Journal of Network and Computer Applications 35 (2012) 1086–1105

Contents lists available at SciVerse ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

A model-driven development method for collaborative modeling tools

Jesús Gallardo [a,*], Crescencio Bravo [b,1], Miguel A. Redondo [b,1]

[a] Departamento de Informática e Ingeniería de Sistemas, Universidad de Zaragoza, Escuela Universitaria Politécnica de Teruel, Ciudad Escolar s/n, 44003 Teruel, Spain
[b] Departamento de Tecnologías y Sistemas de Información, Universidad de Castilla-La Mancha, Escuela Superior de Informática, Paseo de la Universidad 4, 13071 Ciudad Real, Spain

ARTICLE INFO

Article history:
Received 12 May 2011
Received in revised form
27 October 2011
Accepted 5 December 2011
Available online 22 December 2011

Keywords:
Collaborative modeling
Network modeling environments
Model-driven development
Groupware
Computer-supported collaborative work

ABSTRACT

Collaborative modeling tools are useful for many tasks within design or learning processes. However, they are difficult to build and are usually domain-specific. In response to this situation, we propose a model-driven method for the development of domain-independent collaborative modeling tools. This method consists of a methodological framework, a conceptual framework and a technological framework. The methodological framework defines the phases to be carried out when applying the method, whilst the conceptual framework is made up of the meta-models used in the method and the transformation processes established between them. Finally, the technological framework consists of the integration of some plug-ins from the Eclipse Modeling Project with some add-ons which provide collaborative functionality. Some case studies that exemplify this development method for specific domains are explained in detail, together with comparisons with similar approaches. Thus, an initial evaluation of this approach is provided and some advantages over those other approaches are identified. A further evaluation process in the form of an empirical study of use has also been carried out. Hence, the method proves to be useful for any user who does not have advanced knowledge of groupware programming and who needs to use a collaborative modeling tool in his/her work. Moreover, each framework implies a contribution that can be used in different contexts.

1. Introduction

Today's society has an increasing need to communicate and collaborate in the many different tasks carried out in the workplace or in learning or leisure spaces. This requires a new type of software designed for groups, which is known as groupware. The design of groupware tools is an emerging field of Software Engineering, and is gaining in importance (Gerosa et al., 2005). Aspects such as network management, synchronization, awareness and shared workspaces are new concepts which have been introduced with groupware. Unfortunately, groupware is more difficult to design and evaluate than a non-collaborative application since, among other reasons, social protocols and group activities must be taken into account (Grudin, 1993). As such, groupware development needs specific techniques that take into account the aforementioned particularities of this type of software.

In this study, we tackle the problem of building collaborative tools to support modeling tasks. We focus on a specific type of groupware; that of domain-independent modeling tools. In these tools, several distributed designers typically interact to construct a model or artifact, working in shared workspaces which are based on the metaphor of a whiteboard. The model is built according to a specified goal or task. What is unusual about the kind of tools we are dealing with, i.e. domain-independent tools, when compared to other modeling systems covered in the literature (see Section 2), is that the design (model) to be created is not restricted to a specific domain; that is to say, the tool is able to deal with diverse design scopes described by means of a configuration process (Greenfield, 2005). This is in sharp contrast to domain-specific tools, which allow for the building of diagrams within a specific domain, such as those of digital circuits, concept maps, any kind of UML diagrams, etc. Furthermore, these tools could be made up of different workspaces in order to carry out tasks other than the modeling task itself. An example of this could be a task for arranging the work that is to be done among the different users, or a task for doing some sketches prior to the modeling task.

Thus, the research hypothesis we pose here states that it is feasible to provide systematic support for the development of such systems. This is addressed in this study with the proposal of a full model-driven engineering method for the construction of domain-independent modeling groupware tools. Within this method, the use of software engineering techniques such as meta-modeling is very useful. With meta-modeling, we define the structure of the models that are to be built using the modeling tool, as well as the structure of the tool itself, so that meta-models

* Corresponding author. Tel.: +34 978 61 81 02.
E-mail addresses: jesus.gallardo@unizar.es (J. Gallardo), Crescencio.Bravo@uclm.es (C. Bravo), MiguelRedondo@uclm.es (M.A. Redondo).
[1] Tel.: +34 926 29 53 00; fax: +34 926 29 53 54.

1084-8045/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.jnca.2011.12.009

become the basis for building the models that characterize a specific tool (Gallardo et al., 2007). Our method, therefore, makes it possible to work with the same modeling tool in different domains. Take for example a software development company which wants its employees to carry out a discussion using both a flow chart and a class diagram. With our method, this could be done using the same tool, which will be adapted on each occasion to each specific domain.

The development method is intended for users with no advanced knowledge of software programming; the process for defining new application domains is simple and descriptive, and the generation of the final groupware tool is guided by the tools supporting the method. Nevertheless, users can be helped by experts in the application domain, or software engineers if necessary.

Our model-driven development method is based on three frameworks: a methodological framework, a conceptual framework and a technological framework. The methodological framework consists of a series of phases that must be followed by the non-expert user who wishes to develop a collaborative modeling tool. These phases are: (i) the identification of the domain, (ii) the modeling of the domain and the workspaces, (iii) the production of the collaborative modeling tool, which includes the model transformations and the generation of the tool itself, and (iv) the use of the generated tool. The conceptual framework is made up of the models that are used in the meta-modeling process. These models are mainly the domain and workspace meta-models. And, finally, the technological framework consists of a series of plug-ins for the Eclipse platform that have been modified and extended to generate collaborative applications. With these frameworks, comprehensive support for the entire development method is provided.

Our first attempt at building a domain-independent collaborative tool was the SPACE-DESIGN tool (Gallardo et al., 2011b). SPACE-DESIGN is domain-independent, since the tool reads the domain specification from an XML file and spawns the corresponding user interface to carry out the modeling process. This tool includes some fixed awareness and coordination mechanisms implemented as reusable components: tele-pointers, a session panel, a floor control tool, an interactions list and a structured chat tool. In the field of groupware, awareness is defined as the perception of the group and its activity (Dourish and Bellotti, 1992). The present work is an evolution of that done with the SPACE-DESIGN tool and improves certain elements, such as the way in which the domain is specified, the extension of the working process to cover new workspaces, and the reusability and extensibility of the generated tool. Some other ways of carrying out modeling tasks in a collaborative way have been considered (sharing a non-collaborative modeling tool, for example) but they were all ruled out for one reason or another, as explained later.

This article continues by presenting the foundations of collaborative modeling tools and, in particular, of domain-independent tools. In the third section, we explain the concepts and models identified in the meta-modeling process. The fourth section describes in detail our development method for collaborative modeling tools. Then, we discuss the application of our proposal in some case studies. Section 6 is about the empirical study of use carried out to test the suitability of our approach. Finally, in Section 7, we draw some conclusions and outline future work.

2. Related work

2.1. Domain independence in groupware tools

Of the various types of groupware tools, this paper focuses on distributed synchronous tools to support the construction of a model made up, at a conceptual level, of a set of objects and the relationships between them. Designers work in a shared workspace, where they create a model collaboratively. To achieve this, users participate in design sessions in which they make use of domain objects, relationships and other graphical elements found on the workspace toolbars. Typically, the final model is built in response to a specific goal. The setting may be a group work activity, where the problem is a real situation to be dealt with in the scope of a company or institution or an e-learning system, where a learning method based on collaborative problem solving is followed (Bravo et al., 2006). As mentioned above, the whole system can be made up of other workspaces apart from the modeling one in order to give support to other steps of the design process.

