TAOM4E: an Eclipse ready tool for Agent-Oriented Modeling. Issue on the development process.

Davide Bertolini, Aliaksei Novikau, Angelo Susi, and Anna Perini
ITC-irst, Via Sommarive, 18, I-38050 Trento-Povo, Italy
{bertolini,novikau,perini,susi}@itc.it

Abstract. Current Agent-Oriented Software Engineering (AOSE) methodologies adopt a model-based approach for analysis and design, but, in order to become of practical use, they should include it in a clear and customizable software development process and provide CASE tools that support it. In this technical report we describe the TAOM4E (Tool for Agent Oriented visual Modeling for the Eclipse platform) tool supporting the Tropos agent-oriented software engineering methodology. Tropos provides a modeling language based on a multi-agent paradigm; it supports analysis techniques and a structured modeling (design) process. The Tropos process covers whole software development cycle starting from the very first requirements. In developing the TAOM4E environment we are taking into account emerging guidelines and standards from the OMG’ Model-Driven Architecture (MDA) initiative which proposes an approach to software development based on modeling and automated mapping of models to code. The tool is based on the Eclipse Platform that offers a flexible solution to the problem of component integration. The motivations, the requirements and the architecture for the TAOM4E tool are described here in the very details. The report includes the current architectural diagrams and the working documents on the status of the development.

1 Premise

This technical report describes an ongoing work at ITC-irst (Distributed Intelligence research line, at SRA Division) on the development of a software environment for supporting the use and the experimentation of the Tropos methodology, an agent oriented software engineering methodology that was initially proposed in 2001 [26, 9, 6]. The environment, named Tool for Agent Oriented Modeling (TAOM), has been initially conceived as a modeler, supporting agent-oriented modeling in Tropos [5]. A motivation for this effort came from practitioners who tend to ask supporting tools and guidelines when applying a new methodology. Further motivations came together with new research line and projects conducted at ITC-irst, such as the I2F framework aiming at a light integration of model checking techniques for the validation of portions of a model [27], the ASTRO project ¹ aiming

¹ ASTRO is a research project in the field of web services and service-oriented applications, with a focus on the integration of business processes that are distributed among the most disparate entities, http://www.astroproject.org/. The project is funded by MIUR.
at developing new design methods and tools for web-services, adopting a Model Driven Architecture approach [30] and the STAMPS project ² which extends the Astro project objectives [29].

Currently, the development and extension of the TAOM environment are driven by research goals on refining the process supported by Tropos and on customizing it to the design of web services.

A first prototype of the TAOM tool [30] has been developed and distributed under GPL 2 license. The tool has been re-engineered on the ECLIPSE platform (TAOM4E) and it is currently distributed under the GPL 2 license (http://www.gnu.org/copyleft/gpl.html) [35].

The main purpose of this report is to describe basic issues that have been addressed during the development of the TAOM4E environment and to collect information that can be useful when extending and maintaining the tool. Related research results have been published in scientific papers that will be referred in this report.

Besides the authors of this report, several students contributed in different ways to the development of the TAOM tool, they are: Michela Strobbe, Nicola Villa, Antonella Pojer and Nicola Fazzi.

The document is structured as follows. In section 2 we briefly recall the Tropos methodology and the development process it supports, basic ideas from the Model Driven Architecture initiative by OMG and related work. In section 3 we discuss basic requirements for the Tropos environment. In section 4 we illustrate the tool architecture. In section 5 we describe available documentation. In section 7 we will discuss future work.

2 Background

2.1 The Tropos methodology

Tropos [7,9] is an agent-oriented software engineering methodology which provides: a modeling language based on a multi-agent paradigm and which supports analysis techniques; a structured modeling (design) process; a set of tools which support the use of the analysis techniques and of the process itself ³. The methodology consists of five phases, each characterized by a specific objectives, namely: Early Requirement, whose objective is to understand the domain with its stakeholders and their individual and shared goals; Late Requirements focuses on the elicitation of the requirements of the system-to-be; Architectural Design whose objective is to specify the system architecture in terms of a set of interacting software agents; Detailed Design, which is concerned with the specification of agent capabilities and interaction; Implementation, whose objective is the production of code from the detailed design specification. The Tropos modeling language rests on the i* framework [39]. It allows the representation of intentional and social concepts, such as actor, goal and softgoal, plan, resource, and

² The name stands for Software Methodology and Technology for Peer-to-Peer Systems. The project is funded by PAT.

