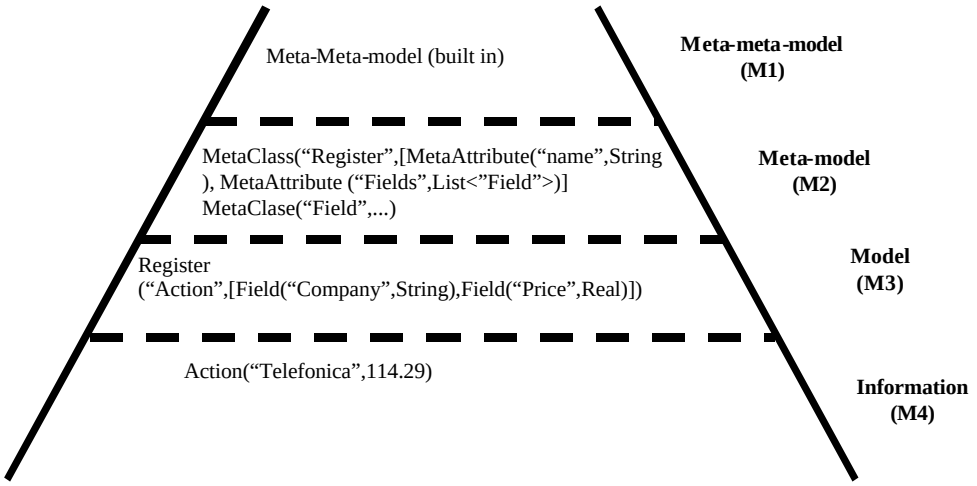


Figure 54. Levels when meta-modelling according to [OMG 2000]

In INGENIAS we use the schema of Figure 54 to structure de definition of the diagrams. However, we change the base M1 meta-meta-model and use a different one from MOF, which is the one chosen for defining UML. Instead, INGENIAS uses GOPRR [Lyytinen 1999] concepts which is simpler than MOF. In INGENIAS, after different experiences with MOF, we realized that most of the diagrams that we needed did not use most of the primitives of MOF, mainly because we were not defining an object oriented language, but an agent modelling language. In this sense, we have experienced that using entity-relationship diagrams is enough for defining INGENIAS diagrams. And a suitable language to define this kind of diagrams is GOPRR. GOPRR stands for Graph Object Property Relationship and Role, since these are the elements used to define any entity-relationship diagram. GOPRR seems to be enough to define UML diagrams. As a proof of that, METAEDIT+, a meta-case tool distributed by METACASE, implements all UML diagrams, except UML sequence diagrams.

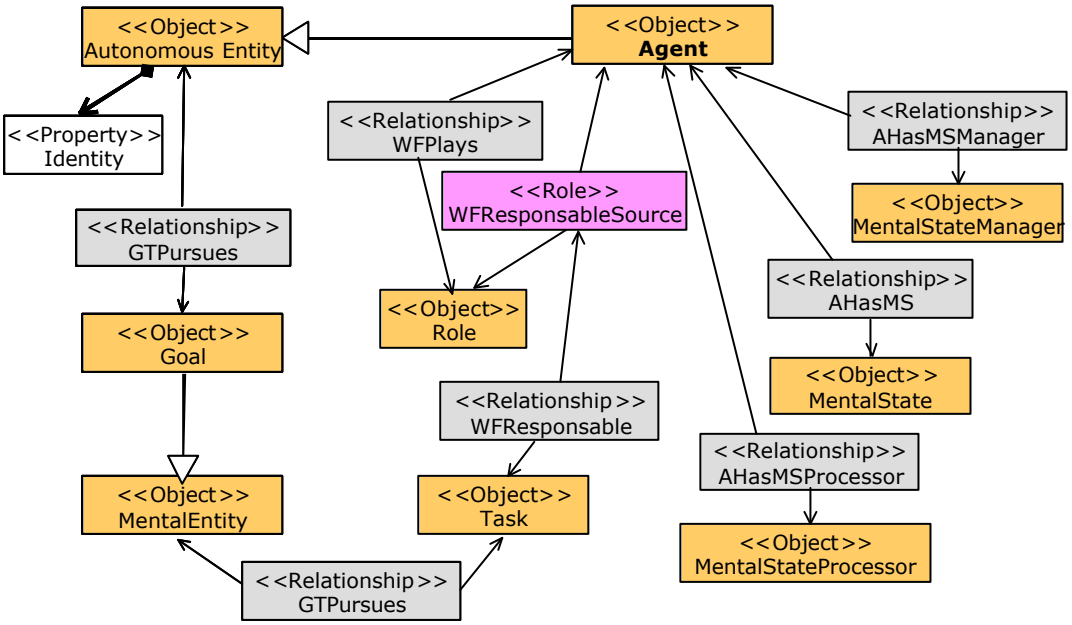


Figure 55. Example of meta-model defined with GOPRR

INGENIAS meta-models are defined in the M2 level. IDK implements M2 meta-models and is used to generate M3 models. Therefore, instances of these meta-models , according to Figure

56, are the concrete diagrams that the developer defines (level M3) with the IDK. There is an extra level, the M4, that is supposed to hold instances of M3 models. In INGENIAS we leave this instantiation to the developer. In our experience, models at the M3 level can be expressive enough to be used as if they were M4, but we do not. However, an M4 could be considered.

Figure 55 shows an example of a meta-model M2 which is part of the agent meta-model. It is represented using a UML class diagrams and stereotypes. GOPRR primitives appear as stereotypes of the different elements of the diagram. Basically, the diagram says that an *agent* is an *autonomous entity* that *pursues goals*. *Goals* are *mental entities* that form part of the *mental state* of the *agent*. An *agent* plays *roles* and, that way, it assumes responsibilities. An agent uses *tasks* to modify its mental state and the environment. These tasks are assigned to agents directly or through roles played. Changes in the Mental state are controlled using the *mental state manager*. This entity takes care of the consistency of the *mental state* and provides the primitives to change it. Decision procedures of the agent are built in the *mental state processor*.

6.3. INGENIAS META-MODELS

INGENIAS meta-models define five kinds of elements in order to define a MAS (see Figure 56).

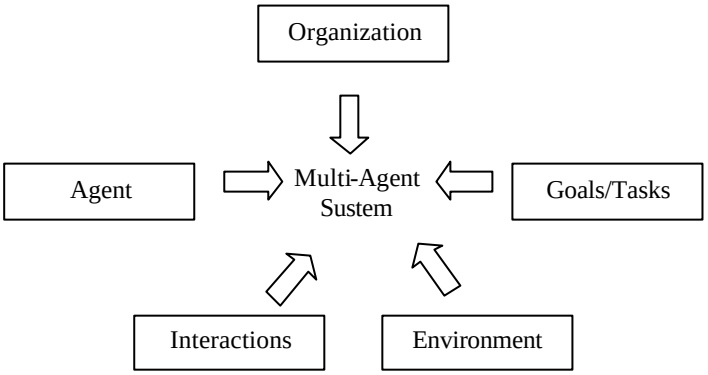


Figure 56. Elements required to define a MAS

So INGENIAS uses five meta-models that describes five types of diagrams of the same name. Entities of these meta-models, i.e. meta-entities, are not unique in the sense that anyone could be used in any of them. As a result, an entity, instance of an meta-entity, could appear in different diagrams.

- **Organization meta-model.** It defines organization diagrams. The organization is the equivalent of the MAS architecture. An organization has structure and functionality. The structure is similar to the one stated in AALAADIN framework [Ferber 1998] that later originated the MADKIT. As a developer, you define the organization attending to how you think your agents should be grouped. Functionality is determined when defining the goals of the organization and what workflows it should execute.
- **Environment meta-model.** It defines environment diagrams. The environment is what surrounds the MAS and what originates agent perception and action, mainly. As a developer, one of your first tasks is to identify system resources, applications, and agents. System resources are represented using TAEMS [Wagner 2001] notation. Applications are wrappers of whatever is not an agent or a resource, and could be understood as the equivalent of objects in INGENIAS. Using these elements, a developer should be able to define how the MAS interact with the system.
- **Task/Goal meta-model.** It describes how the mental state of agents change over the time, what is the consequence of executing a task with respect the mental state of an agent, how to achieve goals, and what happens when a goal cannot be achieved. It also gathers dependencies among different system or agent goals.

- **Agent meta-model.** It defines primitives to describe a single agent. It can be used to define the capabilities of an agent or its mental state. The mental state is an aggregate of mental entities that satisfy certain conditions. The initial or intermediate mental state is expressed in terms of mental entities such as those of AOP [Shoham 1993] and BDI [Kinny 1997].
- **Interaction meta-model.** It describe two or more agents interacting. The interaction itself is a first class citizen whose behavior is described using different languages, such as UML collaboration diagrams, GRASIA interaction diagrams, or AUML protocol diagrams. An interaction has a purpose that has to be shared or partially pursued by interaction participants. Usually it is related with some organizational goal.

An extensive detailed list of the diagrams that we support and the entities that we define, as well as relationships, can be found in the section 9. Icons and notation appears in section 8.