The second basic characteristic of the modeling tools we are looking at is that they are independent of the domain that is being considered. At this point, we can define a domain as the particular syntax and semantics that are used to construct models. In such tools, therefore, the domain is not fixed but instead can be defined by the user employing suitable authoring tools.

These domain-independent modeling tools present some very typical problems. These include the materialization of the shared workspaces, floor control policies, coordination and communication processes, and the definition of the domains. In the following paragraphs, we discuss some examples of tools related to this study, some of which are domain-independent and others which are domain-dependent.

Cool Modes (Pinkwart et al., 2002) is a cooperative modeling tool in which several users work together in order to design a model or artifact. It includes a set of plug-ins containing the elements that can be placed on the whiteboard. The strong point of the tool is that the constructed model can be simulated, since the plug-ins provide this functionality. However, the definition of plug-ins is not very versatile, since they are programmed within the code itself.

A similar tool is Synergo (Avouris et al., 2004), which has several fixed palettes containing design components that can be connected to each other. Designs are stored in a proprietary format, whilst the different modeling actions are stored in an XML file which can be analyzed later. Another feature of Synergo is its communication tool (chat feature), which allows users to have discussions amongst themselves.

There have been a few attempts at developing collaborative modeling tools using the plug-ins included in the Eclipse Modeling Framework (EMF), which is the technological approach we have followed. One of those attempts is the Dawn project (Flügge, 2010). However, the tools generated in this project do not have any awareness or collaboration support elements, and they are only made up of the modeling workspace. These are the main differences between Dawn and our work, which gives great importance to awareness and collaboration support.

In contrast to the described tools, we can also mention some examples of domain-dependent modeling tools, which are conceived and designed to work in a specific application domain. DomoSim-TPC (Bravo et al., 2006) is one such tool, which works in the Domotics domain. Concretely, DomoSim-TPC is a modeling tool which has a shared workspace in which the users synchronously build a model as a solution to a previously defined problem. DomoSim-TPC includes a number of awareness and discussion elements: a session panel, an interactions list, a structured chat tool and a decision-making tool. Another final example is Co-Lab (van Joolingen et al., 2005). Co-Lab is an environment designed for synchronous collaborative inquiry learning that works within the System Dynamics domain. An important difference between Co-Lab and other modeling tools is that Co-Lab uses the metaphor of a building, wherein each building represents a course in a specific domain. Co-Lab also features a number of awareness mechanisms,

although in this tool only one user at a time can edit within the workspace.

An initial conclusion that we can draw from the above discussion is that we can see that the existing domain-independent collaborative modeling tools do not have as much flexibility as we would like. The design palettes are programmed into the code in these tools, preventing end users from extending the functionality by defining new domains. The solution to this problem that we put forward here is the use of meta-modeling and formal specifications to define the domains, with authoring tools being used to make this work easier for end users.

A second conclusion is that domain-specific tools have many more awareness, communication (chat) and coordination mechanisms than domain-independent ones. The effort put into obtaining domain independence seems to be the reason why, in other aspects of the tool such as awareness, some functionality is lost. In the proposal put forward in this article, therefore, domain independence works together with the use of a wide spectrum of awareness, communication and coordination mechanisms. In the area of 3D collaborative modeling tools we can also find a number of groupware systems. NetSketch (Laviola et al., 1998) and WWW3D (Snowdon et al., 1997) are two examples. The underlying complexity of the manipulation of 3D models and the distribution of these actions to the users in a group makes these systems very domain-specific tools, as happens in the cases of architecture, town planning or mechanics domains.

An alternative approach to implementing these kinds of modeling tasks would be the use of a non-collaborative modeling tool (e.g. Microsoft Visio, which uses so-called *stencils* to allow the tool to be extended) in combination with a shared windows system (e.g. NetMeeting). This would be a strict WYSIWIS (What You See Is What I See) situation, with tight coupling and without any awareness mechanisms. Therefore, whereas in groupware tools each user executes an application instance that can be used and configured according to his preferences, in a shared windows system a single-user application is replicated on the computers of all of the users; that is to say, there is only one application instance.

2.2. Developing modeling tools with a meta-modeling approach

Recently, building software, and particularly groupware, has become an increasingly expensive and complex task due to, among other things, the increasing complexity of the tools to be developed. These tools try to take advantage of the new functionalities and software/hardware platforms that are emerging. Over time, many authors have realized how difficult the task of developing groupware is (Roseman and Greenberg, 1996; Schuckmann et al., 1996). In order to make this task easier, one paradigm that is gaining in importance is that of Model-Driven Engineering (MDE) (Favre, 2004), also known in the scope of computer science as Model-Driven Software Development (MDSD). One of the main goals of MDE is to bridge the conceptual gap between the problem to be solved and its software implementation, as well as to facilitate migration between platforms with the reusability of developed code and to keep developers separate from the complexities of the implementation platform (France and Rumpe, 2007). We are going to adopt an approach based on the definition and transformation of models for supporting the development of modeling tools, in contrast to the tools described in Section 2, which have been developed *ad hoc* following classic software development methods.

The MDE paradigm is based on the use of models for the construction of software. A model is a computable representation in which each element of the representation corresponds to an element in the domain (Cook, 2004). In order to work with models, MDE defines two concepts (Schmidt, 2006): domain-specific languages, and transformation and generation engines. Domain-specific languages (DSL) are languages designed to represent aspects of a given domain. These languages are defined by means of meta-models, which are models that define the structure of other models. Usually, a DSL is made up of an abstract syntax, a concrete syntax and its semantics. As for transformation and generation engines, these are any piece of software that, from one or more models, generate other models or source code. Many researchers have found applications for this paradigm in diverse fields and domains, such as Feature-Oriented Programming (Trujillo et al., 2007), GUI (Graphical User Interface) development (Giraldo et al., 2008) or analysis of users' activities in collaborative systems (Duque et al., 2011).

As previously mentioned, model transformations are the basis of the MDE paradigm. Two kinds of model transformations exist: M2M (model to model) transformations and M2T (model to text) transformations. M2M transformations generate one model from another. They can be vertical transformations if they take place between models at different levels of abstraction, or horizontal transformations if they take place between models at the same level of abstraction. As for M2T transformations, they take care of generating code from a specific model.

Another approach to MDSD is that defined by the Eclipse Foundation in the Eclipse Modeling Project.[2] The project is formed of a series of plug-ins that provide support for the different steps in the meta-modeling process. The main one is the Eclipse Modeling Framework (EMF), which features technologies for model creation and transformation. EMF defines its own meta-meta-model, which is called Ecore and is composed of objects called *EClasses* and relations called *ERelationships*. As for transformation languages, Eclipse supports M2M as well as M2T transformations. In order to support M2M transformations, Eclipse offers the possibility, entirely built into the workbench, to work with the ATL language (Atlas Transformation Language). ATL is an M2M transformation language supported in Eclipse with an editor, a debugger, and other features to make it easier to use. As for M2T transformations, the Eclipse Modeling Project offers two possibilities: JET and Xpand. JET (*Java Emitter Pages*) was the first M2T transformation language to be included in the project, and was the language used to implement early EMF M2T transformations. However, the Xpand language is the one now used for this purpose. Xpand is also an M2T transformation language and specializes in model-based code generation. Initially, Xpand was included in the *openArchitectureWare* project[3] but now it has been migrated, along with the rest of the project, to the Eclipse Modeling Project. The *openArchitectureWare* project was an attempt to implement MDE using Java, and was made up of several components, including Xpand.

