Using Eclipse in building model-driven e-Learning supports

A. Natali, A. Del Cinque, E. Oliva

In this paper we discuss the role of modelling and of EMF (Eclipse Modeling Framework) and GMF (Eclipse Graphical Modeling Framework) in the design and development of supports for eLearning applications. A model driven approach to eLearning applications is presented, rooted on the explicit representation of the model of the didactic content, based on a metamodel language expressed in Ecore.

Besides exploiting the integration of Java, XML and UML provided by EMF, we introduce a Prolog-based representation of models as a more convenient way to overcome the gap between the user level and the technology level and as a mean to reduce the cost of design and implementation of learning strategies.

Working in a MDA perspective, the tool exploits GMF and Jet (Java Emitter Templates) to produce code for a knowledge based, SCORM® compliant Platform Specific Model. The content model provides the Platform Independent Model that allows to achieve the intended behavioural semantics.

1 Introduction

Modelling represents a strategic issue in modern eLearning applications, as regards both the organizations of the didactic contents and the architecture of the application itself. From the point of view of content organization, the specification of a model provides a formal and high level representation of the building blocks of a didactic unit and a mean to highlight the logical relationships between the parts that compose a course. From the application point of view, the current vision of eLearning as an evolutionary process is better supported by using an explicit model as the basic artefact to control and support the process itself.

Modelling can also promote discipline in design and development of eLearning platforms and tools, with particular reference to authoring tools capable to interact with eLearning supports and to be modified in a systematic way according to the evolution of the eLearning field.

Moreover, the usage of models allows to make explicit knowledge that usually remains implicit into eLearning supports; such a knowledge can be used as a powerful bridge between the design phase of a course and the run time phase; especially when the *content views* are to be personalized and individualized according to the characteristics of the reader,.

A systematic, model-driven approach to the design and development of eLearning courses, supports and tools has been tackled in the `AlmaTwo` [1] project of the University of Bologna, co-funded by the Region Emilia Romagna. The main aim of `AlmaTwo` is to integrate in a systematic and well-founded way pedagogical approaches to eLearning with the tools and supports that the technology makes available. Our intent is to promote the vision of *eLearning as a process* based on student *activities* performed in the context of learning plans and characterized by the cooperative work of human and artificial agents. The current logical model of `AlmaTwo` applications is expressed in `UML` and is reported at the site http://137.204.107.78/moodle/ (login as guest).

In this paper we concentrate the attention on the part of the model related to the organizations of eLearning contents and on the role of this model to support advanced features at application level.

However, the focus of the work is not on applicative features; it is on discussing the crucial role played by Eclipse Modeling Framework (EMF) [2] and Eclipse Graphical Modelling Framework (GMF) [3] in supporting our vision; moreover, we will discuss our choice to introduce a new representation for models, based on Prolog clauses.

Eclipse has been used not only as a powerful production tool, but also as a conceptual reference for the design and development of (a new class of) authoring tools and possibly of run time supports. eComposer is the tool we developed with GMF to support the teacher in defining the structural model of the content of a course in a way systematically related to the desired application model. The focal point was to make explicit the representation of the metamodel application language, and the constant alignment with the actual application code, according to the Model Driven Architecture (MDA) [4] approach. By layering over GMF, we naturally chose to satisfy functional and non functional requirements according to two basic strategies: MDA and declarative programming.

The Prolog-based representation of models is introduced to further enhance declarative programming and to promote the interpretation of a model as a knowledge base (modelKB) ; the modelKB is exploited to overcome the gap between the technology level (e.g. the XML representation of the model) and the user level.

The work is structured as follows. In section 2 and 3 we introduce respectively an overview of eLearning applications and an introduction of the AlmaTwo approach. Section 4 is devoted to the description of the metamodel language and the role of EMF in producing an editor. Section 5 gives an example of the author workflow and a view of the final result. In section 6 we discuss the role of GMF in producing a user oriented GUI interface for the editor and the possibility to adopt a similar approach for the organizations of the eLearning application itself. Section 7 is devoted to conclusions.

2 Overview of the application field

A eLearning application system can be viewed, in first approximation, as a client-server, distributed Model-View-Controller (MVC) application, in which the model, stored at the server site, is related to a eLearning course made of *learning objects* (LO) [5] that constitute the didactic content and the views, running on the client site, are provided with the support of a browser.