6.4. FAQ ABOUT DIAGRAMS

- What are the semantics of meta-models?
  - Semantics of the meta-entities defined with the meta-language GOPRR are rather naive, nothing further from what a relationship or an entity means. However, with respect MAS, something else could be said. In the original work of INGENIAS [Gomez-Sanz 2002] there is an deep explanation of the intended meaning of each element, though right now it is in plain natural language. We are planning to ellaborate something more formal, but there is much work to do.
- When to use them? It depends of what you intend to do. Here we provide some hints
  - Agent diagrams.
    - When you want to express an intermediate state of the agent. Figure 57 presents an example.

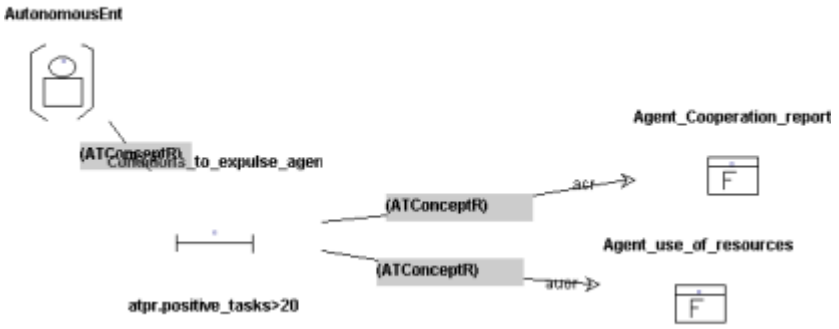


Figure 57. Mental state required to determine if an agent should be expulsed or not

    - When you need to express the initial setting of your agents
  - Interaction diagrams
    - How do I detail the interaction? You can associate different kinds of interaction specification to an interaction. Figure 58 shows an interaction that is associated with the three kinds of specification methods that the IDK supports at the moment

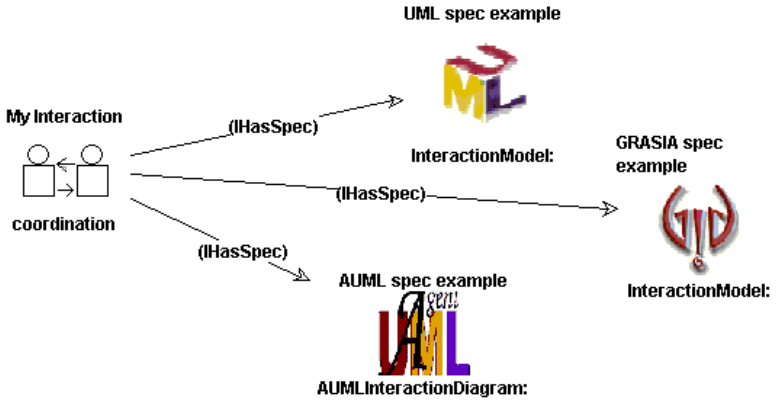


Figure 58. Interaction protocol specification by means of different specification mechanisms

7. CONCLUSIONS

The document is not finished yet and it won't be until we have completed INGENIAS. Developing a methodology is not easy at all, because it is not only about theorizing what could be useful to develop a system, but a matter of experimentation.

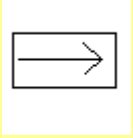
The document has presented key elements of INGENIAS and how to use them. We hope that this text has helped you, but in case it did not, feel free to contact us, mainly through the *ingenias.sourceforge.net* forums and mail lists. We only ask for a little patience since we have many responsibilities and dedicate INGENIAS as much as we can.

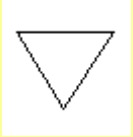
Also, if you felt that you could contribute to INGENIAS, you can contact us at same place or emailing jpavon@sip.ucm.es or jjgomez@sip.ucm.es.

8. ANNEX: NOTATION

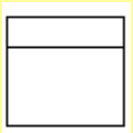
ID	PROLOGAgentDescription
	Provides an prolog based description of an agent. There is no syntax check here. This means that we assume that you write down prolog code. This kind of descriptions is used to represent conditions of goal satisfaction or failure, and interaction collaboration.
	8.1.1.1 When to use ... Initially use AgentDescription or NaturalLanguageAgentDescription. As the development progresses, move to GRASIAAgentDescription. Use this element when you expect to use PROLOG as implementation language for the control of your agent.

ID	AgentDescription
	Just indicates that in the future you should allocate here a more detailed description of conditions of goal satisfaction or failure, or interaction collaboration, or agent requirements.
	8.1.1.2 When to use ... Use this only in the initial stages of the development. As the development progresses, move to GRASIAAgentDescription and others more specific.

ID	ShareTouple
	Indicates that in the implementation a shared touple communication techonology will be used. Since interaction units transmit information from initiator to collaborators, this interaction unit will be translated as 'leave in a shared tuple space the information produced by this task and expect it to be read by collaborators'.
	8.1.1.3 When to use ... Use this interaction unit when you intend to use JavaSpaces or other similar techonologies to get in touch different agents

ID	Resource
	Describes a resource according to TAEMS notation. Opposite to TAEMS, there is no distinction between consumable and non-consumable resources.
	8.1.1.4 When to use ... Use this entity when you want to represent some non-functional requirements, like 'keep the bandwidth usage between this value and this', or 'do not use more than X percent of CPU time', or 'we require that at least X file descriptors be available'

ID	Slot
	It is a common slot used only in Frame Fact entity. Each slot could be understood as an attribute in an object.

	It is a common slot used only in Frame Fact entity. Each slot could be understood as an attribute in an object.

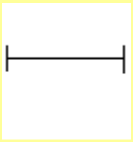
ID	IUConcurrence
	It is a container of interaction units. It represents a non-deterministic execution order of a set of interaction units. *8.1.1.6 When to use ...* You want to represent several possibly concurrent interaction units.

ID	MentalStateManager
	An agent has a mental state upon which the agent takes decisions. This mental state is an aggregate of mental entities (believes, facts, events, etc.). As an aggregate, the mental state must be managed. Say that management of mental state consists of determining how new mental entities are added, how to maintain consistency, and how to remove entities. To specify these elements you can use description field of the entity or also task and goal models. If you use task and goals models, you can detail which management tasks exist and how they act. In these diagrams, tasks can be associated to mental entities by other relationships different from consumes or produces. For instance, you can say that a task create mental entities or removes mental entities depending on certain conditions. To express these conditions, we use MentalStatePatterns. *8.1.1.7 When to use ...* Use this entity to associate mental state management concerns. In the initial stages, just provide informal specifications. In advanced stages, use task and goals diagrams.

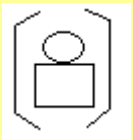
ID	EnvironmentApplication
	It represents and application that already exists in the environment that surrounds our MAS. An Application is a wrapper of an element that it is not an agent, nor a resource. You can configure methods in the application and relate this application with agents. Applications define agents perception. *8.1.1.8 When to use ...* Identify these entities early in the beginning of the development. They correspond to other systems that are already implemented and with which we will have to interact. The level of detail can be increased in further stages by providing accurate descriptions of the application's methods and events produced by the application

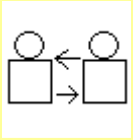
ID	TextNote
	A text note is simply an graphic object that shows text explaining details of a diagram.

	8.1.1.9 When to use ... Whenever you want to clarify some aspect of a diagram

ID	MentalState
	A mental state represents the mental state of an agent in a certain moment. A MentalState is an aggregate of mental entities. *8.1.1.10 When to use ...* Whenever you want to express what mental entities an agent is supposed to have at activation at the beginning. This is expressed in an agent model by associating an agent entity to a mental state. Whenever you want to express a intermediate state of the agent. You also use an agent model, but instead of an agent entity, you use an autonomous entity query. This entity represent an instance of an agent in runtime. You can express the type of the agent, or assume that its type is deduced from another diagram that contains a reference to this one. See Autonomous Entity Query.

ID	AgentModelBelieve
	A believe expressed with an agent model. This entity is an encapsulation for agent models taht express complex requirements *8.1.1.11 When to use ...* Whenever you want to express a believes about other agent abilities or mental state.

ID	AutonomousEntityQuery
	Represent an instance of an agent or a set of agents in runtime. We can refer to that running instance in different ways: with identifiers, with query-like expressions, etc. There are concretions of this entity to represent these different ways of referring to agents *8.1.1.12 When to use ...* Whenever you want to express a running instance of an agent without an special concern of the type of agent. Use this entity in the initial stages and move to more concrete entities as the development progresses.

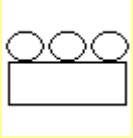
ID	Interaction
	Represents an interaction between two or more agents or roles. There can be only one initiator and at least one collaborator. An interaction also details the goal that pursues. This goal should be related with the goals of the participants. *8.1.1.13 When to use ...* Whenever you want to express interaction among agents. It can be used initially to just state that there is an interaction and that it has a purpose. In advanced stages, this entity can be further detailed by associating specification entities specialised in different notations, like UML-collaboration diagrams.

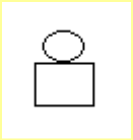
ID	MentalStatePattern
	It is a description of what mental state is required from an agent or set of agents. You can do this in many ways.
	8.1.1.14 When to use ... Whenever you want to express a mental state of an agent.

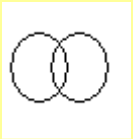
ID	ControlMentalEntity
	Represents an entity used in decision processes
	8.1.1.15 When to use ... Use this entity from general mental management processes

ID	SymbolicMentalStatePattern
	You express a mental state using common expressions. Depending on the language you use, you can decide that you need a more specific type of SymbolicMentalStatePattern
	8.1.1.16 When to use ... If you want to use logics to express a mental state or some agent programming language. Later on, you can substitute these units by more concrete entities.

