The Tropos Metodology and its Metamodel

ITC-irst, Università di Trento

Abstract. Tropos is an Agent-Oriented software development methodology that supports the software development process from the early requirements phase to the implementation. Here we introduce and discuss the Tropos metamodel, based on knowledge level concepts such as those of actor, goal, plan, resource and social dependency, then we present TAOM4E, a modelling environment that supports the designer during the Tropos modelling activities.
Acknowledgements.

This document is part of a research project funded by the FIRB 2001 Programme of the “Ministero dell’Istruzione, dell’Università e della Ricerca” as project number RBNE0195K5.

The partners in this project are: Istituto Trentino di Cultura (Coordinator), Università degli Studi di Trento, Università degli Studi di Genova, Università degli Studi di Roma “La Sapienza”, DeltaDator S.p.A.
Executive Summary

Software development paradigms have exploited a wealth of models to capture requirements and design information about a software system (the “system-to-be”) throughout its development process. Structured software development used SADT and Data Flow Diagrams. Object-oriented software development has used a range of modelling languages which have been integrated into UML. Not surprisingly, agent-oriented software development is following on the same footsteps.

To formally analyze software models, we need a means to define their syntax and semantics. Metamodels have been used for the former task. Metamodels define a set of possible instantiations, which are all and only the syntactically correct models in some modelling language. As such, metamodels have been used for more than two decades as a basis for defining the syntax of (usually graph-theoretic) modelling languages, such as UML as well as Tropos.

In the context of the project we defined and implemented a metamodel for the Tropos methodology. This metamodel has been integrated in an environment that supports the software development process from the Early phase to the system architecture definition. The environment, named Tool for Agent Oriented Modeling (TAOM), has been initially conceived as a modeler, supporting agent-oriented modeling in Tropos [BBD+02]. Main motivations for this effort are: (i) requests from practitioners who ask for supporting tools and guidelines when applying a new methodology; (ii) need to support the informal to formal Tropos framework, aiming at a light integration of model checking techniques for the validation of portions of a Tropos model [PPRA04]; (iii) need to explore the exploitation of Model Driven Architecture principles into the Tropos design process [pS04]. Currently, the development and extensions 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 [pS04] 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) [teab].

The objective of this document is to introduce the Tropos metamodel and discuss some of its uses. Chapter 1 of the document describes the Tropos methodology, while Chapter 2 presents the metamodel and explains its features, also discussing a possible extension of the metamodel to include security-related concepts. In Chapter 3 we describe the Tropos development environment, which uses the metamodel in its basic core, while Chapter 4 describes the history of the deliverable.
# Contents

1 Models and Methodology 1
   1.1 Tropos ....................................................... 1
   1.2 Model Driven Architecture ................................. 3
   1.3 Related Work ................................................. 4

2 The Metamodel 5
   2.1 Metamodel Extension ....................................... 8

3 A Modelling Environment 11
   3.1 Tool Requirements ........................................... 11
      3.1.1 Functional requirements ............................... 11
      3.1.2 Nonfunctional requirements ............................ 13
   3.2 Tool Architecture ........................................... 14
      3.2.1 The TAOM meta-model ................................. 19
   3.3 Tool documentation ......................................... 22
   3.4 Tool maintenance and extensions ......................... 23

4 History of the Deliverable 25
   4.1 1st year ..................................................... 25
   4.2 2nd year ..................................................... 25
   4.3 3rd year ..................................................... 25
   4.4 4th year ..................................................... 26

A TAOM4E requirements table 27

B TAOM4E meta-model diagram 34
Chapter 1

Models and Methodology

1.1 Tropos

*Tropos* is founded on the idea of using the agent paradigm and related mentalistic notions during all phases of the software development process. The methodology [BGG+04] adopts the $i^*$ [Yu95] modelling framework, which proposes the concepts of (social) *actor, goal, task, resource* and social *dependency* to model both the system-to-be and its organizational operating environment. The $i^*$ framework includes the strategic dependency model (actor diagrams in *Tropos*) for describing the network of inter-dependencies among actors, as well as the strategic rationale model (goal diagrams in *Tropos*) for describing and supporting the means-ends analysis conducted by each actor as it attempts to ensure that – through delegations to other actors – its goals will eventually be fulfilled.