Another important plug-in integrated into the Eclipse Modeling Project is the Graphical Modeling Framework (GMF). Using this plug-in, developers can automate the development of graphical editors based on EMF. Thus, it is possible to implement a graphical editor to create and manipulate models of a certain domain specified by means of a meta-model. In order to create these graphical editors, GMF requires four models to be created according to a series of meta-models defined by the plug-in. These models are the domain, the graphical aspects, the modeling tools and the mapping. The plug-in automates the generation of the modeling tool code by means of M2M and M2T transformations upon these models. GMF can be seen as the way of providing a DSL, of which an abstract syntax has already been defined using EMF, with a graphical concrete syntax.

[2] http://www.eclipse.org/modeling
[3] http://www.openarchitectureware.org/

Together with the Eclipse Modeling Project, the Eclipse approach for meta-modeling also includes the Generative Modeling Technologies (GMT) project. This project is an attempt to develop a series of tools for MDSD, and was initially centered on the transformation scope. The most important contributions in this field have been MOFScript, which is an M2T transformation language, and the Epsilon[4] set of languages, which includes languages for validating models, and much more. As most of these projects are still in the incubation phase, we have chosen not to make use of them at this time.

To conclude, in this project we have chosen a model-driven approach for the development of our collaborative tools, mainly because it provides an easy way to define new application domains and the structure of the tools, and also because of the possibilities of automation for the generation of the final tools. In this sense, some research works exist which automatize the collaborative workflow with a model-driven approach (Kim, 2012), so they could serve as a basis to do the same for our generated tools.

3. A model-driven method for developing collaborative modeling tools

In accordance with the ideas discussed above, we have chosen the Eclipse meta-modeling approach to support our development method, mainly to take advantage of the possibilities that the GMF plug-in provides when automating the development of the final tool and particularly in those aspects related to the GUI. Another reason for this choice is due to the possibilities that Eclipse offers for future extension as a container platform for the generated tools. It is worth noting that Eclipse can support the complete software development cycle and, thanks to its widespread use in the software industry, it can be considered a mature, tried and tested platform for supporting software development processes.

Nevertheless, the modeling tools that the plug-in generates are still mono-user tools and defining the domain is not a trivial task for a less than advanced user, since the corresponding models also need to be defined. Our contribution therefore is, on the one hand, to integrate collaborative functionality into the modeling tools and, on the other hand, to facilitate the definition of the application domains, working with just one model for the domain and another for the workspaces. These two issues are, thus, the two technological goals we aim to achieve in the scope of this project.

Users taking part in our development method may play any of four different roles (although in a real application of the method, some roles are often played by the same user):

- The *end user* of the method is the user who needs to build a collaborative modeling tool.
- The *domain expert* has lots of experience in the domain in which groupware tools are required.
- The *software engineer* may participate in those phases in which software development tools are manipulated.
- The *designer* will use the generated collaborative tool to build models in the chosen application domain. Designers are usually organized into groups and work collaboratively in design sessions.

Next, the three frameworks on which our proposal is based are explained in detail: firstly, the methodological framework, with the phases to be followed; secondly, the conceptual framework which includes the different meta-models and the transformations that are established between them; and finally, the technological framework which details how Eclipse plug-ins have been used to implement the method.

3.1. Methodological framework

The methodological process of our proposal is made up of a number of phases that must be followed when developing a collaborative modeling tool for a specific domain (Fig. 1). These phases are explained in detail below, and are related to the concepts in model-driven software development, as presented in Section 3.

Phase 1: **Domain identification**. The first thing to do when applying our development method to a given domain is to identify and isolate the application domain. This phase is carried out by the end user of the method in collaboration with the domain expert.

Phase 2: **Domain and workspace modeling**. The end user and the software engineer are responsible for developing the meta-models that are needed for the subsequent generation of the tool. They must model, on the one hand, the application domain, defining the elements and relationships it is made up of and, on the other hand, the structure of the workspaces, indicating which awareness and collaboration support elements should be included in the final tool.

Phase 3: **Production of the collaborative modeling tool**. Once the application domain and the workspaces definition have been formalized, some automatic steps take place resulting in the generation of the collaborative modeling tool. This phase is automatic, and is initiated by the end user, possibly with the help of the software engineer. This phase is divided into the following two steps (see Fig. 1):

Phase 3.1: **Model transformations**. Using M2M transformations, four GMF-compatible models are generated from the domain model. They are, specifically, the Ecore domain meta-model, the tool model, the graphics model and the mapping model.

Phase 3.2: **Generation of the tool**. Using the M2T transformations that GMF and our plug-in implement, the code for the final modeling tool is generated.

Phase 4: **Use of the collaborative modeling tool**. Once the modeling tool has been generated, designers can build models belonging to the application domain, working collaboratively in organized groups. The models to be created by the designers will be in accordance with the Ecore domain meta-model, which is one of the models generated in phase 3, and the tool to be used will match the workspace definition carried out in phase 1.

In order to provide a clearer comprehension of the phases of the method, a SPEM[5] specification of the methodological framework of the method has been created, as shown in Fig. 2. In the figure, which actors take part in each phase of the method can be seen, as well as what the input and output artifacts of the phases are.

Fig. 3 shows the meta-modeling processes that take place during the steps mentioned above. Each pyramid shows the different models and meta-models that take place in each meta-modeling process that occurs during the application of the method. The relationship between a model and the one above it is a "conform to" relationship. That is to say, a model is made up following the notation given by the meta-model above it (Bézivin, 2005). Thus, the first pyramid (left) corresponds to the domain modeling by means of EMF, which takes place in phase 2. In this phase, a model is built in accordance with the domain meta-model described in Section 4.2.1. Starting from the constructed domain model, the ATL transformations in step 3.1 generate the models that are recognized by GMF; that is, those that appear in the central pyramid. This pyramid shows the use of GMF to generate

[4] http://www.eclipse.org/gmt/epsilon/

[5] http://www.omg.org/technology/documents/formal/spem.htm

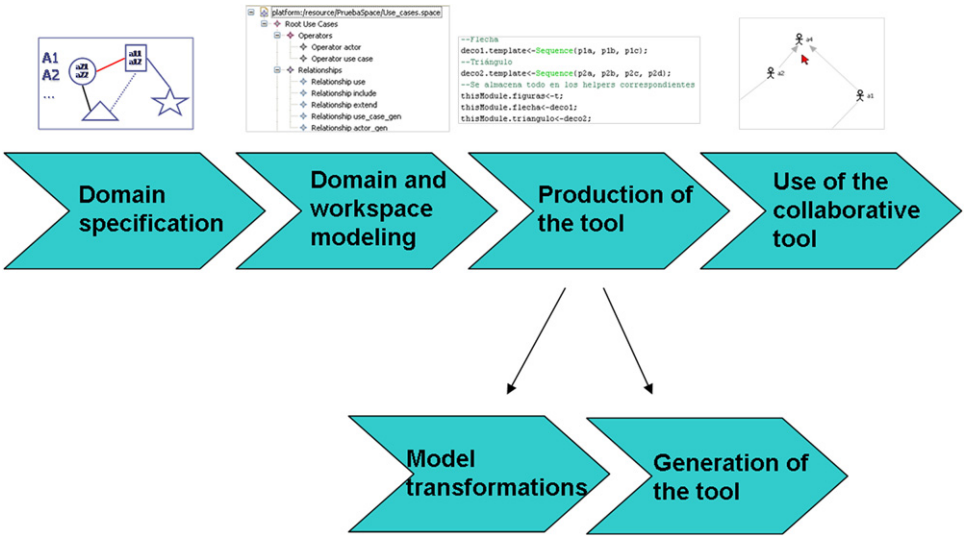

Fig. 1. Methodological framework.