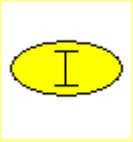
ID	GRASIAMentalStatePattern
	Describes an agent mental state using agent models. In these models you are expected to have only an instance of AutonomousEntityQuery associated with a mental state, and this mental state with required mental entities. Another alternative is to have a conditional mental state entity that allows to express conditions over identified mental entities
	8.1.1.17 When to use ... If you have not decided yet what implementation language to use, this approach is the perfect choice. Agent models can be translated with little effort to the implementation language you decide.

ID	Organization
	An organization is a set of agents, roles and resources that get together to achieve one or several goals. Inside an organization there are not other organizations, just groups. You can think of an organization as an enterprise. Internally it is composed by departments that may be restructured without affecting the external image of an enterprise.
	8.1.1.18 When to use ... Use organizations as higher grouping elements. Identify them soon in the beginning. Organizations may not be required if you have one or two agents. It is reccomendable to use them when the number of agents, roles or resources becomes difficult to handle. Refine organizations using groups.

ID	AgentComponent
	It represents any possible component of the agent. There is no compromise in the way this component will appear in a final agent architecture. There is a compromise in that the functionality determined by this component will appear in one or several elements of the architecture.
	8.1.1.19 When to use ... Use this entity to represent requirements in the architecture

ID	Compromise
	A compromise is an entity expressing that an agent has to execute a task due to a request performed by other agent
	8.1.1.20 When to use ... Whenever you want to register the need to execute tasks due to past requests of other agents

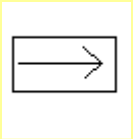
ID	TextUseCase
	A text use case is an UML use case whose description is supplied as natural language text
	8.1.1.21 When to use ... Use text use cases as a first refinement of conventional UML use cases. Then move to others like INGENIASUseCase

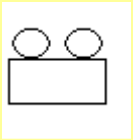
ID	INGENIASUseCase
	This use case is configured with information about preconditions and postconditions, as well as information of the different interactions that may appear
	8.1.1.22 When to use ... In later analysis to describe with detail what a use case consists of

ID	Role
	A role is a self-contained grouping of functionalities. When an agent plays a role we want to express that you have to to execute tasks associated to a role and participate in the same interactions that role.
	8.1.1.23 When to use ... Use roles when you find set of tasks that could be executed by different types of agents

ID	Goal

	According to the BDI model, a goal is a desired state that an agent wants to reach. In planning, a goal is represented by a world state. Here a goal is an entity by itself, however can be related with a representation of the world state using satisfaction relationships with tasks. This relationships contains references to descriptions of mental states of agents, so they refer to the image of the world that agent have
	8.1.1.24 When to use ... Use goals to represent states of the world to achieve. These states of the world can be represented by sets of mental entities associated to mental states. Tasks of the agents should provide media to achieve goals. Also take into account that goals may fail. There are also relationships that can represent this aspect

ID	RemoteProcedureCall
	This is an classic technology. It consists on invoking methods on objects allocated on other machines as if they were in the same local. It assumes that you have to produce an interface for the remote object.
	8.1.1.25 When to use ... This technology is one of the most widespread. Use it if you do not want specially FIPA-like communication facilities.

ID	OrganizationGroup
	A group contains other groups, roles, agents, applications, or resources. It represents the structure of an organization.
	8.1.1.26 When to use ... Whenever you want to structure the elements inside an organization. If you have only one group, perhaps you do not need any organization.Groups, and organizations as well, are useful when the developer foresees a high number of agents that may be working together. You can think of an OrganizationGroup as departments of an enterprise

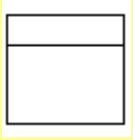
ID	MentalStateProcessor
	The agent takes decisions basing upon its mental state. There is an entity that supports management of the mental state of the agent (MentalStateManager) and this entity that represents the decision capabilities of the agent. As the MentalStateManager you can describe the MentalStateProcessor using tasks and goals diagrams
	8.1.1.27 When to use ... When you want to talk about decision procedures of the agent

ID	AOPMentalStatePattern
	The mental state of the agent is expressed the Agent0 language

	The mental state of the agent is expressed the Agent0 language
ID	Workflow
	A workflow is an abstraction to a process that has been automatised using activities and identifying their responsible. *8.1.1.29 When to use ...* Use workflows to represent tasks executed in common by different agents or roles. The workflow description can be complimented with interactions to determine in what moment a task executes.

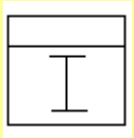
ID	MentalEntity
	It is any element that may form part of the mental state of an agent *8.1.1.30 When to use ...* Use mental entities as joker to define high level mental state management functions or decision procedures

ID	Believe
	A believe is a set of asserts that are not certainties, just expectations *8.1.1.31 When to use ...* Use it to define what you expect from other agents

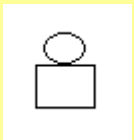
ID	FrameFact
	It is a fact whose informatino is contained in its slots *8.1.1.32 When to use ...* Use this entity in advanced development stages where you can provide types for the information collected.

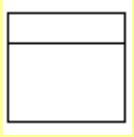
ID	Task
	Tasks is the encapsulation of actions or non-distributable algorithms. Tasks can use Applications and resources. Tasks generate changes in the mental state of the agent that executes them. Changes consist of: (a) modifying, creating or destroying mental entities; or (b) changes in the perception of the world by acting over applications (applications act over the world producing events, that are perceived by the agent). Though tasks can be also assigned to roles, at the end, it will belong to an agent.

	8.1.1.33 When to use ... Whenever you want to represent an activity or an ability associated to an agent or role. You can use them also to express activities performed in a organization. Finally, you can use them also to express management or decision taking activities.

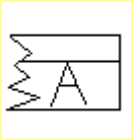
ID	InternalApplication
	It is an application that will be developed for this system. *8.1.1.34 When to use ...* You can use these entities as a representation of internal components of agents or components shared by several agents

ID	GRASIAAgentDescription
	Contains a reference to an agent model. With this model you describe conditions of goal satisfaction or failure, and interaction collaboration. Usually, this is achieved associating mental states to an instance of ConcreteAgent or AutonomousEntityQuery. This instance should be understood as the executor of the task or the performer or colaborator in an interaction unit. You also can use common associations in these diagrams to represent abilities required or other qualities. *8.1.1.35 When to use ...* Use GRASIAAgentDescription when you are in advanced development stages. These descriptions are very detailed and should provide enough information for most implementations.

ID	Agent
	An agent entity is an autonomous entity with identity, purposes and that performs activities to achieve its goals. *8.1.1.36 When to use ...* Whenever you find an entity that behaves according to Newell's definition of an agent: 'a program at the knowledge level that processes knowledge and behaves according with the rationality principle, i.e., that performs tasks only to achieve its goals'

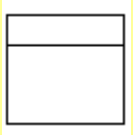
ID	Application
	An application is wraper to computational system entities. By 'computational', we mean 'having an interface and a concrete behavior'. *8.1.1.37 When to use ...* Whenever you have some system entity (physical) that you cannot categorize as an agent, organization, or resource. Use this entity when you do not know from the beginning further details of this entity.

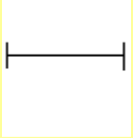
ID	ApplicationEvent

	It is an event produced by an application
	8.1.1.38 When to use ... Use it to define an agent's perception

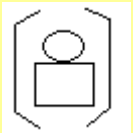
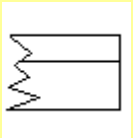
ID	GRASIASpecification
	A description of an interaction using GRASIA elements. This description allows to talk about the technology used to transfer information from one agent to another, refer to the mental conditions that must meet initator and collaborators at each step, what tasks will be executed and when, and what is the execution order of the different communication acts.
	8.1.1.39 When to use ... Use this kind of diagrams in design to generate an accurate description of what kind of interaction you wish. In the analysis just use UML Collaboration diagrams.

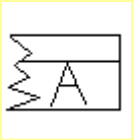
ID	InformationMentalEntity
	An entity that contains information about the world.
	8.1.1.40 When to use ... Use them to represent different aspects of the world's perception or agent state

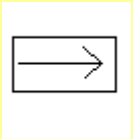
ID	IUIterate
	It is a container of interaction units. It represents a repetitive execution of a set of interaction units. The loop finishes when the next interaction unit, to which IUITerate is connected, appears, or when the number of iterations is satisfied. The number of iterations is a parameter.
	8.1.1.41 When to use ... You want to represent a loop with several interaction units. In the numberIterations field you can use notation from UML to represent the iteration

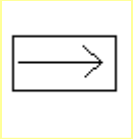
ID	ConditionalMentalState
	A mental state that adds extra information about what conditions must satisfy the entities aggregated in a mental state. Entities in a mental state can be labeled. These labels are used inside the mental state condition.
	8.1.1.42 When to use ... Use conditional mental states at the design stage to determine what you require from each mental entity at a certain moment

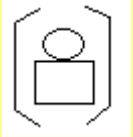
ID	AgentRequirementsQuery
	Describes a running instance of an agent by any kind of

	Describes a running instance of an agent by any kind of
ID	GeneralEvent
	
	8.1.1.44 When to use ...

ID	ApplicationEventSlots
	An application event that determines the information that transmits as slots
	8.1.1.45 When to use ... Use this in the early design to define exactly what information is required by tasks or goals (information from satisfaction conditions) to process them.