³ A large international group is contributing to the extension of Tropos, as testified by the web site http://www.troposproject.org which provides useful resources.
a set of relationships between them, such as actor dependency, goal or plan decomposition, means-end and contribution relationships. The modeling process starts with the identification of critical actors in a domain along with their goals, and proceeds with the analysis of goals on behalf of each individual actor. In particular, given a root goal of an actor, the software engineers may decide to delegate it to an actor already existing in the domain or to a new actor. Such delegations result in a network of relationships among actors in the domain. Moreover the software engineer may decide to analyze a goal producing a set of subgoals. Goal analysis generates a goal hierarchy where the leaves in various combinations represent concrete solutions to the root goal. Finally the software engineer may decide that a certain actor is able to satisfy the goal via a plan the actor is able to execute; in this case the goal is assigned to that actor (with no further delegations). The process is complete when all goals have been dealt with. In [7] this process is described in terms of a non-deterministic concurrent algorithm. Figure 1 gives an intuitive ideas on how a Tropos model can evolve from early requirements to detailed design.

![Fig. 1. Tropos: software development phases.](image)

### 2.2 Model Driven Architecture

In developing the TAOM environment we are taking into account emerging guidelines and standards from the OMG’ Model-Driven Architecture (MDA) initiative which proposes an approach to software development based on modeling and automated mapping
of models to code [8]. A basic motivation of MDA is that of improving quality, by allowing for the reuse of models and mappings between models, and software maintainability by favoring a better consistency between models and code. One of the basic concepts in MDA is that of distinguishing between a software design which is platform independent (Platform-Independent Models, PIM) from a software design that includes all the platform specific details (Platform-Specific Models, PSM). The two models can be related through a transformation process which converts a PIM to its semantically equivalent PSM. A PIM model can be the result of a chain of transformations between different abstraction level PIMs. In MDA, the use of various modeling concepts and notations is foreseen with the idea to favor the exploitation of existing specification languages that are more appropriate to define views on dynamic aspects rather than of structural ones of a given model.

From a practical point of view, the MDA initiative is proposing a standard to which the meta-models of the specification languages used in the modeling process must be compliant with, that is the Meta Object Facility (MOF) [24], and a set of requirements for the transformation techniques that will be applied when transforming a source model into a target model, this is referred as the Query/View/Transformation (QVT) approach [16]. The MOF version which is currently available is the 1.4 which the Tropos modeling language meta-model is compliant to.

For what concerns QVT, on one side OMG is working on the specification of the MOF 2.0 QVT requirements, and on the other side several techniques for model transformation have been proposed.

### 2.3 Related Work

Many Agent-Oriented Software methodologies have been proposed over the last few years [25, 17, 18, 38, 11] and work were they are compared with other to point out differences and complementarities are enriching the AOSE literature. For instance, an evaluation of Tropos with respect to other methodologies can be found in [19, 31], both analysis have been conducted adopting a feature based approach and propose evaluation criteria at support of the choice of the most appropriate methodology to be adopted for a particular application.

Focusing on methodology metamodels, an analysis of their characteristics at the agent and at the system level has been presented in [4] considering ADELFE [3], GAIA [40] and PASSI [2]. The aim of that work was to face interoperability issues between different methodologies.

In [32] we can find an extension of this comparison to Tropos. This comparison points out the fact that different metamodels (methodologies) may allow to model different properties of a system (e.g organizational aspects, communications and protocols), so in some cases it could be appropriate to use different metamodels, provided that an effective mapping between the common concepts is given. On the other side, it shows that even if metamodels share a comparable set of concepts, they can be used in a different way by the different methodologies.

As mentioned above, in developing the TAOM environment we adopt the OMG’s MDA approach to automatically transform source to target models which refer to different meta-models [28].
3 Tool Requirements

A core set of requirements of an environment at support of the use of the Tropos methodology have been presented in [5], where the basic objective was to develop an AO modeler. As stated in section 1, further requirements have been motivated by research ideas emerging in following projects (ASTRO, STAMPS).

This section, describe the set of requirements that have been implemented in the current version of the TAOM4E environment. They have been organized in functional and nonfunctional requirements, and described in the following subsections.

3.1 Functional requirements

The functional requirements are represented in the Figure 2 in an usecase diagram. All the usecases are correspondent to the following functional requirements:

1 Changing the Tropos metamodel in order to support several Tropos dialects. This requirement includes:
a) New nodes or relations can be added or removed from the metamodel.
b) Different notations can be added to existed nodes and relations. An example of the notations may be Formal Tropos. Formal Tropos specifications can be added to the Informal Tropos model as text.