An *actor diagram* is a graph whose nodes represent actors (*agents, positions, or roles*), while edges represent dependencies among them. A dependency represents an agreement between two actors where one actor (the depender) depends on another (the dependee) to fulfill a goal, perform a task or deliver a resource (the dependum). Dependencies may also involve softgoals (such as “having a good quality meeting”) which represent vaguely-defined goals, with no clear-cut criteria for their fulfillment.

A *goal diagram* is also a graph where nodes represent goals or plans\(^1\), while edges represent goal/plan relationships, such as AND/OR-decomposition (i.e., a goal/plan can be decomposed into a set of other goals/plans. Goals/plans can also be related to softgoals through qualitative relationships (labelled “+” or “-”) to indicate that the goal/plan contributes positively or negatively to the fulfillment of the softgoal. Goal diagrams appear inside a balloon associated with a single actor. This is the actor whose goals/plans are being analyzed to determine how they can be fulfilled/executed.

The *Tropos* methodology supports four phases of software development: Early Requirements Analysis, Late Requirements Analysis, Architectural Design, and Detailed

---

\(^1\)Plans in *Tropos* correspond to tasks in $i^*$. 
Design.

*Early requirements* is concerned with understanding the organizational context within which the system-to-be will eventually function. During early requirements analysis, the requirements engineer identifies the domain stakeholders (who have a stake in the system-to-be) and models them as social actors, who have goals and depend on each other for goals to be fulfilled, plans to be performed, and resources to be furnished.

*Late requirements*, on the other hand, is concerned with a definition of the functional and non-functional requirements of the system-to-be. This is accomplished by treating the system as another actor (or a small number of actors) who are dependers/dependees in dependencies that relate them to external actors. The shift from early to late requirements occurs when the system actor is introduced and it participates in delegations from/to other actors.

*Architectural design* is concerned with the global structure of the system-to-be. Unsurprisingly, subsystems and system components are represented as actors too, and their dependencies to other system components are social, rather than procedural/structural. This means that system components need to have the ability to monitor dependencies to other actors to make sure they will be fulfilled. As well, system components need to be able to cancel dependencies that seem ineffective and replace them with new ones through planning, negotiation, etc. As with conventional software architectures, architectural styles constitute critical support for the software developer. Since the fundamental concepts of Tropos architectures are intentional and social, we have turned to theories which study social structures to define architectural styles: namely Organization Theory and Strategic Alliances.

*Detailed design* focuses on the specification of actor communication and behavior. To support this phase, we have adopted existing agent communication languages such as FIPA-ACL [LFP99] or KQML [FLM97]; also message transportation mechanisms and other related concepts and tools. We have also proposed and defined a set of stereotypes, tagged values, and constraints to accommodate *Tropos* concepts within UML [BRJ99].

*Implementation*: here code is produced on the basis of the artifacts produced during the detailed design phase. At this stage agent platforms, such as JADE [BP01] or JACK [Cob00] can be used to implement the system-to-be.