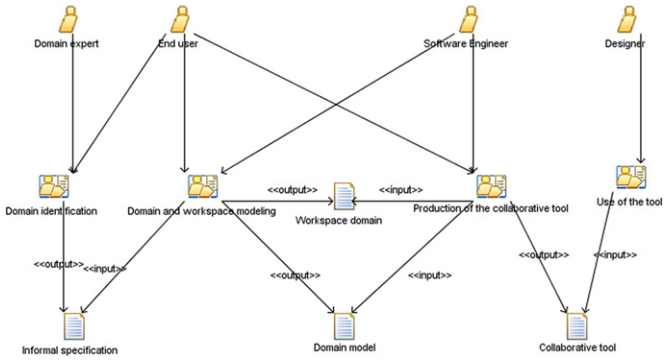

Fig. 2. SPEM specification of the methodological framework.

the modeling tool, as described in step 3. M2T transformations in GMF allow the tool code to be obtained automatically. Finally, the pyramid on the right corresponds to the work carried out by designers in step 4, in which concrete diagrams are built according to the domain meta-model. In the three pyramids, we can see that the meta–meta-model (level M3) is always Ecore, since we are working with the Eclipse meta-modeling approach.

3.2. Conceptual framework

The conceptual framework of our method includes two DSLs defined by means of the same number of meta-models. The first one is the *domain meta-model*, which defines the application domain in which the tools of the method work. It includes the concepts of the application domain, such as entities and relationships. The second is the *workspace meta-model*, which allows the user to define which workspaces make up the final system and which collaboration support elements (chat tool, radar view, telepointers, etc.) are going to be present in them.

The concepts that appear in the meta-models are taken from our research experiments in designing collaborative systems, such as DomoSim-TPC (Bravo et al., 2006), PlanEdit (Redondo et al., 2007) and COLLECE (Duque and Bravo, 2008). They are also taken from the analysis of existing groupware tools that are documented in the literature, such as Cool Modes (Pinkwart et al., 2002), Synergo (Avouris et al., 2004) and Co-Lab (van Joolingen et al., 2005), among others. And finally, we have taken concepts from groupware specification approaches, such as AMENITIES (Garrido et al., 2007) and CIAM (Molina et al., 2009). More specifically, concepts in the domain meta-model are the result of direct observation of the aforementioned groupware tools, and concepts in the workspace meta-model have been inspired by both the tools and the approaches for groupware modeling. For example, tools such as DomoSim-TPC, Co-Lab, CoolModes and Synergo deal with objects and the relationships between them, so these are the main concepts in the domain meta-model. With regard to the workspace meta-model, we have included some of the most popular widgets that appear in the aforementioned tools and approaches, such as chat features, session panels, coordination tools, tele-pointers, etc. In this meta-model, we have also considered the possibility of having several workspaces; not only a modeling workspace as is the case with the DomoSim-TPC system. Overall, we have tried to cover a wide range of groupware tools but always within the scope of tools for supporting modeling activities. Both meta-models include concepts from one of our previous projects in which we defined an ontology for conceptualizing awareness support elements in collaborative systems (Gallardo et al., 2011a).

Below, both meta-models are explained in detail showing their elements, attributes and relationships. As mentioned above, the meta-models are expressed in the Ecore notation, which is the meta-model implementation used in the Eclipse Modeling Project.

3.2.1. Domain meta-model

As previously mentioned, the domain meta-model (Fig. 4) contains the elements relating to the application domain. The *domain*

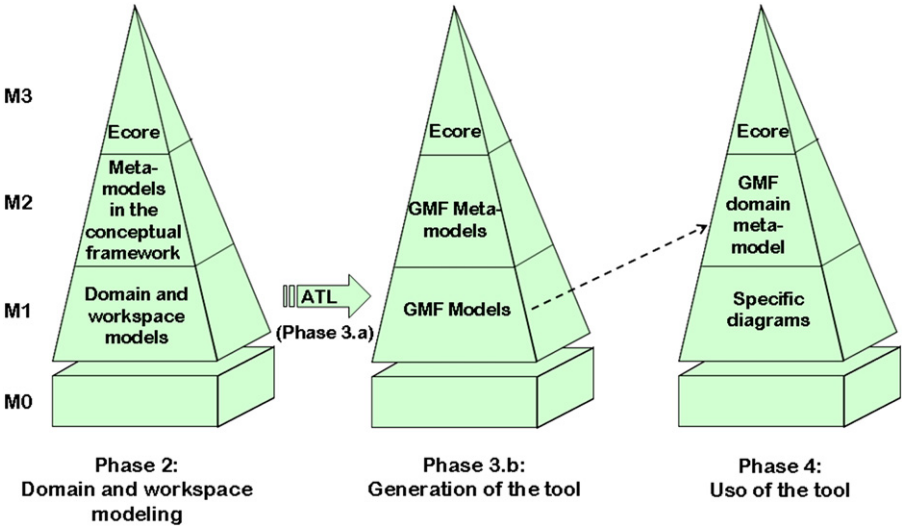

Fig. 3. Our meta-model approach.