2 One of the main functional requirements for the tool is to provide the analyst with model’s editing facilities like:
   a) Construct model with a visual editor.
   b) Query model on subject of some properties, for example number of some elements or consistency.
   c) Have different views on the model. The views can be of different type to represent several properties of the model. They may contain the part of the model in order to be more understandable.
   d) The model should be able to be loaded and saved.

3 The Tropos process should be supported. According to this the following requirements should be satisfied:
   a) The tool should support the Tropos phases. All the diagrams and activities should be separated according to the phases.
   b) The analyst should be guided on the Tropos process. There should be some hints and proper analysis choices during the development.

4 In order to support integration of other tools, TAOM4E should give the possibility of translating its own model to other languages like UML. This includes the following requirements:
   a) Have convenient format of the processed model.
   b) Have XML based of the saved model.

3.2 Nonfunctional requirements

We think that usability, extensibility and flexibility nonfunctional requirements are the most essential for our tool while nonfunctional requirements like size, performance and reliability are not as important for visual modelling environments. Some nonfunctional requirements are correlated with the functional requirements described in the Section 3.1. The three categories of nonfunctional requirements under our interest are described above:

1 **Usability.** The requirements include all the things which make convenient visual modelling process like:
   a) Have the palette to choose the objects drawn in the editor.
   b) Delete/add objects in the views and in the model.
   c) Reconnect links from one object to others.
   d) Different colors for the objects in the editor. Have the possibility to define custom color palettes.
   e) Have the outline tree of the views and of the model.
   f) Drag and drop objects from the outline tree to the editor.
   g) Undo-redo of all the editing actions.
   h) Copy and paste parts of diagrams.
   i) Changing the types of the objects in the editor (like hardgoal to softgoal).
j) Export the diagrams in the picture, print the diagrams.

2 **Flexibility.** This nonfunctional requirement means adjustment of the tool for different Tropos dialects and annotations. Actually the requirements almost coincide with the functional requirements from the item 1, Section 3.1.

3 **Extensibility.** One of the main goals of the tool is to integrate different Tropos techniques around the visual modeler. That is why the tool should be easily extendible with new functionalities. The nonfunctional requirements about extensibility coincide with items 2 and 3 from Section 3.1. However there are additional requirements of this kind introduced here:
   a) Existing tools should be able to be wrapped and incorporated into the tool.
   b) There should be a way to create modules performing new functionality.
   c) There should be a way to integrate the formal checking techniques in TAOM4E.

4 **Tool Architecture**

A *Tropos* modeler called TAOM compliant with MDA metamodell interoperability standards has been described in [30]. The need of a higher flexible architecture which allow to easily extend it induced us to consider the opportunity to re-engineering this tool in the Eclipse Platform [21] that offers a flexible solution to the problem of component integration.

Eclipse is an open source software development project, the purpose of which is to provide a highly integrated tool platform. The Eclipse Platform is designed and built to meet the following requirements (as described in [21]):

1. Support the construction of a variety of tools for application development;
2. Support an unrestricted set of tool providers, including independent software vendors (ISVs);
3. Support tools to manipulate arbitrary content types;
4. Facilitate seamless integration of tools within and across different content types and tool providers;
5. Allows for easy extension by third parties;
6. Support both GUI and non-GUI-based application development environments;
7. Run on a wide range of operating systems;
8. Capitalize on the popularity of the Java programming language for writing tools;
9. Do not focus on any particular vertical domain.

The Eclipse Platform’s principal role is to provide tool providers with mechanisms to use, and rules to follow, that lead to seamlessly-integrated tools. These mechanisms are exposed via well-defined API interfaces, classes, and methods. The Platform also provides useful building blocks and frameworks that facilitate developing new tools. Figure 3 shows the major components of the Eclipse Platform (referring to release 3.0).

Eclipse is built on a mechanism for discovering, integrating, and running modules called plug-ins. A contributor to Eclipse delivers as one or more plug-ins an offering that manifests itself with a product-specific user interface in the workbench (see Figure 4). Multiple, usually unrelated, products can be installed in one Eclipse instance and happily live and cooperate to perform a certain task ([22]).
Except for a small kernel known as the Platform Runtime, all of the Eclipse Platform’s functionality is located in plug-ins. Plug-ins are coded in Java. A typical plug-in consists of Java code in a JAR library, some read-only files, and other resources such as images, web templates, message catalogs, native code libraries, etc. Some plug-ins do not contain code at all.