ID	MessagePassing
	
	8.1.1.46 When to use ...

ID	InteractionUnit
	
	8.1.1.47 When to use ...

ID	ConcreteAgent
	
	8.1.1.48 When to use ...

ID	NaturalLanguageAgentDescription
	Provides an natural language description of an agent. This kind of descriptions is used to represent conditions of goal satisfaction or failure, and interaction collaboration.

	Provides an natural language description of an agent. This kind of descriptions is used to represent conditions of goal satisfaction or failure, and interaction collaboration.

ID	Fact
	Describes an information that the agent accepts as reliable. This general entity contains this information in the description field. *8.1.1.50 When to use ...* Use them to express past experience of the agent, information passed from task to task, and other kind of information assumed as reliable by the agent.

ID	UMLSpecification
	8.1.1.51 When to use ...

9. ANNEX: META-MODEL SUMMARY

Entities

PROLOGAgentDescription
AgentDescription
ShareTouple
Resource
Slot
INGENIASObject
IUConcurrence
MentalStateManager
EnvironmentApplication
TextNote
MentalState
AgentModelBelieve
AutonomousEntityQuery
Interaction
MentalStatePattern
ControlMentalEntity
SymbolicMentalStatePatter
n
StateGoal
GRASIAMentalStatePattern
MethodParameter
Method
Organization
AgentComponent
Compromise
TextUseCase
INGENIASUseCase
Role
Goal
RemoteProcedureCall
OrganizationGroup
MentalStateProcessor
AOPMentalStatePattern
Workflow
MentalEntity
Believe
Autonomous_entity
FrameFact
Task
UseCase
InternalApplication
GRASIAAgentDescription
UMLObject
Agent
Application
ApplicationEvent
GRASIASpecification
Specification
InformationMentalEntity
IUIterate
ConditionalMentalState
AgentRequirementsQuery
GeneralEvent
ApplicationEventSlots

Relationships

AHasMSManager
AGOInconditionalSubordinationR
elationshipOrg
GTDecomposes
WFResponsible
AGOCondSubordinationRelations
hipGroup
OHasMember
EPerceivesPolling
WFParticipates
WFDecomposesWF
UIInitiates
ContributeNegatively
WFUsesMethod
UISelection
EPerceivesNotification
OHasWF
ParticipatesInUseCase
GTCreates
AGOInconditionalSubordinationR
elationshipGroup
WFProduces
GTPursues
ODecomposesWF
EPerceives
WFDecomposes
AGOClientServerRelationshipMe
mber
WFSpecifiesExecution
AInstanceOf
AGORelationshipGroup
AGORelationshipMember
AInherits
GTFails
AGOClientServerRelationshipOrg
AGOSubordinationRelationshipOr
g
GTDepends
Includes
IHasSpec
WFPursue
EResourceBelongsTo
WFConsumes
AHasMSProcessor
AGOSubordinationRelationshipGr
oup
AContainsME
GroupBelongsToOrganization
GTOrDepends
GTAndDepends
ODecomposesGroup
IInitiates
Contribute
AGORelationshipOrg

Diagrams

TasksAndGoalsModel
InteractionModel
UseCaseDiagram
AgentModel
EnvironmentModel
OrganizationModel

MessagePassing
InteractionUnit
ConcreteAgent
NaturalLanguageAgentDescription
Fact
UMLSpecification

AHasMS
UMLDescribesUseCase
IColaborates
GTAffects
UIPrecedes
ARoleInheritance
AGOInconditionalSubordinationRelationshipMember
UMLSendsMessage
UMLAssociation
OHasGroup
WFResponsable
WFContainsTask
WFConnects
Extends
GTInherits
UIColaborates
WFPlays
AGOClientServerRelationshipGroup
GTDestroys
IPursues
ApplicationBelongsTo
AGOSubordinationRelationshipMember
AGOCondSubordinationRelationshipOrg
GTSatisfies
AGOCondSubordinationRelationshipMember
WFUses
ContributePositively

9.1. DIAGRAMS

9.1.1 TasksAndGoalsModel

Entities:

- Agent
- Goal
- Task
- Resource
- Fact
- FrameFact
- Believe
- AgentModelBelieve
- Compromise
- GeneralEvent
- ApplicationEvent
- ApplicationEventSlots
- Application
- EnvironmentApplication
- InternalApplication
- TextNote

Relationships:

- GTPursues
- GTCreates
- GTAffects
- GTDestroys
- GTFails
- GTSatisfies
- WFConsumes
- WFUses
- WFUsesMethod
- WFProduces
- GTDecomposes
- GTDepends
- GTInherits
- GTAndDepends
- GTOrDepends
- WFDecomposes
- Contribute

- ContributePositively
- ContributeNegatively

9.1.2 InteractionModel

Entities:

- Agent
- Role
- Task
- Goal
- Interaction
- IUIterate
- IUConcurrence
- InteractionUnit
- ShareTouple
- RemoteProcedureCall
- MessagePassing
- UMLSpecification
- GRASIASpecification
- TextNote

Relationships:

- IInitiates
- IColaborates
- UIInitiates
- UIColaborates
- UISelection
- IPursues
- GTPursues
- IHasSpec
- UIPrecedes
- UMLSendsMessage

9.1.3 UseCaseDiagram

Entities:

- TextUseCase
- INGENIASUseCase
- Agent
- Role
- Interaction
- TextNote

Relationships:

- Extends
- Includes
- ParticipatesInUseCase
- UMLDescribesUseCase
- UMLAssociation

9.1.4 AgentModel

Entities:

- Agent
- Role
- Goal
- StateGoal
- Fact
- Task
- FrameFact
- Believe
- Compromise
- GeneralEvent
- ApplicationEvent
- ApplicationEventSlots

Relationships:

- GTPursues
- AInherits
- AHasMS
- AHasMSManager
- AHasMSProcessor
- WFResponsable
- AInstanceOf
- AContainsME
- WFPlays
- ARoleInheritance

- MentalStateProcessor
- MentalStateManager
- MentalState
- ConditionalMentalState
- AutonomousEntityQuery
- AgentRequirementsQuery
- ConcreteAgent
- AgentModelBelieve
- TextNote

9.1.5 EnvironmentModel

Entities:

- Agent
- OrganizationGroup
- Resource
- Application
- InternalApplication
- EnvironmentApplication
- TextNote

Relationships:

- EPerceives
- EPerceivesNotification
- EPerceivesPolling
- EResourceBelongsTo
- ApplicationBelongsTo

9.1.6 OrganizationModel

Entities:

- Agent
- Organization
- OrganizationGroup
- Role
- Resource
- Application
- EnvironmentApplication
- InternalApplication
- Task
- Workflow
- Interaction
- Goal
- Fact
- Believe
- GeneralEvent
- ApplicationEvent
- ApplicationEventSlots
- AutonomousEntityQuery
- AgentRequirementsQuery
- ConcreteAgent
- TextNote

Relationships:

- EResourceBelongsTo
- OHasGroup
- OHasMember
- OHasWF
- ODecomposesGroup
- ODecomposesWF
- GTPursues
- WFConnects
- WFUses
- WFContainsTask
- WFConsumes
- WFDecomposes
- WFProduces
- AGORelationshipGroup
- AGORelationshipMember
- AGORelationshipOrg
- AGOSubordinationRelationshipGroup
- AGOSubordinationRelationshipMember
- AGOSubordinationRelationshipOrg
- AGOCondSubordinationRelationshipGroup
- AGOCondSubordinationRelationshipMember
- AGOCondSubordinationRelationshipOrg
- AGOInconditionalSubordinationRelationshipGroup
- AGOInconditionalSubordinationRelationshipMember
- AGOInconditionalSubordinationRelationshipOrg
- AGOClientServerRelationshipGroup

- AGOClientServerRelationshipMember
- AGOClientServerRelationshipOrg
- WFSpecifiesExecution
- WFResponsable
- WFParticipates
- WFPlays
- WFDecomposesWF

9.2. ENTITIES

9.2.1 PROLOGAgentDescription specializes AgentDescription

Provides an prolog based description of an agent. There is no syntax check here. This means that we assume that you write down prolog code. This kind of descriptions is used to represent conditions of goal satisfaction or failure, and interaction collaboration.

Use it when ....

Initially use AgentDescription or NaturalLanguageAgentDescription. As the development progresses, move to GRASIAAgentDescription.
Use this element when you expect to use PROLOG as implementation language for the control of your agent.

Attributes:

- PROLOGDescription which contains java.lang.String

9.2.2 AgentDescription specializes INGENIASObject

Just indicates that in the future you should allocate here a more detailed description of conditions of goal satisfaction or failure, or interaction collaboration, or agent requirements.

Use it when ....

Use this only in the initial stages of the development. As the development progresses, move to GRASIAAgentDescription and others more specific.

Attributes:

9.2.3 ShareTouple specializes InteractionUnit

Indicates that in the implementation a shared touple communication techonology will be used. Since interaction units transmit information from initiator to collaborators, this interaction unit will be translated as 'leave in a shared tuple space the information produced by this task and expect it to be read by collaborators'.