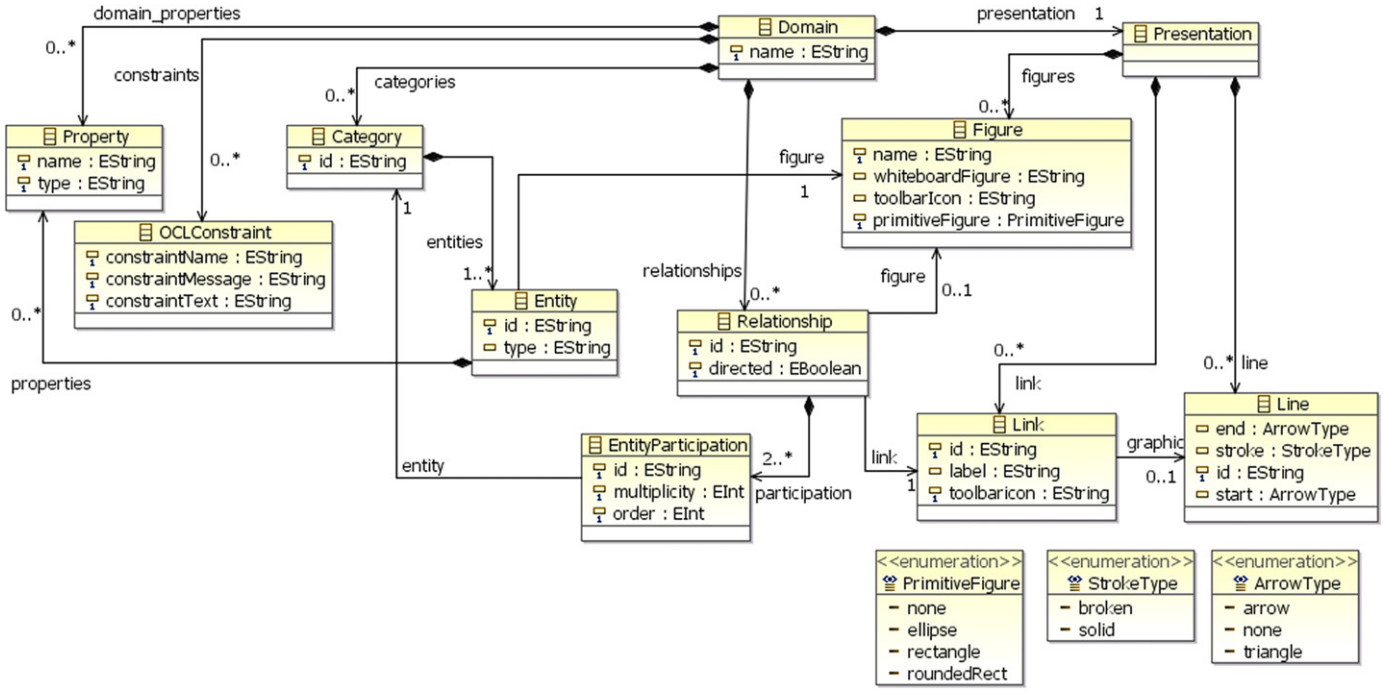

Fig. 4. Domain meta-model.

is the main concept of this meta-model, with the *entity* and *relationship* concepts also being of particular importance. This meta-model is a specific one for defining application domains for modeling tools, which makes it a more interesting approach than using more general meta-models, such as Ecore itself. The meta-models allow for defining both the concepts that are present in the domain and the graphical aspects, but in a way in which the two parts of the definition are not dependent on each other.

The domain meta-model is made up of the following concepts (see Fig. 4):

- Domain. This is the main concept. An application domain is made up of an aggregation of a series of entities, relationships, properties, categories, constraints and presentation elements. Conceptually, the domain is the field of knowledge in which work is being carried out at any given moment using the groupware modeling tool.
- Entity. An entity is a relevant concept of the domain, and can be instantiated in the model. For example, in a Use Cases Diagram an entity can be an actor or a use case, whereas in Digital Circuits, an entity is a logical gate or an input/output. Each entity is characterized by an identifier and a type, and belongs to a category. This is needed when defining the possible relationships in which the entity can take part. Furthermore, entities can have properties, and each entity is associated with a specific graphical representation which is called a figure.
- Category. This concept is a grouping of entities which share some common characteristics. As entities, they have also a type property. Entities can be classified in two different ways. This is due to the fact that some domains require the definition of different logical groupings. For example, in the Domotics domain you can find entity types (receivers, activators and control systems) as well as management areas (comfort,

energy control and security control). The grouping of entities using categories is also used to define which entities can be linked using the different kind of relationships.

- Property. Each entity is characterized by a set of properties, which have a certain name and are of a specific type. The domain itself can also have a set of properties.
- Figure. A figure is the graphical representation of an entity in a diagram. Each figure has a name, an image to be displayed in the diagram and an icon that represents the figure in the toolbar. Images can be extracted from a file as well as selected from a set of primitive figures, such as circles or rectangles.
- Line. Using this concept, it is possible to define different types of lines that will later be used to define the graphical representation of the relationships. Each line has a stroke (broken/ pointed) and can have either open or closed arrows at the start and at the end.
- Link. A link defines the graphical representation of one or more relationships. Each link is associated with a given kind of line, and can also define a fixed label and an image to be displayed in the toolbar.
- Presentation. This concept gathers together all the graphical elements: the figures and the kinds of line as well as the already defined links, so that they can be associated with the relationships in the domain.
- Relationship. This is a conceptual association between two or more entities. The graphical aspects of a relationship are given by its association with a specific EntityParticipation. This concept is used to link the relationships to the categories of entities that can participate in them. Relationships can be *n*-ary, meaning that for each relationship there would be n participation elements, linking a relationship to n categories. Thus, a category can be present in n relationships, or even several times in the same relationship in the case of one which links entities of the same category. In the case of n-ary relationships, the relationship is associated with a figure which is the connector between the elements appearing in the relationship.
- OCLConstraint. These are constraints that a model belonging to the application domain must fulfill in order to be in a valid state. For example, in a model belonging to the Digital Circuits domain, there could be a maximum number of logical gates. The language we have chosen to describe these constraints is OCL (Object Constraint Language), since it is the most widely used in this field.

By using this meta-model, a wide variety of domains can be represented. Of course, there are domains in existence which have some characteristics that have not been considered in the model as we have yet to implement the required functionality in our technological framework. Some characteristics that will be considered in future versions are the possibility of entity composition, i.e. entities made up of the aggregation of other entities, or the addition of geometrical constraints so that entities must be placed in certain places on the shared whiteboard. A number of examples are shown in Section 5. Models following this meta-model, as well as the ones following the workspace meta-model, will be developed using the EMF tree editor generated from the meta-models themselves. By carrying out preliminary studies about the use of these editors, we have checked that users are comfortable with them and are able to develop suitable models. Thus, the development of more complex editors is not one of our priorities, although we consider it to be one of our objectives for the improvement of the method.

3.2.2. *Workspace meta-model*

The second meta-model of the conceptual framework is the workspace meta-model (Fig. 5), which describes the tools with which the tasks that the collaborative system supports are carried out. This system can be made up of different workspaces, each one including some tools for awareness and collaboration support in order to make the specific task that is to be carried out easier. There is a mechanism for navigating through the workspaces (see Section 4.3). There are different workspace configurations but we have identified the most common tasks related to modeling. Some