Each plug-in has a manifest file declaring its interconnections to other plug-ins. The interconnection model is simple: a plug-in declares any number of named extension points, and any number of extensions to one or more extension points in other plug-ins. A plug-ins extension points can be extended by other plug-ins. An extension point may have a corresponding API interface. Other plug-ins contribute implementations of this interface via extensions to this extension point. Any plug-in is free to define new extension points and to provide new API for other plug-ins to use (see Figure 5).

On start-up, the Platform Runtime discovers the set of available plug-ins, reads their manifest files, and builds an in-memory plug-in registry. The Platform matches
extension declarations by name with their corresponding extension point declarations so new features can be added not only easily but seamlessly. As you perform different tasks using Eclipse, it is usually impossible to tell where one plug-in ends and another begins (see [13]). In order to avoid a lengthy startup sequences, a plug-in is only activated when its code is actually needed based on user activity (lazy-loading [14]).

Figure 6 show the structure of the TAOM4E tool with regards to the Eclipse Platform and the other required plug-ins namely GEF and EMF that are described in the following section.
GEF The Graphical Editing Framework (GEF) is an open source framework dedicated to easily create rich graphical editors within Eclipse from an existing application model (see [20]).

![GEF Architecture](image)

**Fig. 7.** GEF and the Eclipse platform, see [20]

It has been developed in order to:
1. Display a Model graphically;
2. Allow the User to interact with that model;
3. Process and interpret user input from Mouse & Keyboard;
4. Provide hooks for updating the model making it full undo/redo-able.

Figure 7 show the relation between the Eclipse Platform and the 2 main component of the GEF framework (while their key responsibility are depicted in Figure 8):
1. Draw2D which provides figures and layout managers which form the graphical layer of a GEF application;
2. GEF core which provide an editing framework based on Viewers.

![GEF Architecture](image)

**Fig. 8.** GEF architecture, see [20]

GEF employs a model-view-controller (MVC) architecture:
1. **Model**: the framework is model agnostic, it works with any models that support the following:
   (a) Notification mechanism;
   (b) Persist and restore state;
   (c) Commands which operate on the model.
2. **View**: Gef offers two types of Viewers:
   (a) Graphical one, based on Draw2D Figures;
   (b) Tree one, based on SWT Tree and Tree items.
3. **Controller**: bridges the view and model. Each controller (EditPart) is responsible both for mapping the model to its view, and for making changes to the model. The EditPart also observes the model, and updates the view to reflect changes in the model’s state.

**EMF**

The Eclipse modeling framework (EMF) is a Java framework and code generation facility for building tools and other applications based on a structured model (see [33]). EMF helps you rapidly turn your models into efficient, correct, and easily customizable Java code. While Eclipse gives you an user interface and file level integration, EMF aims at a data integration level thus enabling model driven development [14].

EMF consists of two fundamental frameworks: the core framework and EMF.Edit. The core framework provides basic generation and runtime support to create Java implementation classes for a model. EMF.Edit extends and builds on the core framework, adding support for generating adapter classes that enable viewing and command-based (undoable) editing of a model, and even a basic working model editor.

While EMF uses XMI (XML Metadata Interchange) as its canonical form of a model definition, there are several equivalent ways of creating a model into that form (see Figure 9):

1. Create the XMI document directly, using an XML or text editor;
2. Export the XMI document from a modeling tool such as Rational Rose;
3. Annotate Java interfaces with model properties.

EMF started out as an implementation of the MOF (Meta Object Facility, an abstract language and framework for specifying, constructing, and managing technology neutral meta-models proposed by OMG), but evolved from there based on the experience gained from implementing a large set of tools using it. EMF can be thought of as a highly efficient Java implementation of a core subset of the MOF API. To avoid confusion, the MOF-like core meta model in EMF is called Ecore instead of MOF. Given the new MOF 2.0 standard, EMF is essentially the same as the EMOF subset.

4.1 **The TAOM meta-model**

According to EMF input requirements, the TAOM4E metamodel has been defined using the Rational Rose modeler ⁴. In order to assure an high level of flexibility the diagrams presented in this section have been slightly simplified in order to better let the reader grab the main concepts without looking at the implementation detail. See “Appendix B: TAOM4E meta-model diagram [as of version 0.1.5].” for the complete diagrams.
meta-model has been divided into two logical areas: business model (Core) and view (Diagram). The first should contain only the schema of the data (semantic information) that are related to the Tropos meta-model. It defines packages/classes etc. related to the methodology concept without referring to the diagram section. The second area is concerned with defining what is called the view model: it contains all the graphical information that define a graphical diagram (a view on the model) with the necessary link to the core model elements. This solution decouples model elements from view ones. A third utility area (Project) has been defined in order to better manage the different artifacts produced by the activity conducted during the various phases of the methodology supported by the tool.
The three high-level packages composing the meta-model are depicted in Figure 10:

1. The Project package (Figure 11) define the concept of Project as an aggregation of different Diagrams (i.e. actor diagram and goal diagram) that are related to a BusinessModel (namely the Tropos meta-model). This package serves as an aggregator of all the data related to a Tropos Project into a single logical structure that will be saved into an xmi document, avoiding synchronization problem of the different component with the shared model.