Through the models constructed during these phases, one can answer “why” questions, in addition to “what” and “how” ones, regarding system functionality. For example, one can ask “Why does this component of the system need to notify library users when a book becomes available”. Answers to why questions ultimately link system functionality to stakeholder needs, preferences and objectives. Such answers serve as ultimate justifications for all elements of a proposed design. Table 1.1 summarizes the Activities and Artifacts in the *Tropos* development phases.
Table 1.1: Activities performed and Artifacts produced during the *Tropos* development phases.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Description of Actors, Goals, Plans, Resources, Dependencies in the organizational setting; internal analysis of domain actors via Decompositions, Means-ends and Contributions</td>
<td>Introduction of the System Actor, and specification of the Dependencies with ER organizational Actors; internal analysis of the system actor via Decompositions, Means-ends and Contributions</td>
<td>Introduction of Subsystem Actors; definition of subsystem Dependencies</td>
<td>Agent specification</td>
<td>Agent implementation</td>
<td></td>
</tr>
<tr>
<td>Artifacts</td>
<td>ER Actor diagram ER Goal diagram</td>
<td>LR Actor diagram LR Goal diagram</td>
<td>Architectural diagram</td>
<td>FIPA-ACL, AUML/UML class, sequence, activity, diagrams</td>
<td>JADE or Jack agent definition</td>
</tr>
</tbody>
</table>

### 1.2 Model Driven Architecture

In the definition of the methodology and of the requirements for the supporting tools 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 [Bro04]. 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 ex-
ploitation 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) [Obj], and a set of require-
ments 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) ap-
proach [GGKH03]. 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.

1.3 Related Work

Many Agent-Oriented Software methodologies have been proposed over the last few
years [OGM04, GMO03, GOW02, WCW01, CW01] and work were they are compared
with other to point out differences and complementarities are enriching the AOSE liter-
ature. For instance, an evaluation of Tropos with respect to other methodologies can be
found in [HSG05, SS02], 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 [BCG04] considering ADELFE [BGPP02],
GAIA [ZJW03] and PASSI [pas03]. The aim of that work was to face interoperability
issues between different methodologies.

In [SPGM05] we can find an extension of this comparison to Tropos. This compar-
ison points out the fact that different metamodels (methodologies) may allow to model
different properties of a system (e.g organizational aspects, communications and proto-
cols), 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 [PS05].
Chapter 2

The Metamodel

In this chapter we introduce the *Tropos* metamodel and some examples to explain some of its characteristics. Figure 2.1 shows the portion of the *Tropos* metamodel, where agent, role and position are specialization of the concept of actor. A position can *cover* 1...n roles, whereas an agent can *play* 0...n roles and can *occupy* 0...n positions. An actor can have 0...n goals, which can be both hard and softgoals and are wanted by 1 actor.

An actor *dependency* is a quaternary relationship and relates respectively a depender, dependee, and dependum (i.e. goal, plan, resource). It is possible to specify also a reason for the dependency (labeled as *why*).

A model is an instance of the metamodel and can have a graphical representation in terms of actor and goal diagrams.

Figure 2.2 depicts an example of an actor diagram for the domain of the Conference Review Process and represents a model that can be obtained instantiating the metamodel discussed so far. Three actors are involved: the Program Committee Chair (*PC Chair*), the Program Committee Member (*PC Member*) and the Reviewer. Dependencies take place between them; in particular the goal *review papers* is delegated by the *PC Chair* to the *PC Member*, moreover the *PC Chair* also expects to have the information of the possible conflicts (a resource dependency) between the *PC Member* and the authors of the papers. On the other hand, the *PC Member* depends on the *PC Chair* to obtain the papers to distribute and the review form. Many critical goal and resource dependencies occur between the *PC Member* and the Reviewer. In particular, the *PC Member* depends on the Reviewer for review the papers and to obtain the information about the possible conflicts on assigned papers. The Reviewer depends on the *PC Member* in order to obtain a set of assigned papers as well as the review form. Finally, the *PC Member* wants to be fair in the review assignment, and this is represented as a softgoal wanted by the *PC Member*.

The concepts related to the *Tropos* goal diagram are depicted in Figure 2.3. The central
Figure 2.1: The UML class diagram specifying the actor concept and the dependency relationship in the Tropos metamodel. UML notation is compliant with the OMG MOF 1.4.

Figure 2.2: The Tropos actor diagram describing a sketch of the conference review process.