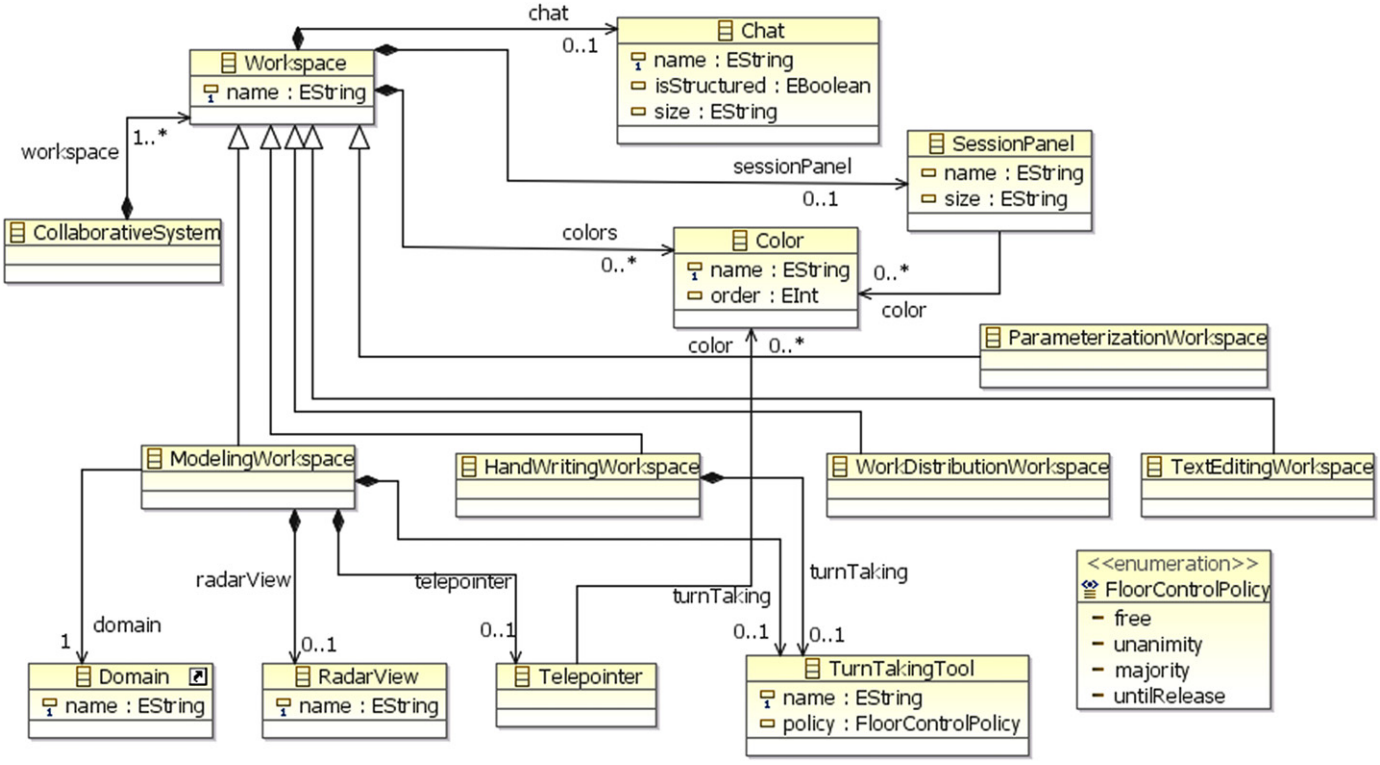

Fig. 5. Workspace meta-model.

of the support tools can appear in every workspace (i.e. the chat room and the session panel), so these are linked to the global *Workspace* concept. There is also a *Collaborative System* concept, which groups the different workspaces and represents the system as a whole.

These are the concepts that make up the workspace meta-model:

- Collaborative system. This is the system as a whole, and is made up of an aggregation of workspaces.
- Workspace. A workspace is made up of an aggregation of the tools that allow the designer to carry out his/her work collaboratively.
- Modeling workspace. This is a type of workspace that allows modeling tasks to be carried out, i.e. the creation and editing of diagrams. It is associated with a specific domain, with the domain concept being the same as that of the domain meta-model.
- Hand-writing workspace. This workspace is used to draw sketches and notes. A possible application could be the first steps of a software development project or any other kind of engineering project.
- Work distribution workspace. When working collaboratively, it is often necessary for the users to do some kind of task allocation or work distribution before starting work. This process is carried out in this workspace, in which, according to different work organization criteria, the users specify tasks or entity categories for working with via proposal-based tools, in a similar way to DomoSim-TPC (Bravo et al., 2006).
- Parameterization workspace. In modeling systems which have parameters that must take a value before modeling starts, there is usually a prior step for defining parameters, known here as parameterization. For this purpose, proposal-based and democratic coordination protocols are used; the specific protocol that is to be used can be chosen.
- Chat room. All collaborative workspaces should have a chat tool to provide support for communication between the participating designers. The chat feature has some properties, such as its size or a structured chat option. A structured chat room offers a pre-established set of communication acts, providing a structure for the conversation (Bravo et al., 2004). These acts, called structured messages, have a semantic meaning which can be understood by the participants.
- Session panel. The session panel is the widget that shows who is working on the current session. For each participant, several pieces of information are shown, including his/her avatar, the workspace he/she is working in (in the case of people working in several workspaces simultaneously) or the specific action or task he/she is carrying out within the workspace. This widget is a typical element for providing awareness, along with the radar view and the tele-pointers.
- Color. It is possible to define the colors used to identify the participants in the work session. Colors are assigned to the participants when they join the session. The session panel and tele-pointers will then use that same color to identify each participant.
- Radar view. This widget is useful when big models are being handled as it identifies the part of the model that is being watched at any time.
- Tele-pointer. Tele-pointers allow people working in a modeling workspace to know where the rest of the participants are pointing. To make this possible, a representation of each of the other participant's pointers is shown. Each one of these pointers is highlighted with the color assigned to that particular participant. Of course, tele-pointers move around the workspace at the same time as the participant's pointer.
- Turn taking tool. This tool makes it possible to manage the floor control during the session. Different floor control policies can be established, for example free editing, turn taking by unanimity, turn taking by majority or turn taking until explicitly released. As such, an attribute is included for the specific policy whose values are taken from a given enumeration.

Regarding generality issues, we believe that this workspace meta-model is a generic meta-model that can be instantiated to configure a wide range of workspaces and collaborative system configurations. However, some extensions to the meta-model would increase its applicability, such as including more widgets in the workspaces or new workspaces in the collaborative system.

3.2.3. Model transformations

During the application of our method, two transformation processes take place. The first one is made up of the M2M transformations that we have developed in order to generate the four models required by GMF. The second one is code generation carried out by GMF, which generates the modeling tools with the collaborative functionality due to the extensions we have implemented. Next, we are going to explain both processes briefly and show how they contribute to our development method.

Four M2M transformations have been defined and implemented using the ATL transformation language. Each one of them generates one of the models required by GMF to generate the modeling tool, as shown in Fig. 6. Thus, GMF can later process the generated models in order to build a graphical editor. In the first process, the Ecore model is generated from the application domain model. In Ecore models, each element must be modeled as an Eclass. Therefore, an EClass element is created for each category, entity or relationship found in the domain model. Entities belonging to categories are implemented as EClasses, which are the subclasses of the EClasses that represent the categories. This way, all subclasses that model the entities included in the category will be taken into account when creating relationships between the EClasses which model the categories. Also, EAttributes for the labels of the entities and relationships are created. And finally, each EClass has two EReferences which allow it to link with the source and target entities of the relationship.

The second M2M transformation generates the GmfGraph model, which defines the graphical aspects to be considered when producing the tool. A Canvas is created to later include the rest of the elements that have been added to the application domain model. The elements that are included in this Canvas are instances of the Node class which model the entities and instances of the Polyline Connection class which model the relationships. Both Nodes and

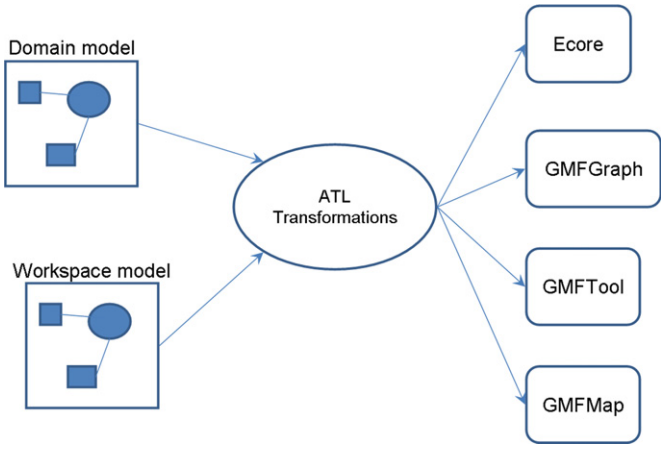

Fig. 6. ATL Transformations.

Connections own a Figure Descriptor attribute and a Diagram Label. This way, they can be related to labels that may have been defined in the Ecore model.