2.1 The technologal view

ELearning applications run on platforms provided by Learning Management Systems (LMS). A modern LMS usually implements the SCORM® (Sharable Content Object Reference Model) [6] standard that defines rules to represent the content structure (*Content Aggregation Model*) and rules to handle the information that can be exchanged between the client and the server (*Run Time Environment*). The aim of SCORM® is to enhance *interoperability*, by assuring that the content can be read and handled by any SCORM® compliant LMS. The goal to promote a conceptual space for eLearning design is out of the scope of SCORM®.
The SCORM® model of the content is expressed in XML and stored in the imsmanifest.xml file, which must be included in the (zip) file used to deploy a course on the chosen LMS. Such a deployment file must contain the description of the learning objects, their metadata and all the required learning resources. The SCORM® model of the data that can be dynamically exchanged between the client and the LMS concur to form a *lesson data model* which represents the student and the status of the student activity with respect to a specific learning object. The SCORM®2004 specification [7] extends SCORM®1.2 with particular reference to content *navigation rules*.
The run time behaviour of a eLearning application is the combined result of the actions performed by the student through the content views (both the views provided by the LMS and those provided by the content itself) together with the behaviour embedded in each learning object, under the constraints imposed by SCORM® and by the planned navigation rules.
Since content (re)factoring implies the production/modification of learning objects and the definition of a new XML model of the content, the teacher is usually assisted in the course production phase by an *instructional designer*, in order to overcome the gap between the design level and the implementation level. This approach however increases the costs (in money and time) of eLearning production and discourages teachers to directly control, exploit and improve the potential benefits of eLearning.

2.2 The logical view

The designer of eLearning applications can be considered at first glance as a special kind of software designer which writes "programs" not only for machines but also for human beings. A eLearning course and its composing LO should represent the final products of a design process devoted to promote learning by driving the activities of students in synergy with the behaviour planned for the machines. However students cannot be reduced to machines and student activities cannot be reduced to the execution of simple commands or to the reading of contents written in electronic form.

Since the programmed behaviour of machines should be conceived to stimulate student actions directly related to learning, great attention is given to constructivist approaches [8] in which students are viewed as *co-creators* of their own knowledge. However, the methodology and the support required to plan (predict), control and validate "true" learning actions in a way closely related to the structure of a course constitute still an open problem.

3 The AlmaTwo approach

One of the goals of AlmaTwo is to support a process of content production based on a systematic, model-driven relationship between the structure of a course intended as an eLearning application and the expected behaviour at application level.

To achieve this goal we adopt a MDA approach by introducing a *Platform Independent Model* (PIM) of a eLearning application and a systematic relationship to a specific *Platform Specific Model* (PSM). Actually the PSM of reference is based on the SCORM® and Web specification as implemented in an extension (called AlmaLMS) of the Moodle 1.7 platform [9] .

The content production process is supported by eComposer, a configuration tool built using GMF and Jet [10] to perform automatic generation of code for our target machine.

The vision supported by eComposer is that the author should be mainly concerned with the *logical organization* of the content of the course (application), by leaving to the tool the task of introducing the stuff necessary to support the intended behavioural semantics. The intent is to increase teacher's control over the design of the course, by limiting the role of the instructional designer to very specific technological aspects.

The behavioural semantics associated to the model of the content is rooted in the concept of *action* performed by the student on the grounds of the content itself within the support and the constraints of the chosen LMS.

A course specification is viewed as the specification of a working plan in a partially known environment. At course design time, the teacher acts as the planner of the actions that students will perform by using the views of the content provided by mechanical agents.

Relevant learning activities are usually directly related to the actions promoted by specific LO; it is for example the case of a LO embedding a virtual lab or an interactive LO that stimulates the student with questions, experiments, etc.

By recognizing these points, AlmaTwo introduces a conceptual space including the following concepts:

- the idea of *learning artefact* [11] - inspired by the *activity theory* [12] - to denote a LO properly designed to promote activities (beyond simple select-open-read loop) considered useful for learning;
- the concept of *pedagogical type* [13] to characterize each LO or artefact in terms of the *learning strategies* it can promote. A set possible of pedagogical types is reported in (Tab1);
- the concept of logical *relations* between LO, in order to support the definition of *conceptual maps* of the contents; possible relations are discussed in section 4.1.1;
- the concept of *learning path*, intended as planned proposals for navigation in the content.