The concept of goal is represented by the class Goal. Goals can be analyzed, from the point of view of an actor, by Means-end analysis, Contribution analysis and Boolean decomposition. Means-end Analysis is a ternary relationship defined among an Actor, whose point of view is represented in the analysis, a goal (the end), and a Plan, Resource or Goal (the means). Contribution Analysis is a ternary relationship between an actor, whose point of view is represented, and two goals. Contribution analysis strives to identify goals that can contribute positively or negatively towards the fulfillment of other goals (see association...
Figure 2.3: The UML class diagram specifying the concepts related to the goal diagram in the Tropos metamodel.

Figure 2.4: The Tropos goal diagram related to the actor PC Member.

relationship labeled contribute in Figure 2.3). A contribution can be annotated with a qualitative metric, as proposed in [CNYM00], denoted by $+, ++, -, --$. In particular,
if the goal $g_1$ contributes positively to the goal $g_2$, with metric $++$ then if $g_1$ is satisfied, so is $g_2$. Analogously, if the plan $p$ contributes positively to the goal $g$, with metric $++$, this says that $p$ fulfills $g$. A $+$ label for a goal or plan contribution represents a partial, positive contribution to the goal being analyzed. With labels $--$, and $-$ we have the dual situation representing a sufficient or partial negative contribution towards the fulfillment of a goal. Decomposition, whose metamodel is described in Figure 2.3, is also a ternary relationship which defines a generic boolean decomposition of a root goal into subgoals, that can be in particular an AND- or an OR-decomposition specified via the attribute Type in the class Boolean Decomposition specialization of the class Decomposition.

The concept of plan in Tropos is specified in Figure 2.2 and 2.3. Means-end analysis and AND/OR decomposition, defined above for goals, can be applied to plans also. In particular, AND/OR decomposition allows for modelling the plan structure.

Figure 2.4 gives a sketchy view of goal diagram for the actor PC Member and for the goal review papers and for the softgoal be fair in the review assignment. The goal review papers has been AND-decomposed in two sub goals: assign papers to reviewers and collect the reviews. This latter represents the “Why” for the dependency review the papers between PC Member and Reviewer, as shown in Figure 2.1. The goal assign papers to reviewers is decomposed in two subgoals: send the papers, that is operationalized as send papers by e-mail, and select reviewers decomposed in verify the competences and verify conflicts. This latter represents the “Why” for the resource dependency conflicts between the PC Member and the reviewer. Moreover, the fulfillment of these two sub-goals can contribute positively to the fulfillment of the softgoal be fair in the review assignment as described by the positive contribution relationships in the diagram.

2.1 Metamodel Extension

Secure Tropos has been proposed in [GMMZ04] as a formal framework for modelling and analyzing security. It enhances Tropos introducing four new concepts and relationships behind Tropos dependency: trust, delegation, provisioning, and ownership. The basic idea of ownership is that the owner of a resource (goal or plan) has full authority concerning access and disposition of his resource (goal or plan). The distinction between owning a resource makes it clear how to model situations in which, for example, a client is the legitimate owner of his/her personal data and a Web Service provider that stores customers’ personal data, provides the access to her/his data. We use the relation for delegation when in the domain of analysis there is a formal passage of authority (e.g. a signed piece of paper, a digital credential is sent, etc.). The trust relations have their intuitive meaning among agents, namely the believe of an agent that the actor does not misuse some resources.
Figure 2.5 shows the new part of the *Tropos* metamodel concerning trust and ownership. An actor (the *truster*) trusts another actor (the *trustee*) about the achievement of a goal, the fulfillment of a plan or the delivering of a resource. The content of the trust relationship is called *trustum*. An actor can be the owner of a resource, a plan and goal and he/she has authority concerning the use of the resource, the execution of the plan and achievement of the goal, respectively.

![Figure 2.5: The Tropos metamodel related to the concept of Trust.](image)

The metamodel describing delegation relationships is basically identical to the metamodel for the dependency relationship as presented in Figure 2.1. The *delegater* delegates the *delegatee* for the achievement of a goal, the execution of a plan or the delivering of a resource. As for the dependency relationship, it is also possible here to specify the reason (why) of a delegation.