Another M2M transformation process is the one that gives rise to the GmfTool model. This model defines which creation tools will appear in the drawing palette of the tool. A Tool Registry element will be included in this model, and a Creation Tool will be added for each entity or relationship that is present in the original application domain model.

Finally, the last model generated by means of the ATL transformations is the GmfMap model, which is the model that maps the concepts present in the remaining models. Some Mapping and Canvas Mapping elements will be the root concepts in this model, while a series of elements will be created for each Node and Connection in the GmfGraph model. These will be a Top Node Reference, a Node Mapping and a Feature Label Mapping in the case of Nodes, and a Link Mapping and a Feature Label Mapping for Connections. Also, links will be established between these new concepts and the ones in the Ecore, GmfGraph and GmfTool models. It is also worth noting that this is the model in which OCL constraints defined in the domain model are stored.

In addition to the M2M transformations, we have developed some extensions of the GMF code generation templates in order to get the M2T transformations to generate the collaborative tool. These transformations basically consist of invoking the methods of our new CGMF package so that its functionality is included in the generated tools. Fig. 7 shows the main changes made to these templates.

3.3. Technological framework

Our contribution to the technological framework consists mainly of integrating different technologies from the Eclipse Modeling Project and extending them to obtain collaborative functionality. Therefore, Eclipse is used as a container platform for the developed modeling tools. The following plug-ins and tools from the Eclipse Modeling Project have been used:

- EMF. This allows models to be created in a tree view, according to a pre-defined Ecore meta-model. In our case, we use EMF plug-ins so that the end user, perhaps in collaboration with the software engineer, defines the domain and workspace models according to the Ecore meta-models that have been described within the conceptual framework.
- ATL. This has been used to transform the domain and workspace models into the models that the GMF plug-in needs to generate the modeling tool, as shown in Fig. 6. One of those models is the Ecore meta-model itself, which defines the structure of the models to be created by the designers. For example, when generating the GMF tool definition model, a corresponding entry in the toolbar is created from each object or relationship in our domain model.
- GMF. This plug-in implements the generation of graphical modeling tools based on Ecore meta-models. The plug-in has been extended so that it generates collaborative tools, with which several designers can work on the same diagram in a distributed, synchronous way. This is achieved by adapting the templates handled by GMF for M2T transformations and by coding the classes needed to use the functionality provided by the ECF (Eclipse Collaboration Framework) plug-in. Thus, the new plug-in receives the respective models and generates the collaborative tool without any further action from the user being required.

Accordingly, the software architecture of the generated collaborative modeling tool is provided by the architecture that GMF defines for all the tools generated using it. To be specific, the source code of the modeling tool is separated into three plug-ins generated by GMF, which makes use of the rest of the plug-ins taking part in the process.

As shown in Fig. 8, and considering that the main modeling project is named *DomainModeling*, the three plug-ins generated by GMF would be the *DomainModeling, DomainModeling.edit* and *DomainModeling.diagram* plug-ins. *DomainModeling* contains the domain classes that represent the EClasses and ERelationships defined in the Ecore file. An interface and a class to implement it are generated for each EClass or ERelationship. This plug-in therefore makes use of the functionality of EMF. *DomainModeling.edit* only contains the *provider* classes, which are used in the creation of objects corresponding to the classes of the model. As such, this plug-in depends on EMF and *DomainModeling*. The tool's main plug-in is *DomainModeling.diagram*, which implements the proper modeling tool. This plug-in implements the graphical part using GMF, Draw2D and CGMF (Collaborative GMF), which also uses ECF. CGMF is our extension of the GMF plug-in for implementing collaborative functionality, as will be explained later. As far as model creation and modifying are concerned, our plug-in makes use of the *diagram* plug-in and EMF.

Regarding the architecture of the specific classes that implement the modeling tool, GMF uses the Model-View-Controller pattern. Therefore, the model is the set of classes that implement the business model, the view is the graphical representation that uses the Draw2D package, and the controller is made up of a set of classes, denominated *EditParts*, which coordinate both model

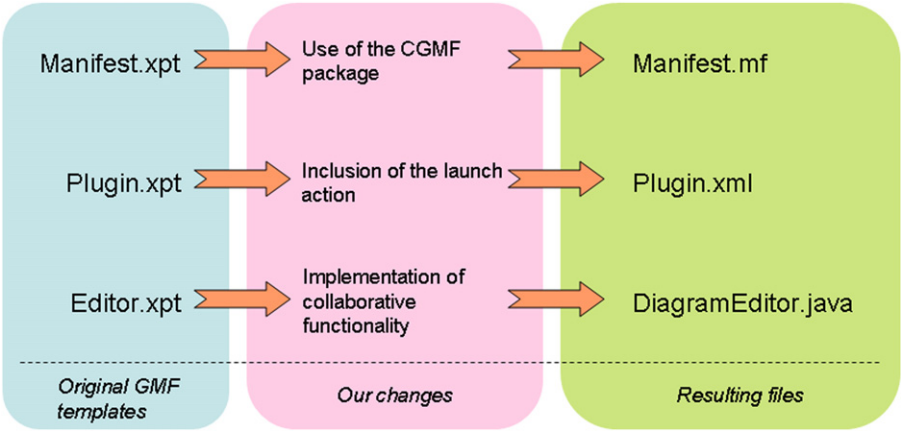

Fig. 7. Main changes made over the GMF code generation templates.

and view. Of course, all of these classes are generated automatically by GMF using M2T transformations. The usual process for the creation of the diagrams begins with the controller reading the model and then creating the view that matches that model. Once the view exists, the controller receives update notifications from the model so that it can call the view to be updated.

Since it is necessary to implement distributed synchronous collaboration between the different participants, the use of some type of network technology to support this is required. For this purpose, the ECF plug-in has been chosen. ECF is a framework which, among other services, includes some APIs that provide the necessary support for synchronous collaboration. By including calls to some of those APIs within the GMF-generated classes, we have managed to make the modeling tool generated by GMF a synchronous collaborative tool. Notably, the services provided by ECF used in our approach are the ones offered by the Datashare API, which provides services for the creation of channels, and by the Shared Object API, which is used to create and replicate objects within an ECF Container. This way, a new plug-in is obtained, which is the collaborative extension of GMF and which we have named CGMF. In particular, the code of GMF as well as the Xpand templates used by GMF, have been extended so that when a diagram is modified by a designer, the rest of the instances of the tool that are working on the same diagram all receive the latest modifications. The messages that inform the different instances of the tool about the changes made are implemented as ECF shared objects.

The network architecture of the modeling tool is therefore supported by the architecture of ECF. When using the tool, a generic ECF server is running on a well-known computer to which the participants connect. To do so, they need to know the URL corresponding to the machine on which the server is running and the pre-determined name of the work session's server.

ECF handles work sessions using the most generic abstraction, known as a *container*. In this case, we have a container for each shared diagram on which someone is working. Within that container, and using the *IChannelContainerAdapter* adapter, a main channel is created for the exchange of messages related to work with the diagram itself. In addition, for the use of those other components of the application which require network communication (i.e. collaboration), such as the chat feature or the session panel, different channels are created within the same main container.