2. The Diagram package (Figure 12) define the structure and the relationship of the graphical element that will be used in the different diagrams. The GenDiagram package define the generic structure of a diagram supported by the tool (Diagram) as an aggregation of different kind of DiagramObject (namely DiagramConnection and DiagramNode) representing graphical connection and graphical element respectively. A special kind of node object (ContainerDiagramNode) is created in order to manage containment relationship between graphical element (i.e. the Actor container in the Goal Diagram). Every DiagramObject could be related to the core element/s it is representing (an instance of TroposModelElement). Given that structure in place representing a particular kind of diagram is a matter of creating a new package defining the graphical element as extension of the one already present in GenDiagram. At the moment we support two Tropos diagrams, namely actor and goal diagram. The main objective of this packages (and all its sub-ones) is to create a view on the different element of the model that could be persisted in an xmi file maintaining a clear separation and independence between the different layer.

3. The Core packages (Figure 13) mimic the structure used in the Diagram one in order to define the core model of the tool hosting the Tropos meta-model. The GenCore package define a model (BusinessModel) has an aggregation of different kind of TroposModelElement namely TroposClass and TroposRelation object that map respectively to an element and a relation defined in the Tropos meta-model. The InformalCore package contain the definition of the element representing the Tropos meta-model (i.e. Actor, Goal, Dependency, Contribution). The twin pack-
For the FormalCore extension, add the Formal Tropos specific element (i.e. GlobalProperty, LTL formula and attribute) to the pure Tropos corresponding one.

5 Tool documentation

All the public information about the TAOM4E tool can be found in the tool’s web site [35]. The site contains the link to the TAOM4E plug-in; the requirements on the installation of the tool and the installation process; the information about the license; the development team list and papers on the TAOM4E tool. Also it allows for sending founded bugs and proposed improvements.

A brief demonstration about how to work with the tool can be downloaded from the following link [34].
Fig. 13. Core package of the TAOM4E meta-model.

6 Tool extensions and maintenance

This section contains the information about the current work on the TAOM4E tool and current extensions on the architecture of the tool.

In the Appendix A can be found the document with all the current requirements, bugs to be corrected and the features under development. It contains the short description of the problems, the status and the difficulties.

Appendix B contains the current versions of the diagrams placed in the Sections 4.

7 Future Work

In this report we described: the motivations for the development of a CASE tool at support of Agent-Oriented Modelling in Tropos, called TAOM4E; the requirements on the tool’s functionalities; and the tool’s architecture.

Flexibility and modularity requirements motivated the choice of Eclipse Platform.

Among the next extensions of TAOM4E: the integration of plug-ins implementing automatic model-to-model transformations following the Model Driven Architecture approach to software development as described in [28]; the integration of tools supporting specific analysis techniques of Tropos like Goal Reasoning Tool (GR-Tool) [36] which support the goal analysis, the Secure Tropos Tool (ST-Tool) [37] which support security modelling; the implementation of the functionalities for the support of the Tropos process providing guidelines on the development of Tropos models.
References