Use it when ....

Use this interaction unit when you intend to use JavaSpaces or other similar techonologies to get in touch different agents

Attributes:

9.2.4 Resource specializes AgentComponent

Describes a resource according to TAEMS notation. Opposite to TAEMS, there is no distinction between consumable and non-consumable resources.

Use it when ....

Use this entity when you want to represent some non-functional requirements, like 'keep the bandwidth usage between this value and this', or 'do not use more than X percent of CPU time', or 'we require that at least X file descriptors be available'

Attributes:

- CurrentValue which contains java.lang.String
- MinValue which contains java.lang.String
- MaxValue which contains java.lang.String

9.2.5 Slot specializes INGENIASObject

It is a common slot used only in Frame Fact entity. Each slot could be understood as an attribute in an object.

Use it when ....

It can be used only in FrameFact entities

Attributes:

- Name which contains java.lang.String
- Type which contains java.lang.String
- Value which contains java.lang.String

9.2.6 INGENIASObject specializes Entity

Use it when ....

Attributes:

- Description which contains java.lang.String

9.2.7 IUConcurrence specializes InteractionUnit

It is a container of interaction units. It represents a non-deterministic execution order of a set of interaction units.

Use it when ....

You want to represent several possibly concurrent interaction units.

Attributes:

- **InteractionUnits** which is a collection of **ingenias.editor.entities.InteractionUnit** elements

9.2.8 MentalStateManager specializes AgentComponent

An agent has a mental state upon which the agent takes decisions. This mental state is an aggregate of mental entities (believes, facts, events, etc.). As an aggregate, the mental state must be managed. Say that management of mental state consists of determining how new mental entities are added, how to maintain consistency, and how to remove entities. To specify these elements you can use description field of the entity or also task and goal models. If you use task and goals models, you can detail which management tasks exist and how they act. In these diagrams, tasks can be associated to mental entities by other relationships different from consumes or produces. For instance, you can say that a task create mental entities or removes mental entities depending on certain conditions. To express these conditions, we use MentalStatePatterns.

Use it when ....

Use this entity to associate mental state management concerns. In the initial stages, just provide informal specifications. In advanced stages, use task and goals diagrams.

Attributes:

- **ManagerDescription** which is a collection of **ingenias.editor.entities.TasksAndGoalsModelModelEntity** elements

9.2.9 EnvironmentApplication specializes Application

It represents and application that already exists in the environment that surrounds our MAS. An Application is a wrapper of an element that it is not an agent, nor a resource. You can configure methods in the application and relate this application with agents. Applications define agents perception.

Use it when ....

Identify these entities early in the beginning of the development. They correspond to other systems that are already implemented and with which we will have to interact. The level of detail can be increased in further stages by providing accurate descriptions of the application's methods and events produced by the application

Attributes:

9.2.10 TextNote specializes UMLObject

A text note is simply an graphic object that shows text explaining details of a diagram.

Use it when ....

Whenever you want to clarify some aspect of a diagram

Attributes:

- Text which contains java.lang.String

9.2.11 MentalState specializes INGENIASObject

A mental state represents the mental state of an agent in a certain moment. A MentalState is an aggregate of mental entities.

Use it when ....

Whenever you want to express what mental entities an agent is supposed to have at activation at the beginning. This is expressed in an agent model by associating an agent entity to a mental state.
Whenever you want to express a intermediate state of the agent. You also use an agent model, but instead of an agent entity, you use an autonomous entity query. This entity represent an instance of an agent in runtime. You can express the type of the agent, or assume that its type is deduced from another diagram that contains a reference to this one. See Autonomous Entity Query.

Attributes:

9.2.12 AgentModelBelieve specializes Believe

A believe expressed with an agent model. This entity is an encapsulation for agent models taht express complex requirements

Use it when ....

Whenever you want to express a believes about other agent abilities or mental state.

Attributes:

- **Believed** which is a collection of **ingenias.editor.entities.AgentModelModelEntity** elements

9.2.13 AutonomousEntityQuery specializes INGENIASObject

Represent an instance of an agent or a set of agents in runtime. We can refer to that running instance in different ways: with identifiers, with query-like expressions, etc. There are concretions of this entity to represent these different ways of referring to agents

Use it when ....

Whenever you want to express a running instance of an agent without an special concern of the type of agent. Use this entity in the initial stages and move to more concrete entities as the development progresses.

9.2.14 Interaction specializes InteractionUnit

Represents an interaction between two or more agents or roles. There can be only one initiator and at least one collaborator. An interaction also details the goal that pursues. This goal should be related with the goals of the participants.

Use it when ....

Whenever you want to express interaction among agents. It can be used initially to just state that there is an interaction and that it has a purpose. In advanced stages, this entity can be further detailed by associating specification entities specialised in different notations, like UML-collaboration diagrams.

Attributes:

- Nature which contains java.lang.String

9.2.15 MentalStatePattern specializes INGENIASObject

It is a description of what mental state is required from an agent or set of agents. You can do this in many ways.

Use it when ....

Whenever you want to express a mental state of an agent.

9.2.16 ControlMentalEntity specializes MentalEntity

Represents an entity used in decision processes

Use it when ....

Use this entity from general mental management processes

9.2.17 SymbolicMentalStatePattern specializes MentalStatePattern

You express a mental state using common expressions. Depending on the language you use, you can decide that you need a more specific type of SymbolicMentalStatePattern

Use it when ....

If you want to use logics to express a mental state or some agent programming language.
Later on, you can substitute these units by more concrete entities.

9.2.18 StateGoal specializes Goal

This specialization of Goal is associated with a state that represents the lifecycle of a goal according to INGENIAS.

-Satisfied: means that this goal has been achieved
-Failed: means that this goal could not be achieved
-Pending: the goal has been activated and is waiting for possible refinements
-Refined: the goal has been refined into different subgoals. SubGoals are determined with
-Solving: there is a task that can satisfy this goal and it has been executed

Field Goal refers to the goal that is being described.

Use it when ....

This entity contains information about a goal in runtime so it is recommended to use it only to express required mental states of agents. Note that there is a reference to the original goal in case you need to represent different states of the same goal.

Use this entity when you plan to characterize goals with a state that fits into the lifecycle described in INGENIAS.

Attributes:

- State which contains java.lang.String
- LinkedGoal which contains ingenias.editor.entities.Goal

9.2.19 GRASIAMentalStatePattern specializes SymbolicMentalStatePattern

Describes an agent mental state using agent models. In these models you are expected to have only an instance of AutonomousEntityQuery associated with a mental state, and this mental state with required mental entities. Another alternative is to have a conditional mental state entity that allows to express conditions over identified mental entities

Use it when ....

If you have not decided yet what implementation language to use, this approach is the perfect choice. Agent models can be translated with little effort to the implementation language you decide.

Attributes:

- DescriptionWithAgentModel which contains ingenias.editor.entities.AgentModelModelEntity

9.2.20 MethodParameter specializes INGENIASObject

Use it when ....

Attributes:

- Name which contains java.lang.String
- Type which contains java.lang.String

9.2.21 Method specializes INGENIASObject

Use it when ....

Attributes:

- Result which contains java.lang.String
- Name which contains java.lang.String
- **Parameter** which is a collection of **ingenias.editor.entities.MethodParameter** elements

9.2.22 Organization specializes Autonomous_entity

An organization is a set of agents, roles and resources that get together to achieve one or several goals. Inside an organization there are not other organizations, just groups. You can think of an organization as an enterprise. Internally it is composed by departments that may be restructured without affecting the external image of an enterprise.

Use it when ....

Use organizations as higher grouping elements. Identify them soon in the beginning.
Organizations may not be required if you have one or two agents. It is reccomendable to use them when the number of agents, roles or resources becomes difficult to handle.
Refine organizations using groups.

Attributes:

9.2.23 AgentComponent specializes INGENIASObject

It represents any possible component of the agent. There is no compromise in the way this component will appear in a final agent architecture. There is a compromise in that the functionality determined by this component will appear in one or several elements of the architecture.

Use it when ....

Use this entity to represent requirements in the architecture

9.2.24 Compromise specializes ControlMentalEntity

A compromise is an entity expressing that an agent has to execute a task due to a request performed by other agent

Use it when ....

Whenever you want to register the need to execute tasks due to past requests of other agents

9.2.25 TextUseCase specializes UseCase

A text use case is an UML use case whose description is supplied as natural language text

Use it when ....

Use text use cases as a first refinement of conventional UML use cases. Then move to others like INGENIASUseCase

Attributes:

- Postcondition which contains java.lang.String
- Precondition which contains java.lang.String

9.2.26 INGENIASUseCase specializes UseCase

This use case is configured with information about preconditions and postconditions, as well as information of the different interactions that may appear

Use it when ....

In later analysis to describe with detail what a use case consists of

Attributes:

- **Postcondition** which is a collection of **ingenias.editor.entities.AgentModelModelEntity** elements
- **Scenarios** which is a collection of **ingenias.editor.entities.InteractionModelModelEntity** elements
- **Precondition** which is a collection of **ingenias.editor.entities.AgentModelModelEntity** elements

9.2.27 Role specializes INGENIASObject

A role is a self-contained grouping of functionalities. When an agent plays a role we want to express that you have to to execute tasks associated to a role and participate in the same interactions that role.