We have shown in [GMMZ05] how the original concept of *Tropos* dependency can be expressed in terms of trust and delegation. Roughly, when an actor depends on another actor to achieve a goal (to fulfill a task or to deliver a resource), it is implicitly intended that the actor trusts the other actor and delegates it for such activities. A precise formalization of dependency refinement in terms of trust and delegation has been presented in [GMMZ05].

Figure 2.6 presents an example of application the extended metamodel. The *Author* trusts the *PC Chair* to implement a fair review process and he/she is the owner of the paper sent to the *PC Member* and reviewed by the *Reviewer*. The *PC Chair* trusts and delegates *PC Member* to review a certain number of papers, and in turn the *PC Member* trusts and delegates the *Reviewer* to review the papers. The *PC member* (*Reviewer*) depends on the *PC Chair* (*PC Member*) to receive the paper to review.
Figure 2.6: The *Tropos* actor diagram with the trust concepts.
Chapter 3

A Modelling Environment

In order to support the designer in exploiting the Tropos methodology during the system development process, we designed and implemented the TAOM4E development environment. It is based on the Tropos metamodel described in the previous chapter. In the following we show the TAOM4E system requirements then we give some details about its design and implementation.

3.1 Tool Requirements

A core set of requirements of an environment at support of the use of the Tropos methodology have been presented in [BBD+02], where the basic objective was to develop an AO modeler. Here we describe a set of functional and nonfunctiona requirements that have been implemented in the current version of the TAOM4E environment.

3.1.1 Functional requirements

The functional requirements are represented in Figure 3.1 as a use case diagram. All the use cases correspond 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. Providing 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 Support of the *Tropos* process. In particular:

   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.

3.1.2 Nonfunctional requirements

We believe that usability, extensibility and flexibility nonfunctional requirements are 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 3.1.1. The three categories of nonfunctional requirements we are interested in are described in the following:

1 **Usability.** The requirements include all the things which make convenient visual modelling process like:

   a) Have the palette to choose the objects drown 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 3.1.1.

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 extensible with new functionalities. The nonfunctional requirements about extensibility coincide with items 2 and 3 from Section 3.1.1. 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.
3.2 Tool Architecture

A Tropos modeler called TAOM, compliant with MDA metamodel interoperability standards, has been described in [pS04]. 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 [Inc03] 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 [Inc03]) :

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 3.2 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 3.3). Multiple, usually unrelated, products can be installed in one Eclipse instance and happily live and cooperate to perform a certain task ([J. 04]).

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-in 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 3.4).

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 [D. 03]). 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 [F. 03]).

Figure 3.5 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 [Hud04]).

![Diagram of GEF and Eclipse platform]

Figure 3.6: GEF and the Eclipse platform, see [Hud04]

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 3.6 show the relation between the Eclipse Platform and the 2 main component of the GEF framework (while their key responsibility are depicted in Figure 3.7):

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 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:
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 [tea02]). 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 [F.03].

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 3.8):

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 Architecture](image)

Figure 3.8: EMF Architecture, see [Mer04]

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.

### 3.2.1 The TAOM meta-model

According to EMF input requirements, the TAOM4E metamodel has been defined using the Rational Rose modeler. In order to assure an high level of flexibility the meta-model has been divided into two logical area: 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 define packages/classes etc. related to the methodology concept without referring 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.

---

1 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.
elements. This solution decouple model element from view ones. A third utility area (Project) has been defined in order to better manage the different artifact 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 3.9:

1. The Project package (Figure 3.10) 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.

Figure 3.10: Project package of the TAOM4E meta-model.

2. The Diagram package (Figure 3.11) 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

Figure 3.11: Diagram package of the TAOM4E meta-model.

3. The *Core* packages (Figure 3.12) 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 package, FormalCore, add the Formal Tropos specific element (i.e. GlobalProperty, LTL formula and attribute) to the pure Tropos corresponding one.