<table>
<thead>
<tr>
<th>N°</th>
<th>Description</th>
<th>Inquirer</th>
<th>Status</th>
<th>Conclusion</th>
<th>Priority</th>
<th>Complexity</th>
<th>Interdependence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Metamodel, basics</td>
<td>Angelo</td>
<td>change</td>
<td>done - released with version 0.1.5</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1.1</td>
<td>A problem of consistency between different diagrams: The examples of the scenario can be the following two: creation of an Actor 1, creation of a goal A in the actor diagram, creation of the goal diagram for Actor 1 and analysis of the goal A in that goal diagram, back to actor diagram, creation of an Actor 2, delegation of goal A from Actor 1 to Actor 2, ... What happens to the goal analysis of the goal A for in the goal diagram related to Actor 1? &quot;ho trovato difficoltà a delegare un goal, prima integrato ad un attore, poi ad un dettaglio di analisi non ancora implementato, poi ad un altro attore&quot;</td>
<td>Angelo, Leo, Andrea</td>
<td>bug, improvement</td>
<td>to do, there are several of such kind of inconsistency cases, requires general view and approach</td>
<td>0</td>
<td>5</td>
<td>partially depends on 1.4, 3.1;</td>
</tr>
<tr>
<td>1.3</td>
<td>Possible means end relationships between plan and resources?</td>
<td>Angelo</td>
<td>change</td>
<td>no</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1.4</td>
<td>Create single command stack for all the views</td>
<td>Davide, Andrea</td>
<td>improvement</td>
<td>done - released with version 0.1.5</td>
<td>2</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>1.5</td>
<td>Metamodel changes: a) core metamodel change</td>
<td>Ana, Angelo</td>
<td>change</td>
<td>planned</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td>b) security profile</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**TAOM4E: the evaluation results, the things to improve and to fix.**
<table>
<thead>
<tr>
<th>No</th>
<th>Description</th>
<th>Initiator</th>
<th>Status</th>
<th>Conclusion</th>
<th>Priority</th>
<th>Complexity</th>
<th>Interdependency</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>The appearance, comfort of the analyst</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.1.1</td>
<td>Change icons in the palette and the outline tree</td>
<td>Anna, Angelo, Alakser</td>
<td>improvement</td>
<td>to do, to think more about design in general</td>
<td>3</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>2.1.2</td>
<td>Make a special &quot;Tropos&quot; Eclipse perspective to hide unnecessary tabs and views</td>
<td>Anna</td>
<td>future feature</td>
<td>done – the complexity was overestimated, the real complexity – 4 – released with version 0.15</td>
<td>4</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>2.1.3</td>
<td>Not as evident &quot;that&quot; relationship between actor and goals in actor diagram</td>
<td>Anna</td>
<td>improvement</td>
<td>done – released with version 0.15 – becomes automatically with visual properties</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.1.4</td>
<td>Multiple dependencies graphically overlap at creation time</td>
<td>Anna</td>
<td>future feature</td>
<td>to do</td>
<td>5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>2.1.5</td>
<td>Clarify the names of the commands appearing when we do undo/redo</td>
<td>Anna, Alakser</td>
<td>bag</td>
<td>to do</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2.1.6</td>
<td>Create and/or decomposition directly from the palette</td>
<td>Anna, Angelo</td>
<td>improvement</td>
<td>done – released with version 0.15 – 2</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2.1.7</td>
<td>When open a new project, one (main) actor diagram for every requirements phase should be created and opened in the editor</td>
<td>Alakser</td>
<td>improvement</td>
<td>Davide says not to do, Alakser says to do</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2.1.8</td>
<td>The order of the elements in the actor diagram palette has to be reversed elements first, then relationships</td>
<td>Angelo</td>
<td>improvement</td>
<td>done – released with version 0.15 – 3</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2.1.9</td>
<td>AND/OR links: is it useful a label AND OR on the link?</td>
<td>Angelo</td>
<td>improvement</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.1.10</td>
<td>The initial creation of a goal diagram from an actor takes too much time, the goal diagram has to be created directly from the icon of an actor in an actor diagram</td>
<td>Angelo, Alakser</td>
<td>improvement</td>
<td>done – released with version 0.15 – 5</td>
<td>2</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>2.1.11</td>
<td>Multiple goal diagrams has to be accessible directly from the editor</td>
<td>Angelo</td>
<td>improvement</td>
<td>done – released with version 0.15 – 3</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>Description</td>
<td>Initiator</td>
<td>Status</td>
<td>Conclusion</td>