Table 1: Pedagogical types (provisory from [13])

These concepts constitute the basic building blocks of the meta model discussed in section 4. Educational metadata and lesson data model are also introduced to improve the expressive power of the metamodel and to create a pragmatic link with SCORM® compliant platforms.

By modelling a course with these concepts, we can interpret a structural specification - i.e. the logical organization of a course - as a behaviour plan. The logical organization of a course expressed by using the metamodel language can be considered itself as an artefact that can help students in becoming co-creators of their own knowledge. In particular, conceptual maps are artefacts that eComposer is able to build in a

personalized way, by taking into account the values of the metadata and also of the lesson model. At run time, a map gives to students the capability to navigate with relative freedom into the contents by providing a constant reference point for orientation. The map is also an artefact that can be build by the students themselves to create a personal view of the body of knowledge. The comparison between the map planned by the teacher and the map built by the students can provide useful feedbacks to validate the pedagogical correctness of the course model and to reduce the uncertain on the environment.

3.1 The author workflow

The workflow of design and production supported by `eComposer` can be summarized as follows:

- The author defines a `Unit` as an aggregate of logical contents called `Item` and physical resources called `File`. A `File` that can be written in any format that can be handled by a conventional browser (e.g. `xml, html, pdf, rtf`, ..)
- The `Items` are organized in a hierarchical tree having `Unit` as its root.
- Each `Item` can be associated to one or more `File` that represent its content.
- Each `Item` can be associated to a (empty) set of *metadata*, to one (and only one) *pedagogical type*.
- Each `Item` can be associated to a pre-condition and a post-condition.
- Each `Item` can participate to a (empty) set of binary *relations* with other `Items`.
- The author can define one or *more learning paths* (see section 4.1.2).

The intended semantics is that `Unit` represents a course to be followed by a set of students. To follow a course, each student must select an `Item` (whose precondition is true) in the context of one or more learning paths. The pedagogically relevant actions performed by the students are implicitly or explicitly related to the selected `Item` in the context of the selected learning path.

4 The role of EMF

In our `MDA` approach, the `PIM` is mainly represented by the course model, which is a datamodel obtained as an instance of a meta model defined by using `Ecore` [14]. `Ecore` is the core metamodel implementation in `EMF`, which is an implementation of the `OMG` Meta Object Facility (`MOF`) specification [15]. `Ecore` allows us to express abstract language syntax categories, their attributes, and relations between them (associations, compositions and generalizations). In our metamodel language we express mainly relations and compositions of e-learning content; `Unit` is the root of the language abstract parse tree.

4.1 The metamodel

Since each `Ecore` sentence can be represented as a kind of `UML` class diagram, the `UML`-like representation of a simplified version of the `AlmaTwo` metamodel language is shown in (Fig. 1) (shaded classes are abstract):

Figure 1: Language metamodel (subset);

The metamodel defines in a more formal way the fact that a `Unit` is an aggregate of `1` or more `Item`, `1` or more contents (`ContentOfItem`), `0` or more relations (`ItemRelation`) between items and `1` or more `LearningPath`. Each `ContentOfItem` makes reference to `1` or more `File`. The information carried by an instance of this meta model includes what can be expressed in the manifest file of `SCORM®1.2`; main extensions are the concepts of item relation, learning path and pedagogical type.

4.1.1 Relations between items

The concept of relation between items has been introduced mainly to allow a teacher to specify logical relations betweens the parts of a course and to allow a representation of the content in terms of *conceptual maps*. The usage of conceptual maps – instead of conventional tree-based indexes – should help the reader in capturing the logical organization of the course and in providing support for semantic-based navigation. The set of relation types is defined by a set of classes that specialize the class `ItemRelation`; if we denote with `IT` the item of interest (source `Item` of the relation) and with `A` another item (target `Item`) then possible relations are:

- `preknowledge`: `IT` assumes that the reader knows what is written in A
- `clarification`: A should make the content of `IT` more clear;
- `conceptualize`: A presents the content of `IT` is a more formal and conceptual way;
- `widening`: A is a study in depth of `IT`;

- experiment: A is an experiment related to IT;
- exercise: A is an exercise related to IT;

Other relations can be introduced by simply adding new classes.
An ItemRelation can be associated with one or more Condition involving item metadata values and lesson model values; in other words the concept of relation is not absolute, but can depend on data values.