Use it when ....

Use roles when you find set of tasks that could be executed by different types of agents

Attributes:

9.2.28 Goal specializes ControlMentalEntity

According to the BDI model, a goal is a desired state that an agent wants to reach. In planning, a goal is represented by a world state. Here a goal is an entity by itself, however can be related with a representation of the world state using satisfaction relationships with tasks. This relationships contains references to descriptions of mental states of agents, so they refer to the image of the world that agent have

Use it when ....

Use goals to represent states of the world to achieve. These states of the world can be represented by sets of mental entities associated to mental states. Tasks of the agents should provide media to achieve goals. Also take into account that goals may fail. There are also relationships that can represent this aspect

Attributes:

9.2.29 RemoteProcedureCall specializes InteractionUnit

This is an classic technology. It consists on invoking methods on objects allocated on other machines as if they were in the same local. It assumes that you have to produce an interface for the remote object.

Use it when ....

This technology is one of the most widespread. Use it if you do not want specially FIPA-like communication facilities.

Attributes:

9.2.30 OrganizationGroup specializes INGENIASObject

A group contains other groups, roles, agents, applications, or resources. It represents the structure of an organization.

Use it when ....

Whenever you want to structure the elements inside an organization. If you have only one group, perhaps you do not need any organization.Groups, and organizations as well, are useful when the developer foresees a high number of agents that may be working together. You can think of an OrganizationGroup as departments of an enterprise

Attributes:

9.2.31 MentalStateProcessor specializes AgentComponent

The agent takes decisions basing upon its mental state. There is an entity that supports management of the mental state of the agent (MentalStateManager) and this entity that represents the decision capabilities of the agent. As the MentalStateManager you can describe the MentalStateProcessor using tasks and goals diagrams

Use it when ....

When you want to talk about decision procedures of the agent

Attributes:

- **ProcessorDescription** which is a collection of **ingenias.editor.entities.TasksAndGoalsModelModelEntity** elements

9.2.32 AOPMentalStatePattern specializes SymbolicMentalStatePattern

The mental state of the agent is expressed the Agent0 language

Use it when ....

When the final implementation language it is Agent0

Attributes:

- AOPExpression which contains java.lang.String

9.2.33 Workflow specializes INGENIASObject

A workflow is an abstraction to a process that has been automatised using activities and identifying their responsible.

Use it when ....

Use workflows to represent tasks executed in common by different agents or roles. The workflow description can be complimented with interactions to determine in what moment a task executes.

Attributes:

9.2.34 MentalEntity specializes INGENIASObject

It is any element that may form part of the mental state of an agent

Use it when ....

Use mental entities as joker to define high level mental state management functions or decision procedures

Attributes:

9.2.35 Believe specializes InformationMentalEntity

A believe is a set of asserts that are not certainties, just expectations

Use it when ....

Use it to define what you expect from other agents

Attributes:

9.2.36 Autonomous_entity specializes INGENIASObject

Use it when ....

Attributes:

9.2.37 FrameFact specializes Fact

It is a fact whose informatino is contained in its slots

Use it when ....

Use this entity in advanced development stages where you can provide types for the information collected.

Attributes:

- **Slots** which is a collection of **ingenias.editor.entities.Slot** elements

9.2.38 Task specializes AgentComponent

Tasks is the encapsulation of actions or non-distributable algorithms. Tasks can use Applications and resources. Tasks generate changes in the mental state of the agent that executes them. Changes consist of: (a) modifying, creating or destroying mental entities; or (b) changes in the perception of the world by acting over applications (applications act over the world producing events, that are perceived by the agent). Though tasks can be also assigned to roles, at the end, it will belong to an agent.

Use it when ....

Whenever you want to represent an activity or an ability associated to an agent or role. You can use them also to express activities performed in a organization. Finally, you can use them also to express management or decision taking activities.

Attributes:

9.2.39 UseCase specializes UMLObject

Describes a use case of the system.

Use it when ....

Attributes:

- Description which contains java.lang.String

9.2.40 InternalApplication specializes Application

It is an application that will be developed for this system.

Use it when ....

You can use these entities as a representation of internal components of agents or components shared by several agents

Attributes:

9.2.41 GRASIAAgentDescription specializes AgentDescription

Contains a reference to an agent model. With this model you describe conditions of goal satisfaction or failure, and interaction collaboration. Usually, this is achieved associating mental states to an instance of ConcreteAgent or AutonomousEntityQuery. This instance should be understood as the executor of the task or the performer or colaborator in an interaction unit. You also can use common associations in these diagrams to represent abilities required or other qualities.

Use it when ....

Use GRASIAAgentDescription when you are in advanced development stages. These descriptions are very detailed and should provide enough information for most implementations.

Attributes:

- DescriptionWithAgentModel which contains ingenias.editor.entities.AgentModelModelEntity

9.2.42 UMLObject specializes Entity

Use it when ....

Attributes:

9.2.43 Agent specializes Autonomous_entity

An agent entity is an autonomous entity with identity, purposes and that performs activities to achieve its goals.

Use it when ....

Whenever you find an entity that behaves according to Newell's definition of an agent: 'a program at the knowledge level that processes knowledge and behaves according with the rationality principle, i.e., that performs tasks only to achieve its goals'

Attributes:

9.2.44 Application specializes AgentComponent

An application is wraper to computational system entities. By 'computational', we mean 'having an interface and a concrete behavior'.

Use it when ....

Whenever you have some system entity (physical) that you cannot categorize as an agent,
organization, or resource. Use this entity when you do not know from the beginning further details of this entity.

Attributes:

- **Methods** which is a collection of **ingenias.editor.entities.Method** elements

9.2.45 ApplicationEvent specializes GeneralEvent

It is an event produced by an application

Use it when ....

Use it to define an agent's perception

Attributes:

9.2.46 GRASIASpecification specializes Specification

A description of an interaction using GRASIA elements. This description allows to talk about the technology used to transfer information from one agent to another, refer to the mental conditions that must meet initator and collaborators at each step, what tasks will be executed and when, and what is the execution order of the different communication acts.

Use it when ....

Use this kind of diagrams in design to generate an accurate description of what kind of interaction you wish. In the analysis just use UML Collaboration diagrams.

Attributes:

- ModelThatContainsSpecification which contains ingenias.editor.entities.InteractionModelModelEntity

9.2.47 Specification specializes INGENIASObject

Use it when ....

Attributes:

9.2.48 InformationMentalEntity specializes MentalEntity

An entity that contains information about the world.

Use **it** **when** **....**

Use them to represent different aspects of the world's perception or agent state

Attributes:

9.2.49 IUIterate specializes InteractionUnit

It is a container of interaction units. It represents a repetitive execution of a set of interaction units. The loop finishes when the next interaction unit, to which IUITerate is connected, appears, or when the number of iterations is satisfied. The number of iterations is a parameter.

Use **it** **when** **....**

You want to represent a loop with several interaction units. In the numberIterations field you can use notation from UML to represent the iteration

Attributes:

- NumberIterations which contains java.lang.String
- **InteractionUnits** which is a collection of **ingenias.editor.entities.InteractionUnit** elements

9.2.50 ConditionalMentalState specializes MentalState

A mental state that adds extra information about what conditions must satisfy the entities aggregated in a mental state. Entities in a mental state can be labeled. These labels are used inside the mental state condition.

Use **it** **when** **....**

Use conditional mental states at the design stage to determine what you require from each mental entity at a certain moment

Attributes:

- Condition which contains java.lang.String

9.2.51 AgentRequirementsQuery specializes AutonomousEntityQuery

Describes a running instance of an agent by any kind of

Use **it** **when** **....**

Attributes:

- Requirements which contains ingenias.editor.entities.AgentDescription

9.2.52 GeneralEvent specializes InformationMentalEntity

Use it when ....

Attributes:

- Source which contains ingenias.editor.entities.Application

9.2.53 ApplicationEventSlots specializes ApplicationEvent

An application event that determines the information that transmits as slots

Use it when ....

Use this in the early design to define exactly what information is required by tasks or goals (information from satisfaction conditions) to process them.

Attributes:

- **Slots** which is a collection of **ingenias.editor.entities.Slot** elements

9.2.54 MessagePassing specializes InteractionUnit

Use it when ....

Attributes:

9.2.55 InteractionUnit specializes INGENIASObject

Use it when ....

Attributes:

- SpeechAct which contains java.lang.String

9.2.56 ConcreteAgent specializes AutonomousEntityQuery

Use it when ....

Attributes:

9.2.57 NaturalLanguageAgentDescription specializes AgentDescription

Provides an natural language description of an agent. This kind of descriptions is used to represent conditions of goal satisfaction or failure, and interaction collaboration.

Use it when ....

Use this only in the initial stages of the development. As the development progresses, move to GRASIAAgentDescription and others more specific.

Attributes:

9.2.58 Fact specializes InformationMentalEntity

Describes an information that the agent accepts as reliable. This general entity contains this information in the description field.

Use it when ....

Use them to express past experience of the agent, information passed from task to task, and other kind of information assumed as reliable by the agent.

Attributes:

9.2.59 UMLSpecification specializes Specification

Use it when ....