<td>Priority</td>
<td>Complexity</td>
<td>Interdepenency</td>
</tr>
<tr>
<td>---</td>
<td>------------------------------------------------------------------------------</td>
<td>-----------</td>
<td>------------</td>
<td>------------------------------------------------------------------------------</td>
<td>----------</td>
<td>------------</td>
<td>---------------</td>
</tr>
<tr>
<td>2.1.2</td>
<td>E possibile finire la dimensione dei构件 per gli user? Eò notato che di scivolare svuotano longhi, si prevede a segnare il taggio automaticamente.</td>
<td>Loris</td>
<td>improvement</td>
<td>we have found at least two variables to give to the user as the flexibility options ... it needs to be discussed more</td>
<td>4</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>2.1.13</td>
<td>Diagram's outline should start from the name of the diagram as the root, as it is done for the project outline tree (it starts from the name of the project).</td>
<td>Alaksei</td>
<td>improvement</td>
<td>no</td>
<td>4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2.1.14</td>
<td>Would not be better if we remove the &quot;Actor Diagram&quot; phase after the actor diagram picture because the picture should say if it is actor or goal diagram.</td>
<td>Alaksei</td>
<td>improvement</td>
<td>strongly correlates with the item on the scenes and design</td>
<td>4</td>
<td>1</td>
<td>strongly with 2.1.2</td>
</tr>
<tr>
<td>2.1.15</td>
<td>Do we need the &quot;Diagram outline&quot; at all?</td>
<td>Alaksei</td>
<td>proposition</td>
<td>yes, we need it, do not change for this moment</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.1.16</td>
<td>The black frame around the exported picture -- the frame around the diagram</td>
<td>Angelo</td>
<td>improvement</td>
<td>done -- together with it the mechanism of previewing empty editors and launching previously edited diagrams created -- released with version 0.1.5</td>
<td>5</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2.2</td>
<td>The functionality</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.2.1</td>
<td>&quot;Save as&quot; does not work</td>
<td>Alaksei</td>
<td>bug</td>
<td>done -- released with version 0.1.5</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2.2.2</td>
<td>Increase the number of active points around the figures (change the anchor of the figures)</td>
<td>Angelo, Davide, Alaksei</td>
<td>future feature</td>
<td>to do</td>
<td>4</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>2.2.3</td>
<td>There is a problem in the definition of the names of the diagrams</td>
<td>Angelo</td>
<td>improvement</td>
<td>not exist</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.2.4</td>
<td>Off the MAC - the diagram scroll is not working well</td>
<td>Angelo</td>
<td>bugs</td>
<td>check if it works or not in other GEF editors</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No.</td>
<td>Description</td>
<td>Initiator</td>
<td>Status</td>
<td>Conclusion</td>
<td>Priority</td>
<td>Complexity</td>
<td>Interdependency</td>
</tr>
<tr>
<td>-----</td>
<td>-------------</td>
<td>-----------</td>
<td>----------</td>
<td>------------</td>
<td>----------</td>
<td>------------</td>
<td>----------------</td>
</tr>
</tbody>
</table>
|     | - Area selection (Marker function) uses a white (unselectable) line  
|     | - BMP image export does not work, there are some  
|     | - jpg is not a good format for the representation  
|     | - problems in printing the diagrams | Loris, Aliaksei | improvement | to do | 2 | 4 |
| 2.2.5 | Repeat the model tree structure as the diagram tree structure. Make from the model tree the view on the model tree as it was discussed with Davide, Aliaksei, Angelo and Loris. The related comment: "Nella descrizione del modello, ciascun elemento dovrebbe essere maggiore o minore in proporzione con il dipendente e il dipendente. Cioè, l'attuale lista piatta dei dipendenti non  
|     | | | | | | | facilitates its use in the diagrams in succession. For example, ensuring a hierarchy  
|     | | | | | | | is a level of early requirement. Even more, the  
|     | | | | | | | description of the property would be articulated in  
|     | | | | | | | such a manner." | Loris, Aliaksei | improvement | to do | 3 | 5 | strongly with 1 2 |
| 2.2.6 | Add to the properties of an intentional element the possibility to see the owner-actor and to change the  
|     | | | | | | | owner-actor also to see some things in the properties (like dependences, contributions) and ownership | Aliaksei, Loris | improvement | to do | 3 | 5 | strongly with 1 2 |
| 2.2.7 | Se c’è una cancellazione di un attore, potrebbe essere un  
|     | | | | | | | altro, come faccio a rimpiazzare le dipendenze rimaste in  
|     | | | | | | | segno della prima cancellazione? Le dovo riportare  
|     | | | | | | | tutto di nuovo?  
|     | | | | | | | Cioè, credo che piutosto di procedere cancellando  
|     | | | | | | | (diporre) il link per poi ripozzerarlo, sia più  
|     | | | | | | | comodo rendere  