4.1.2 Learning paths

A learning path makes reference to 1 or more Item; it is a reading sequence suggestion related to some learning goal (e.g. achieving theoretical background rather than practical skill). By defining different learning paths the teacher give to students the opportunity to follow different, specialized workflows. This specification is actually a placeholder, waiting for the implementation of SCORM®2004-SN navigation rules.

4.1.3 Pedagogical types

Pedagogical types are at the moment modelled as enumeration types. Each pedagogical type implicitly defines a set of learning strategies that must be supported with the help of mechanical agents. Since the definition of these strategies is crucial for learning and requires the contribution of pedagogists, learning strategies are expressed as much as possible in a high level, declarative language, as discussed in section 4.3

4.2 From the model to the editor

Once defined a metamodel specification, EMF can take such a definition and produce a good, easily customizable Java implementation for it. Moreover, EMF.Edit allows us to produce an editor that will display instances of the model using standard Eclipse JFace viewers and a *property sheet*, it provides also a set of generic *commands* to modify EMF models, with unlimited undo/redo.
EMF is not just a generator tool; it is also a powerful runtime that unifies three important technologies: Java, XML and UML. A EMF model can be defined using either a XML schema or a UML diagram or a set of Java interface; regardless of how the EMF model is provided, the power of the framework and generator will be the same.

4.3 From the model to the application

The possibility to conceive Java, XML and UML as different technologies to build a different representation of the same model is very important for the software developer. But the usage of a (meta)model is not only a means to improve the software production process; it is also a way to enhance communication among people and to highlight the conceptual space underlying a software application.
For this reason we have introduced yet another representation for a model: a textual representation based on Prolog [16] syntax.
In fact, we believe that a teacher can better read and understand a text written in clausal form rather than a UML diagram. This does not exclude the usage of graphical tools based on UML-like notations; we ourselves have defined one of them, discussed in section 6. The textual representation should simply help teachers in getting more involved in the definition of pedagogical strategies and in sharing a more conventional language with the instructional designer (if still necessary).

4.3.1 Relations between models

eComposer exploits Jet to provide two transformers: one (EcoreToKb) that builds the Prolog representation by taking as input the object Ecore internal representation; and one (KbToEcore) that (re)builds the Ecore representation from the Prolog representation.
In the MDA perspective, the usage of Jet is a shortcut to avoid the explicit representation of a platform specific meta model related to Prolog clauses (PSPM) and an explicit mapping between the PIM and PSPM. The drawback is of course that the Jet-based transformers have to be changed whenever the PIM changes.
Our feeling is that, as teachers become more expert, they can give preference to the textual representation, in particular to extend or modify the model; in this case the KbToEcore transformer will allow them to check the correctness of the work and to automatically produce the manifest imsmanifest.xml file and all the other resources to be included in the deployment file.

5. An example

We report here a simple example of a top-down specification of the model of an introductive course to AlmaTwo. The following figure

shows the logical part the model built by using eComposer:

Figure 2: A simple course model (logical part)

The model should be quite clear for a software developer; however teachers of human disciplines could feel more comfortable in reading a text. In the textual representation that follows each clause can be viewed as the representation, in the concrete syntax of Prolog, of a phrase of the abstract language defined by the metamodel.

We start by giving to the course the name LearningAlmaTwo and by specifying the items that compose it.

```
courseName("LearningAlmaTwo").

item( intro, "Introduction",  loType1 ).
item( almaTwo, "AlmaTwo",  loType3 ).
item( elPaths,"Learning path", loType2 ).
item( mmodel, "Metamodel", loType3 ).

preCondition( almaTwo, 'intro=completed').
```

For each Item we specify a name (to be used as an internal identifier), a label (to be used as a visible name) and the related pedagogical type. For the item AlmaTwo a precondition is specified (in the SCORM® syntax) .

Now we associate to each item a non empty set of standard educational metadata

```
meta( intro, 'SCORM1.2',
   educational, difficulty, 'EASY').
meta( intro, 'SCORM1.2',
   educational, semanticDensity, 'LOW').
meta( almaTwo, 'SCORM1.2',
    educational, difficulty, 'VERY_EASY').
meta( elPaths, 'SCORM1.2',
    educational, semanticDensity, 'LOW').
meta( mmodel, 'SCORM1.2', educational,
    difficulty, 'VERY_DIFFICULT').
```

In the next step, we state some logical relation between the items.

```
itemRel( preknowledge, elPaths, almaTwo).
itemRel( widening, intro, mmodel ) :-
  metaData( intro, semanticDensity,V ),V<=2.
itemRel(conceptualize, almaTwo, mmodel).
```

The item almaTwo is considered as *preknowledge* for the item elPaths; mmodel is considered a conditioned *widening* for intro; the condition is based on the value of the metadata semanticDensity associated to intro. The item mmodel *conceptualizes* the item almaTwo .