Attributes:

- ModelThatContainsSpecification which contains ingenias.editor.entities.InteractionModelModelEntity

9.3. RELATIONSHIPS

9.3.1 AHasMSManager

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **AHasMSManagersource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Agent**
- Extreme: **AHasMSManagertarget** (min arity:1, max arity:1). It can contains entities of the following types:
  - **MentalStateManager**

9.3.2 AGOInconditionalSubordinationRelationshipOrg

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **AGORelationship1source** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Organization**
- Extreme: **AGORelationship1target** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Organization**

9.3.3 GTDecomposes

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **GTDecomposessource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Goal**
- Extreme: **GTDecomposestarget** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Goal**

9.3.4 WFResponsible

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **WFResponsiblesource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Agent**
  - **AutonomousEntityQuery**
  - **Role**
- Extreme: **WFResponsibletarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **Task**

9.3.5 AGOCondSubordinationRelationshipGroup

Attributes:

- Label which contains java.lang.String
- Condition which contains ingenias.editor.entities.MentalStatePattern

Connected elements:

- Extreme: **AGORelationship2source** (min arity:1, max arity:1). It can contains entities of the following types:
  - **OrganizationGroup**
- Extreme: **AGORelationship2target** (min arity:1, max arity:1). It can contains entities of the following types:
  - **OrganizationGroup**

9.3.6 OHasMember

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **OHasMembersource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **OrganizationGroup**
- Extreme: **OHasMembertarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **Agent**
  - **Role**
  - **Resource**

- **Application**

9.3.7 EPerceivesPolling

Attributes:

- Event which contains ingenias.editor.entities.GeneralEvent
- ApplicationMethod which contains java.lang.String
- Label which contains java.lang.String

Connected elements:

- Extreme: **EPerceivessource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Agent**
- Extreme: **EPerceivestarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **Application**

9.3.8 WFParticipates

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **WFParticipatessource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Agent**
  - **AutonomousEntityQuery**
  - **Role**
- Extreme: **WFParticipatestarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **Workflow**

9.3.9 WFDecomposesWF

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **WFDecomposesWFsource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Workflow**
- Extreme: **WFDecomposesWFtarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **Task**
  - **Workflow**

9.3.10 UIInitiates

Attributes:

- Label which contains java.lang.String
- Condition which contains ingenias.editor.entities.MentalStatePattern

Connected elements:

- Extreme: **UIInitiatessource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **InteractionUnit**
  - **Interaction**
- Extreme: **UIInitiatestarget** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Agent**
  - **Role**
- Extreme: **UIExecutestarget** (min arity:0, max arity:2147483647). It can contains entities of the following types:
  - **Task**

9.3.11 ContributeNegatively

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **Contributesource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Goal**
- Extreme: **Contributetarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **Goal**

9.3.12 WFUsesMethod

Attributes:

- ApplicationMethod which contains ingenias.editor.entities.Method
- Label which contains java.lang.String

Connected elements:

- Extreme: **WFUsessource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Task**
- Extreme: **WFUsestarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **Application**
  - **Resource**

9.3.13 UISelection

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **UISelectionsource** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **InteractionUnit**
  - **Interaction**
- Extreme: **UISelectiontarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **InteractionUnit**
  - **Interaction**

9.3.14 EPerceivesNotification

Attributes:

- Event which contains ingenias.editor.entities.GeneralEvent
- ApplicationMethod which contains java.lang.String
- Label which contains java.lang.String

Connected elements:

- Extreme: **EPerceivessource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Agent**
- Extreme: **EPerceivestarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **Application**

9.3.15 OHasWF

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **OHasWFsource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Organization**
- Extreme: **OHasWFtarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **Workflow**

9.3.16 ParticipatesInUseCase

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **ParticipatesInUseCasesource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Agent**
  - **Role**
- Extreme: **ParticipatesInUseCasetarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **UseCase**

9.3.17 GTCreates

Attributes:

- Label which contains java.lang.String
- Condition which contains ingenias.editor.entities.MentalStatePattern

Connected elements:

- Extreme: **GTCreatessource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Task**
- Extreme: **GTCreatestarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **MentalEntity**

9.3.18 AGOInconditionalSubordinationRelationshipGroup

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **AGORelationship2source** (min arity:1, max arity:1). It can contains entities of the following types:
  - **OrganizationGroup**
- Extreme: **AGORelationship2target** (min arity:1, max arity:1). It can contains entities of the following types:
  - **OrganizationGroup**

9.3.19 WFProduces

Attributes:

- Label which contains java.lang.String
- Condition which contains ingenias.editor.entities.MentalStatePattern

Connected elements:

- Extreme: **WFProducessource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Task**
- Extreme: **WFProducestarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **MentalEntity**
  - **Interaction**
  - **Resource**

9.3.20 GTPursues

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **GTPursuessource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Autonomous_entity**
  - **Role**
  - **AutonomousEntityQuery**
- Extreme: **GTPursuestarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **Goal**

9.3.21 ODecomposesWF

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **ODecomposesWFsource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Workflow**
- Extreme: **ODecomposesWFtarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **Workflow**

9.3.22 EPerceives

Attributes:

- Event which contains ingenias.editor.entities.GeneralEvent
- Label which contains java.lang.String

Connected elements:

- Extreme: **EPerceivessource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Agent**
- Extreme: **EPerceivestarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **Application**

9.3.23 WFDecomposes

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **WFDecomposessource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Task**
- Extreme: **WFDecomposestarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **Task**

9.3.24 AGOClientServerRelationshipMember

Attributes:

- OfferedGoalService which contains ingenias.editor.entities.Goal
- Label which contains java.lang.String

Connected elements:

- Extreme: **AGORelationship3source** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Agent**
  - **Role**
- Extreme: **AGORelationship3target** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Agent**
  - **Role**

9.3.25 WFSpecifiesExecution

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **WFSpecifiesExecutionsource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Task**
- Extreme: **WFSpecifiesExecutiontarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **Interaction**

9.3.26 AInstanceOf

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **AInstanceOfsource** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **Agent**
  - **Role**
- Extreme: **AInstanceOftarget** (min arity:1, max arity:1). It can contains entities of the following types:
  - **AutonomousEntityQuery**

9.3.27 AGORelationshipGroup

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **AGORelationship2source** (min arity:1, max arity:1). It can contains entities of the following types:
  - **OrganizationGroup**
- Extreme: **AGORelationship2target** (min arity:1, max arity:1). It can contains entities of the following types:
  - **OrganizationGroup**

9.3.28 AGORelationshipMember

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **AGORelationship3source** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Agent**
  - **Role**
- Extreme: **AGORelationship3target** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Agent**
  - **Role**

9.3.29 AInherits

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **AInheritssource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Agent**
- Extreme: **AInheritstarget** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Agent**

9.3.30 GTFails

Attributes:

- FailureCondition which contains ingenias.editor.entities.MentalStatePattern
- Label which contains java.lang.String

Connected elements:

- Extreme: **GTFailssource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Task**
- Extreme: **GTFailstarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **Goal**
- Extreme: **GTFailsHelperTasktarget** (min arity:0, max arity:1). It can contains entities of the following types:

- **Task**

9.3.31 AGOClientServerRelationshipOrg

Attributes:

- OfferedGoalService which contains ingenias.editor.entities.Goal
- Label which contains java.lang.String

Connected elements:

- Extreme: **AGORelationship1source** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Organization**
- Extreme: **AGORelationship1target** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Organization**

9.3.32 AGOSubordinationRelationshipOrg

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **AGORelationship1source** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Organization**
- Extreme: **AGORelationship1target** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Organization**

9.3.33 GTDepends

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **GTDependssource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Goal**
- Extreme: **GTDependstarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **Goal**

9.3.34 Includes

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **Includessource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **UseCase**
- Extreme: **Includestarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **UseCase**

9.3.35 IHasSpec

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **IHasSpecsource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Interaction**
- Extreme: **IHasSpectarget** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Specification**

9.3.36 WFPursue

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **WFPursuesource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Agent**
  - **Role**
- Extreme: **WFPursuetarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **Goal**

9.3.37 EResourceBelongsTo

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **EResourceBelongsTosource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **OrganizationGroup**
  - **Agent**
  - **Role**
- Extreme: **EResourceBelongsTotarget** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Resource**
  - **Application**

9.3.38 WFConsumes

Attributes:

- Label which contains java.lang.String
- Condition which contains ingenias.editor.entities.MentalStatePattern

Connected elements:

- Extreme: **WFConsumessource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Task**
  - **Workflow**
- Extreme: **WFConsumestarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **MentalEntity**

9.3.39 AHasMSProcessor

Attributes:

Connected elements:

- Extreme: **AHasMSProcessorsource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Agent**
- Extreme: **AHasMSProcessortarget** (min arity:1, max arity:1). It can contains entities of the following types:
  - **MentalStateProcessor**

9.3.40 AGOSubordinationRelationshipGroup

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **AGORelationship2source** (min arity:1, max arity:1). It can contains entities of the following types:
  - **OrganizationGroup**
- Extreme: **AGORelationship2target** (min arity:1, max arity:1). It can contains entities of the following types:
  - **OrganizationGroup**

9.3.41 AContainsME

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **AContainsMEsource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **MentalState**

- Extreme: **AContainsMEtarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
    - **MentalEntity**