|     | | | | | | | eliminaibili i link rimasti liberi | Loris | bug | seems not existent |
| 2.2.8 | Quando c’è una cancellazione di un attore, lo stesso  
|     | | | | | | | anche un altro, lo stesso  
|     | | | | | | | anche un altro  
|     | | | | | | | attore. Inoltre, è chiaro che c’è un errore al momento del  
|     | | | | | | | creazione del  
<p>|     | | | | | | | link | Loris | bug | seems not existent |</p>
<table>
<thead>
<tr>
<th>N.</th>
<th>Description</th>
<th>Initiator</th>
<th>Status</th>
<th>Conclusion</th>
<th>Priority 0-10</th>
<th>Complexity 1-10</th>
<th>Interdependency</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3</td>
<td>Drag &amp; Drop</td>
<td>Angelo</td>
<td>improvement</td>
<td>to do</td>
<td>3</td>
<td>6</td>
<td>1.3.2, 1.3.3</td>
</tr>
<tr>
<td>2.3.1</td>
<td>Drag and Drop of a goal hierarchy also from the diagram tree. It would be nice to have a possibility to drop a number of elements once.</td>
<td>Angelo</td>
<td>improvement</td>
<td>to do</td>
<td>3</td>
<td>6</td>
<td>1.3.2, 1.3.3</td>
</tr>
<tr>
<td>2.3.2</td>
<td>Control of what element can be created with drag&amp;drops. I think it should be done in the add element command. Now you can drop the elements that already exist in the diagram or put many actors to the goal diagram. Also we can put the goal on the dependency as the goal without the dependency.</td>
<td>Alahkari</td>
<td>bug, improvement</td>
<td>partially done, done for the element part, need more works for the Goal diagram relation - released with version 0.1.5</td>
<td>1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>2.3.3</td>
<td>It would be nice to have a possibility to create all the links to the adddropped object (like dependency to the add actor).</td>
<td>Alahkari</td>
<td>improvement</td>
<td>to do</td>
<td>3</td>
<td>5</td>
<td>1.3.2</td>
</tr>
<tr>
<td>2.3.4</td>
<td>It seems like when we add a goal to the actor diagram we do not have to point the owner actor because it is already known. The command should check if there is the owner-actor on this diagram and then allow to put the goal there.</td>
<td>Alahkari</td>
<td>preposition</td>
<td>to do, discussed with Davide, Angelo, Lorca and Alahkari</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Planned for future</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| 3.1 | Mixed diagram! It includes some comments like: "E' possibile avere dipendenza esterno all'"sconto" di un goal da un diagramma di un altro? E' importante che il Goal Diagram venga inserito dal contesto gia mediatato."
"Aggiungendo alla istruzione (C), serve anche la possibilita di creare un diagramma con più aspetti, un diagramma con più possibili dipendenze." | future feature | to do | 0 | 8 | strongly with 1.2 |
<table>
<thead>
<tr>
<th>N°</th>
<th>Description</th>
<th>Initiator</th>
<th>Status</th>
<th>Conclusion</th>
<th>Priority</th>
<th>Complexity</th>
<th>Interdepenedy</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2</td>
<td>Concenere le salvataggi in formati web (eps, jpg, png, pdf, bmp)</td>
<td>Lotua</td>
<td>improvement</td>
<td>it seems like it is quite a problem for the release of GEP, needs more investigations</td>
<td>2</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>3.3</td>
<td>Export/Print the diagrams of WHOLE the project</td>
<td>to do</td>
<td></td>
<td></td>
<td>4</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>3.4</td>
<td>To have the arc links for the dependences and contributions</td>
<td>Lotua, Aliaski</td>
<td>improvement</td>
<td>to discuss more</td>
<td>4</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>3.5</td>
<td>Documentation</td>
<td>done</td>
<td></td>
<td></td>
<td>2</td>
<td>5</td>
<td>With everything</td>
</tr>
<tr>
<td>3.6</td>
<td>Visual properties and project's properties</td>
<td>done - it was underestimated, the real complexity was 9 - released with version 0.1.5</td>
<td>1</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.7</td>
<td>Think about the order in the picture (put to background/foreground). We will probably need to change the metamodel - properties of the diagram object</td>
<td>to do</td>
<td></td>
<td></td>
<td>5</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3.8</td>
<td>Undo/redo commands for deletion of the elements from the model</td>
<td>done - released with version 0.1.5</td>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.9</td>
<td>To think how we can use OCL (Keep OCL plug-in for example) for manipulations on the model (queries, views)</td>
<td>Aliaski</td>
<td>future feature</td>
<td>to do</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
Appendix B: TAOM4E meta-model diagram [as of version 0.1.5].
Fig. 14. TAOM4E meta-model main packages
Fig. 15. The Project package.
Fig. 16. The GenDiagram package.
Fig. 17. The ActorDiagram package.
Fig. 18. The GoalDiagram package.
Fig. 19. The GenCore package.
Fig. 20. The InformalCore package.
Fig. 21. The FormalCore package.
Fig. 22. The Property package.