To make thinks simple, we plan now a single learning path, called book, since it should provide a book-like reading of the content:

```
path( book ).
path( book, intro, 1 ).
path( book, almaTwo, 2 ).
path( book, elPaths, 3 ).
path( book, mmodel, 4 ).
```

Now the specification of the logical organization of the course is completed. At the end, we introduce the specification of the files that store the content:

```
itemContent( intro, "Intro1.html").
itemContent( intro, "Intro2.html").
itemContent( almaTwo, "almaTwo.html").
itemContent( elPaths, "LearnPaths.html").
itemContent( mmodel, "Metamodel.pdf").
```

Note that to the item intro we have associated two files written in html, while to the item mmodel we associate a single pdf file.

From this input model, eComposer can build the learning resources necessary to our PSM machine, create the SCORM®1.2 manifest, and the zip deployment file.

5.1 The visible result

In (Fig. 3) we report a view of the final application running on a Moodle platform.

Figure 3: A moodle -based view

The figure shows (in the leftmost frame) the index provided by the LMS; the item currently selected is AlmaTwo (since the item intro has been completed). The frame in the middle is built by the envelope (see section 6.2) created by eComposer for the pedagogical type loType3; such an index shows that the current learning path is Book and that the item is composed of two pages (one for each content file). The table represents the logical map related to the item; the view (e.g. the map colour) depends on the value of the metadata difficulty. At the bottom of the frame there are the buttons to give commands to change the lesson status and to access to the internal indexes.

6 Towards model driven run time support

To obtain the GUI of eComposer, (shown in fig. 4) we followed the GMF workflow based on the definition of two new models: a *graphical model* that defines figures, nodes and links to display and a *tooling* model that defines a palette for the selection and drag of language constructs. Because these models are independent of the domain application model (and possibly reusable for several domains) we defined also a *mapping* model that realizes the mapping between the business logic (the meta language of section 4) and visual model (graphical and tooling definition). After this mapping, GMF can build a *generator model* from which an Eclipse plug-in is produced through code generation and compilation.

Figure 4: eComposer GUI

This model-driven approach promoted a smart software development cycle that helped us to face a critical aspect of any user-oriented tool: identify variation points related to different aspects, achieve immediate feedback from the user, and build in a short time a different release of the tool. But modifiability extendibility, rapid prototyping, user feedback and re-factoring are critical aspects for eLearning applications too and they should not be limited to the production of offline tools. The main problem in exporting the model driven approach to the application field is that the standardized part of LMS does not provide a reference framework comparable with the Eclipse framework.

Nevertheless we tried to follow the main principles of the model driven approach also at application level by looking at the Web as our reference framework and by adopting an interpreted approach instead of a compiled one.

The run time has been designed to support a different form of editing (called *extension-editor*) that provides personalized views of the model defined by the author and allows a limited set of editing actions only.

6 .1 Supporting maps and annotations

In conventional eLearning applications, the course model - in the imsmanifest.xml form - is used by the LMS to build a tree-based index; such an index is the main course view and allows students to navigate into the course content, according to the preconditions, if any, associated to each item.

From a technical point of view, each *learning path* is a specialized form of this kind of index; its implementation requires a strong relationship with the specific eLearning platform. that we provide trough a predefined LO (index.html) which is automatically included in the content of each course

Besides index-views, our run time support provides also a map view for each LO whose pedagogical type is not simply *reproductive* (i.e. of level higher than loType1). Map views not only situate each item into a semantic perspective, but are also used as a mean to allow students to build their personal logical organization of the body of knowledge. No modification to the original model defined by the teacher is allowed; instead we allow the constructions of news set of relations.

Strictly speaking in the MDA perspective, also the creation of new relations should be forbidden, since it involves a modification of the application defined by the author. However the possibility that students can act as co-editors of the course model is essential to give them the role of co-constructor of their knowledge; the important point is that that model provided by the teacher can always be used as a reference point and that there is a common meta language.