Figure 3.12: Core package of the TAOM4E meta-model.

3.3 Tool documentation

All the public information about the TAOM4E tool can be found in the tool’s web site [teab]. 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 [teaa].
3.4 Tool maintenance end extensions

This chapter 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. In Figure 3.13 is shown a screenshot of the tool.

![Figure 3.13: The TAOM4E Graphic User Interface. The largest window, on the right side, supports models editing according to the Tropos graphical notation that is provided by the Palette window (in the center). Notice that, visualized diagrams are often partial views on the whole model represented in the outline window on the left.](image)

Flexibility and modularity requirements motivated the choice of Eclipse Platform.

Concerning system maintenance, Appendix A contains the document with all the current requirements, bugs to be corrected and the features under development. In particular, 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 3.2.

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 [PS05]; the integration of tools supporting specific analysis techniques of Tropos like Goal Reasoning Tool (GR-Tool) [Tro05a] which support the goal analysis, the Secure Tropos Tool (ST-Tool) [Tro05b] which support security modelling; the implementation of the functionalities for the support of the
*Tropos* process providing guidelines on the development of *Tropos* models.
Chapter 4

History of the Deliverable

In this chapter a short description of the history of the deliverable along the four years of the KLASE project.

4.1 1st year

During the first year of the KLASE project, the main effort has been devoted to the definition of the Tropos metamodel via knowledge level concepts such as those of actor, goals, social dependencies. In parallel, a first set of requirements for a tool that supports the Tropos modelling activities has been defined. These requirements allowed for the design and implementation of a first prototype of Tropos modelling environment, named TAOM, that integrates the Tropos metamodel.

4.2 2nd year

In the second year, an activity of refinement of the Tropos metamodel has been conducted. Moreover, the TAOM prototype has been validated in-field in order to capture and define the set of requirements for the implementation of the final TAOM4E modelling framework.

4.3 3rd year

In the third year, the TAOM4E framework has been designed and implemented on the basis of the requirements discovered during the second year and considering the refined Tropos metamodel. The tool has been tested and released under a GPL license.
4.4 4th year

In the last year of the project the TAOM4E framework has been validated in-field in several case studies and projects. This activities allowed for a revision of some initial requirements that will be considered in the new versions of the tool.
Appendix A