9.3.42 GroupBelongsToOrganization

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **GroupBelongsToOrganizationO** (min arity:1, max arity:1). It can contains entities of the following types:
    - **Organization**
- Extreme: **GroupBelongsToOrganizationT** (min arity:1, max arity:2147483647). It can contains entities of the following types:
    - **OrganizationGroup**

9.3.43 GTOrDepends

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **GTDependssource** (min arity:1, max arity:1). It can contains entities of the following types:
    - **Goal**
- Extreme: **GTDependstarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
    - **Goal**

9.3.44 GTAndDepends

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **GTDependssource** (min arity:1, max arity:1). It can contains entities of the following types:
    - **Goal**
- Extreme: **GTDependstarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
    - **Goal**

9.3.45 ODecomposesGroup

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **ODecomposesGroupsource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **OrganizationGroup**
- Extreme: **ODecomposesGrouptarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **OrganizationGroup**

9.3.46 IInitiates

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **IInitiatessource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Interaction**
- Extreme: **IInitiatestarget** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Agent**
  - **Role**

9.3.47 Contribute

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **Contributesource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Goal**
- Extreme: **Contributetarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **Goal**

9.3.48 AGORelationshipOrg

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **AGORelationship1source** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Organization**
- Extreme: **AGORelationship1target** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Organization**

9.3.49 AHasMS

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **AHasMSsource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Agent**
  - **AutonomousEntityQuery**
- Extreme: **AHasMStarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **MentalState**

9.3.50 UMLDescribesUseCase

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **UMLDescribesUseCasesource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **UseCase**
- Extreme: **UMLDescribesUseCasetarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **Interaction**

9.3.51 IColaborates

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **IColaboratessource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Interaction**
- Extreme: **IColaboratestarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **Agent**
  - **Role**

9.3.52 GTAffects

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **GTAffectssource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Task**
- Extreme: **GTAffectstarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **MentalEntity**

9.3.53 UIPrecedes

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **UIPrecedessource** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **InteractionUnit**
  - **Interaction**
- Extreme: **UIPrecedestarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **InteractionUnit**
  - **Interaction**

9.3.54 ARoleInheritance

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **ARoleInheritancesource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Role**
- Extreme: **ARoleInheritancetarget** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Role**

9.3.55 AGOInconditionalSubordinationRelationshipMember

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **AGORelationship3source** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Agent**
  - **Role**
- Extreme: **AGORelationship3target** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Agent**
  - **Role**

9.3.56 UMLSendsMessage

Attributes:

- Order which contains java.lang.String
- Message which contains java.lang.String

Connected elements:

- Extreme: **UMLSendsMessagesource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Agent**
  - **Role**
- Extreme: **UMLSendsMessagetarget** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Agent**
  - **Role**

9.3.57 UMLAssociation

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **UMLAssociationsource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **UseCase**
- Extreme: **UMLAssociationtarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **UseCase**

9.3.58 OHasGroup

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **OHasGroupsource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Organization**
- Extreme: **OHasGrouptarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **OrganizationGroup**

9.3.59 WFResponsable

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **WFResponsablesource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Agent**
  - **Role**
  - **AutonomousEntityQuery**
- Extreme: **WFResponsabletarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **Task**

9.3.60 WFContainsTask

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **WFContainsTasksource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Workflow**
- Extreme: **WFContainsTasktarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **Task**
  - **Workflow**

9.3.61 WFConnects

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **WFConnectssource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Workflow**
  - **Task**
- Extreme: **WFConnectstarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **Workflow**
  - **Task**

9.3.62 Extends

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **Extendssource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **UseCase**
- Extreme: **Extendstarget** (min arity:1, max arity:1). It can contains entities of the following types:
  - **UseCase**

9.3.63 GTInherits

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **GTInheritssource** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **MentalEntity**
- Extreme: **GTInheritstarget** (min arity:1, max arity:1). It can contains entities of the following types:
  - **MentalEntity**

9.3.64 UIColaborates

Attributes:

- Label which contains java.lang.String
- Condition which contains ingenias.editor.entities.MentalStatePattern

Connected elements:

- Extreme: **UIColaboratessource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Interaction**
  - **InteractionUnit**
- Extreme: **UIColaboratestarget** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Agent**
  - **Role**
- Extreme: **UIExecutestarget** (min arity:0, max arity:2147483647). It can contains entities of the following types:
  - **Task**

9.3.65 WFPlays

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **WFPlayssource** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **Agent**
  - **AutonomousEntityQuery**
- Extreme: **WFPlaystarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **Role**

9.3.66 AGOClientServerRelationshipGroup

Attributes:

- OfferedGoalService which contains ingenias.editor.entities.Goal
- Label which contains java.lang.String

Connected elements:

- Extreme: **AGORelationship2source** (min arity:1, max arity:1). It can contains entities of the following types:
  - **OrganizationGroup**
- Extreme: **AGORelationship2target** (min arity:1, max arity:1). It can contains entities of the following types:
  - **OrganizationGroup**

9.3.67 GTDestroys

Attributes:

- Label which contains java.lang.String
- Condition which contains ingenias.editor.entities.MentalStatePattern

Connected elements:

- Extreme: **GTDestroyssource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Task**
- Extreme: **GTDestroystarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **MentalEntity**

9.3.68 IPursues

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **IPursuessource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Interaction**
- Extreme: **IPursuestarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **Goal**

9.3.69 ApplicationBelongsTo

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **ApplicationBelongsTosource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **OrganizationGroup**
  - **Agent**

    - **Role**
- Extreme: **ApplicationBelongsTotarget** (min arity:1, max arity:1). It can contains entities of the following types:
    - **Application**

9.3.70 AGOSubordinationRelationshipMember

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **AGORelationship3source** (min arity:1, max arity:1). It can contains entities of the following types:
    - **Agent**
    - **Role**
- Extreme: **AGORelationship3target** (min arity:1, max arity:1). It can contains entities of the following types:
    - **Agent**
    - **Role**

9.3.71 AGOCondSubordinationRelationshipOrg

Attributes:

- Label which contains java.lang.String
- Condition which contains ingenias.editor.entities.MentalStatePattern

Connected elements:

- Extreme: **AGORelationship1source** (min arity:1, max arity:1). It can contains entities of the following types:
    - **Organization**
- Extreme: **AGORelationship1target** (min arity:1, max arity:1). It can contains entities of the following types:
    - **Organization**

9.3.72 GTSatisfies

Attributes:

- SatisfactionCondition which contains ingenias.editor.entities.MentalStatePattern
- Label which contains java.lang.String

Connected elements:

- Extreme: **GTSatisfiessource** (min arity:1, max arity:1). It can contains entities of the following types:
    - **Task**
- Extreme: **GTSatisfiestarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
    - **Goal**

9.3.73 AGOCondSubordinationRelationshipMember

Attributes:

- Label which contains java.lang.String
- Condition which contains ingenias.editor.entities.MentalStatePattern

Connected elements:

- Extreme: **AGORelationship3source** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Agent**
  - **Role**
- Extreme: **AGORelationship3target** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Agent**
  - **Role**

9.3.74 WFUses

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **WFUsessource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Task**
- Extreme: **WFUsestarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **Application**
  - **Resource**

9.3.75 ContributePositively

Attributes:

- Label which contains java.lang.String

Connected elements:

- Extreme: **Contributesource** (min arity:1, max arity:1). It can contains entities of the following types:
  - **Goal**
- Extreme: **Contributetarget** (min arity:1, max arity:2147483647). It can contains entities of the following types:
  - **Goal**

REFERENCES

[Caire 2001] Caire, G., F. Leal, P. Chainho, R. Evans, F. Garijo, J. J. Gomez-Sanz, J. Pavon, P. Kerney, J. Stark and P. Massonet (2001). *Agent Oriented Analysis using MESSAGE/UML*, Springer Verlag.

[Ferber 1998] Ferber, J. and O. Gutknecht (1998). *A Meta-Model for the Analysis and Design of Organizations in Multi-Agent Systems*, Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS98), IEEE CS Press.

[Gamma 1995] Gamma, E., R. Helm, R. Johnson and J. Vlissides (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*.

[Gomez-Sanz 2002] Gomez-Sanz, J. J. (2002). *Modelado de Sistemas Multi-Agente*. Dpto. Sistemas Informáticos y Programación. Madrd, Universidad Complutense de Madrid.

[Kinny 1997] Kinny, D., M. Georgeff and A. Rao (1997). *A Methodology and Modelling Technique for Systems of BDI Agents*. Australian Artificial Intelligence Institute.

[Lyytinen 1999] Lyytinen, K. S. and M. Rossi (1999). *METAEDIT+ --- A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment*, Springer-Verlag.

[OMG 2000] OMG (2000). *MOF. Meta Object Facility (specification)*.

[Pavón 2003] Pavón, J. and J. Gómez-Sanz (2003). *Agent oriented software engineering with INGENIAS.* International Central and Eastern European Conference on Multi-Agent Systems, Springer Verlag.

[Shoham 1993] Shoham, Y. (1993). "*Agent Oriented Programming*." Artificial Intelligence **60**: 51-92.

[Wagner 2001] Wagner, T. and B. Horling (2001). *The Struggle for Reuse and Domain Independence: Research with TAEMS, DTC and JAF*, Proceedings of the 2nd Workshop on Infrastructure for Agents, MAS, and Scalable MAS.