In the same spirit, our run time supports the creation/modification of content *annotations.*

All these features are provided by a run time framework built around the model of the course expressed in the Prolog form.

Rapid prototyping of critical aspects, such as the implementation of learning strategies, is supported by exploiting interpretation and declarative programming in Prolog.

6.2 Usage of the Prolog model at run time

The Prolog representation of the content model provides a *knowledge base* (modelKB) that constitute the bridge between the design phase and the run time support, i.e. the AlmaLMS machine. While most of the (SCORM® compliant) functionalities of AlmaLMS are inherited from Moodle, the knowledge based features of our platform are produced by the EcoreToKb transformer.

These features are automatically added to the content by eComposer; that embeds each content file into a HTML *envelope* file (written in automatic way) that acts an adapter between the content and the run time support.

The bridge between Prolog and the web technologies is done by javascript with the help of an Applet written in tuProlog. tuProlog is an opensource project [17] of the Alma Mater University of Bologna that provides a Prolog interpreter, built in Java and interoperable with Java (and therefore with javascript). The modelKB representation of the course model is downloaded on the client by the HTML envelope each time an item is selected. Any extension to the model is saved on the server through HTTP requests with the support of AJAX [18].

7 Conclusions

eLearning applications, that cannot exist without ICT technologies, are actually requiring better attention to pedagogical aspects and to the suitable way to exploit technology to improve learning and teaching. However, the cooperation between pedagogists and engineers cannot be based on a master-slave approach (whatever be the master and the slave); rather a more deep cooperation is necessary, based on common goals, and a shared conceptual space.

In this perspective, the model driven approach to application building can be strategic not only form a software production point of view, but also in order to create a common reference language, usable both by experts in technology and by experts in pedagogy.

In this paper we have discussed an approach of this kind, in which EMF and GMF have proven to be mature enough to support smart production processes built around formal specifications, that resulted useful also in a very pragmatic way.

Although GMF is still lacking in the documentation, the usage of Eclipse framework gave us the opportunity to build in a short time not only a tool able to fully automate the production process of a SCORM® compliant course, but also to delineate and support a new conceptual space. The possibility of refactoring the tools by introducing changes at model level rather than at code level, is fundamental, in this evolutive phase of eLearning applications, not only to improve the software production process, but, and most importantly, to create an effective operational bridge between two cultures.

References

1. ALMATWO, Documento tecnico di dettaglio, 2002,
2. BUDISNSKY,F, STEINBERG D., MERKS E., ELLERSICK R., GROSE T., *Eclipse Modeling Framework*,, Addison Wesley 2004
3. ECLIPSE GMF http://wiki.eclipse.org/GMF_Documentation_Index,
4. KLEPPE A., WARMER J., BAST W., *MDA explained*, Addison Wesley. 2003
5. WILEY,D. A.: *Learning object design and sequencing theory* http://opencontent.org//docs/dissertation.pdf
6. ADL, SCORM, http://www.adlnet.gov/scorm/
7. ADL, SCORM2004, http://www.adlnet.gov/scorm/20043ED/Documentation.aspx
8. DUFFY,T. M. CUNNINGHAM,D. J. : *Constructivism: Implications for the design and delivery of instruction*., D. H. Jonassen (Ed.), Handbook of research for educational communications and technology, 1996
9. MOODLE, http://docs.moodle.org/en/Main_Page
10. JET, http://www.ibm.com/developerworks/library/os-ecemf2/
11. NATALI A., DE COI J., *Learning Artefacts, modello dei dati e di elaborazione*, AlmaTwo report, 2005
12. ACTIVITY theory, http://carbon.cudenver.edu/~mryder/itc_data/activity.html
13. GUERRA, L., PACETTI E., FABBRI M., *Documento contenente le indicazioni relative allo sviluppo di dei LO dal punto di vista metodologico ed alla loro implementazione in diversi setting formativi*, AlmaTwo report, 2005
14. ECORE, http://www.eclipse.org/modeling/emf/?project=emf
15. MOF, http://www.omg.org/technology/documents/formal/mof.htm
16. O' KEEFE, R., *The Craft of Prolog*, The MIT press, 1990
17. tuPROLOG user's guide, http://www.alice.unibo.it/tuProlog/
18. http://it.wikipedia.org/wiki/AJAX