TAOM4E requirements table
<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>Interdependence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>It is not possible for a given plan or goal to take part to more than one decomposition.</td>
<td>Angelo, Loris, Alkazi</td>
<td>change</td>
<td>done - released with version 0.1.5</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A problem of consistency between different diagrams.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The examples of the scenarios can be the following two:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;creation of an Actor 1,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>creation of a goal A in the actor diagram, creation of the goal diagram for Actor 1 and analysis of the goal A</td>
<td>Angelo, Loris, Alkazi</td>
<td>change</td>
<td>bug, improvement</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>in this goal diagram, back to actor diagram, creation of an Actor 2,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>delegation of goal A from Actor 1 to Actor 2, ...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>What's happen to the goal analysis of the goal A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>for in the goal diagram related to Actor 1&quot;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;Ho trovato difficoltà a delegare un goal, primo assegnato ad un attoore, poi ad un dettaglio di analisi successivo...&quot;</td>
<td>Angelo, Loris, Alkazi</td>
<td>change</td>
<td>done - released with version 0.1.5</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1.3</td>
<td>Possible means-end relationships between planning and resources?</td>
<td>Angelo</td>
<td>change</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.4</td>
<td>Create single command stack for all the views</td>
<td>Davide, Alkazi</td>
<td>improvement</td>
<td>done - released with version 0.1.5</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1.5</td>
<td>Metamodel changes:</td>
<td>Anna, Angelo</td>
<td>change</td>
<td>planned</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>a) core metamodel change</td>
<td></td>
<td></td>
<td></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>
<tr>
<td>N=</td>
<td>Description</td>
<td>Initiator</td>
<td>Status</td>
<td>Conclusion</td>
<td>Priority</td>
<td>Complexity</td>
<td>Interdepedency</td>
</tr>
<tr>
<td>------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------------</td>
<td>------------------</td>
<td>---------------------------------------------------------------------------</td>
<td>----------</td>
<td>------------</td>
<td>----------------</td>
</tr>
<tr>
<td>2.1</td>
<td><strong>The appearance, comfort of the analyst</strong></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, Aliaksei</td>
<td>improvement</td>
<td>to do, to think more about design in general</td>
<td>3</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>2.1.2</td>
<td>Make a special “Tropos” 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</td>
<td>4</td>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- released with version 0.1.5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.1.3</td>
<td>Not as evident “has” relationship between actor and goals in actor diagram</td>
<td>Anna</td>
<td>improvement</td>
<td>done - released with version 0.1.5</td>
<td>4</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>comes automatically with visual properties</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.1.4</td>
<td>Multiple dependences 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, Aliaksei</td>
<td>bug</td>
<td>to do</td>
<td>2</td>
<td>2</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 0.1.5</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</td>
<td>Aliaksei</td>
<td>improvement</td>
<td>done - released with version 0.1.5</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>early requirements phase should be created and opened in the editor</td>
<td></td>
<td></td>
<td>Davide says not to do, Aliaksei says to do</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.1.8</td>
<td>The order of the elements in the Actor diagram palette has to be</td>
<td>Angelo, Aliaksei</td>
<td>improvement</td>
<td>done - released with version 0.1.5</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>inverted, elements first, than relationships</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.1.9</td>
<td>AND/OR links: is it useful a label AND OR on the links?</td>
<td>Angelo</td>
<td>improvement</td>
<td>no</td>
<td>2</td>
<td>2</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, Aliaksei</td>
<td>improvement</td>
<td>done - released with version 0.1.5</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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 0.1.5</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>Interdependency</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><em>E' possibile fissare la dimensione dei cerchi per gli attori? Ho notato che si s&quot;arrivano nomi lunghi, si tende a ingrandire il raggio automaticamente.</em></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>2.1.3</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>40</td>
<td>4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2.1.4</td>
<td>Would not be better if we remove the &quot;Actor Diagram&quot; phrase 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 icons and design</td>
<td>4</td>
<td>1</td>
<td>strongly with 2.1.2</td>
</tr>
<tr>
<td>2.1.5</td>
<td>Do we need the &quot;diagram outline&quot; at all?</td>
<td>Alaksei</td>
<td>preposition</td>
<td>yes, we need it, do not change for this moment</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.1.6</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 presenting empty editor and launching previously edited diagrams created - released with version 0.1.5</td>
<td>5</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

### 2.2 The functionality

- **2.2.1** "Save as" does not work
  - Initiator: Alaksei
  - Status: bug
  - Conclusion: done - released with version 0.1.5

- **2.2.2** Increase the number of active points around the figures (change the anchors of the figures)
  - Initiator: Angelo, Davide, Alaksei
  - Status: future feature
  - Conclusion: to do

- **2.2.3** There is a problem in the definition of the names of the diagrams
  - Initiator: Angelo
  - Status: improvement
  - Conclusion: not existent

- **2.2.4** ON the MAC - The diagram scroll is not working well
  - Initiator: Angelo
  - Status: bugs
  - Conclusion: check if it works or not in other GEF editors
<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>Interdependence</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.25</td>
<td>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, Alaksie, Angelo and Loris. The related comment: 'Nella descrizione del modello, ciascun elemento dovrebbe essere meglio collegato con il deponente e il dipendente. Così, l'attuale lista punta dei dipendenti non facilita nel loro riuso per la vista/diagramma successivi. Per esempio, riportando una gerarchia come fatto a livello d'early-requirement. Forse anche la descrizione delle proprietà andrebbe arricchita in tal senso.'</td>
<td>Loris, Alaksie</td>
<td>improvement</td>
<td>to do</td>
<td>2</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>Interdependency</td>
</tr>
<tr>
<td>----</td>
<td>-------------</td>
<td>-----------</td>
<td>--------</td>
<td>------------</td>
<td>----------</td>
<td>------------</td>
<td>----------------</td>
</tr>
<tr>
<td>2.3</td>
<td>Drag &amp; Drop</td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td>6</td>
<td>2.3.2; 2.3.3</td>
</tr>
</tbody>
</table>

2.3.1 Drag and Drop of a goal hierarchies also from the diagram tree. It would be nice to have a possibility to add a number of elements once.

<table>
<thead>
<tr>
<th>Initiator</th>
<th>Status</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Angelo</td>
<td>improvement</td>
<td>% done</td>
</tr>
</tbody>
</table>

2.3.2 Control of what element can be created with drag&drop. I think it should be done in the d&d 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 in the dependency as the goal without the dependency.

<table>
<thead>
<tr>
<th>Initiator</th>
<th>Status</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aliaksei</td>
<td>bug, improvement</td>
<td>partially done, Done for the element part, need more work for the Goal diagram relations released with version 0.1.5</td>
</tr>
</tbody>
</table>

2.3.3 It would be nice to have a possibility to create all the links to the d&d dropped object (like dependencies to the d&d actor)

<table>
<thead>
<tr>
<th>Initiator</th>
<th>Status</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aliaksei</td>
<td>improvement</td>
<td>% done</td>
</tr>
</tbody>
</table>

2.3.4 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.

<table>
<thead>
<tr>
<th>Initiator</th>
<th>Status</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aliaksei</td>
<td>preposition</td>
<td>% done, discussed with Davide, Angelo, Loris and Aliaksei</td>
</tr>
</tbody>
</table>

3 Planned for future

3.1 Mixed diagram! It includes some comments like "E' possibile avere dipendenze esterne al 'ballo' di un goal diagram di un attore? E' importante che il Goal Diagram non sia isolato dal resto del contesto gia modellato."

<table>
<thead>
<tr>
<th>Initiator</th>
<th>Status</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>future feature</td>
<td>% done, discussed with Davide, Angelo, Loris and Aliaksei</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Priority</th>
<th>Complexity</th>
<th>Interdependency</th>
</tr>
</thead>
<tbody>
<tr>
<td>O - max 5 - min</td>
<td>1 - min 10 - max</td>
<td>strongly with 1.2</td>
</tr>
<tr>
<td>N=</td>
<td>Description</td>
<td>Initiator</td>
</tr>
<tr>
<td>----</td>
<td>-------------</td>
<td>-----------</td>
</tr>
<tr>
<td>3.2</td>
<td>Consentire il salvataggio in formati diversi (eps, ps, png, pdf, bmp)</td>
<td>Loris</td>
</tr>
<tr>
<td>3.3</td>
<td>ExportPoint the diagrams of WHOLE the project</td>
<td>Loris, Alkasei</td>
</tr>
<tr>
<td>3.5</td>
<td>Documentation</td>
<td></td>
</tr>
<tr>
<td>3.6</td>
<td>Visual properties and project's properties</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></td>
</tr>
<tr>
<td>3.8</td>
<td>Undo/redo commands for deletion of the elements from the model</td>
<td></td>
</tr>
<tr>
<td>3.9</td>
<td>To think how we can use OCL (Kent OCL plug-in for example) for manipulations on the model (queries, views...)</td>
<td>Alkasei</td>
</tr>
</tbody>
</table>
Appendix B

TAOM4E meta-model diagram

Figure B.1: TAOM4E meta-model main packages
Figure B.2: The Project package.
Figure B.3: The GenDiagram package.
Figure B.4: The ActorDiagram package.
Figure B.5: The GoalDiagram package.
Figure B.6: The GenCore package.
Figure B.7: The InformalCore package.
Figure B.9: The Property package.
Bibliography


David Gallardo. How to create, debug, and install your plug-in, 2002.