With regard to the messages that the different clients in the application exchange, they encapsulate the changes made in the model so that the rest of the instances of the tool being executed can replicate them. These messages are serialized and sent in the form of byte arrays to the corresponding channel. The rest of that channel's clients, through the respective listener process, receive this message and replicate the changes to their application instance. The diagrams of the different designers are thereby constantly updated with all the changes that have taken place. This means that a sort of hybrid architecture is used (in the sense in which Roseman and Greenberg (1996) define it), with replicated instances of the main process and the diagrams in each client, but also with a centralized component which is the ECF register. When the floor control policy allows many users to edit the model at the same time, concurrency control mechanisms are used at entity and relationship levels. That is to say, the system prevents a user from manipulating an element when it has been selected (or is in use) by a different user. This allows parallel working to take place. For example, in a UML Activity Diagram several users could be working at the same time, each one modeling in a different swim lane. Another example could be a UML Use Case Diagram, in which the different designers could deal with different user requirements by each working on different actors and use cases in different areas of the diagram. Since the model is always updated in real time, version control is not needed.

The tools that are generated when the development method is used are made up of the widgets that are selected when instantiating the workspace meta-model. As previously stated, the tools can have several workspaces, implemented as Eclipse perspectives. Each perspective is made up of a set of views, which correspond to the aforementioned widgets (e.g. chat room, session panel, etc.). When a modeling task in a specific workspace has finished and the designers want to move to a new workspace, a new perspective is loaded so that the same system can be used

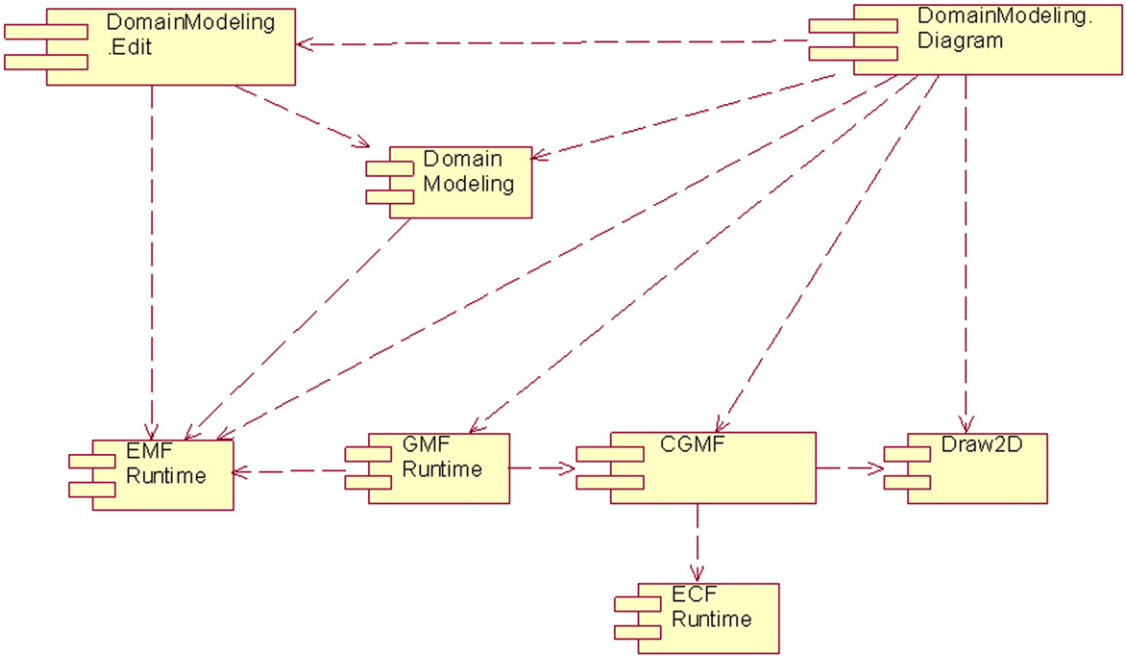

Fig. 8. Component diagram.

throughout the entire general modeling work. To achieve this, a coordination panel is used. This way, a user can suggest that the group moves to another workspace, and the rest of the users can choose whether or not to accept this proposal.

Fig. 9 shows how a modeling workspace is generated from the workspace definition. This workspace includes some of the tools for collaboration support that are supported by the method, such as a chat feature, a session panel and a tool for taking turns.

3.4. Comparison to similar approaches

Our proposal for the model-driven development of collaborative modeling tools, as presented above, has a number of differences when compared to other similar approaches. With regard to other approaches for the modeling of collaborative systems, Table 1 presents a comparison with two other methodologies, CIAM (Molina et al., 2009) and AMENITIES (Garrido et al., 2007).

By analyzing the above comparison, some interesting conclusions can be reached. First of all, our method is the only one of the three which has been conceived for a specific kind of collaborative system, just as modeling systems are. This makes our method less generic, but it provides an answer to specific problems with these systems such as the difficulties in developing systems from scratch or the way in which domains are defined. Regarding the notations and tools that are used in each approach, our method has the advantage of dealing with notations and tools that are widely used by both domestic and professional users (Eclipse technology), whilst the other approaches either do not have a specific associated notation or tool or have a new tool which is unfamiliar to most users.

As for the elements that are supported, our method lacks support for the definition of requirements, users and sessions, and tasks. We consider that the ability to define the application domain, something which is not supported by other approaches, makes defining the requirements unnecessary as the functionality of the tool is quite clear. On the other hand, support in our method for defining tasks and also for defining users and sessions is being developed as we speak and will mark a definite improvement in the conceptual framework.

Lastly, our method is the only one that provides full support for the implementation of the final tool. This is one of the most important contributions of our method, and it has been made possible thanks to the use of the plug-ins from the Eclipse Modeling Project. Other approaches either do not provide direct support for the implementation or do so indirectly through the use of other notations, as is the case of CIAM which uses the CTT (Paternò, 2004) notation for task modeling.

Table 2 below shows another comparison, this time between the tools generated by our method and some other tools mentioned in Section 2. This comparison shows how our tools are more complete than the other tools being analyzed in these features: first of all, the tools our approach produces can work with any application domain made up of entities and relationships, not just with a given set of fixed domains. Another important advantage of our tools is the existence of different workspaces that can be put together within the same system, with users being able to move between them. The last advantage is the number of elements for collaboration and awareness support.

4. Case studies

Next, in order to evaluate the whole method we have proposed, the entire process of creating a collaborative tool for the domain of Use Case Diagrams domain is explained in detail. Then, some more examples of application will be briefly described. With these case studies, we aim to show that our method is well-suited to the purpose for which it was conceived, which is to systematize the development of groupware systems that provide support for modeling activities.

4.1. The use case diagrams case study

This case study consists of the formalization of both the domain and workspace, following the methodological process, and the subsequent creation of a collaborative modeling tool to support the building of UML Use Case Diagrams. The users who took part in the process were two Software Engineering teachers who simultaneously took on the roles of end users and domain experts, one software engineer, and the potential designers who were Computer Science degree students. The teachers and the engineer were asked for their opinions about the method after completing the study, and they stated that the method was useful for achieving the goals for which it was initially conceived.

4.1.1. Domain identification

First of all, the domain description for the Use Case Diagrams was made by the domain experts. The result of this first phase was an informal specification of the concepts or entities, their attributes and the relationships between them. The experts made a paper sketch, drawing use cases as ovals and actors as figures, with the typical relationships between use cases and actors being represented by different types of arrows with labels.

4.1.2. Domain and workspace modeling

In the second phase of the method, the formal specification of the application domain was completed by working from the informal description. Two models were created, in accordance with the corresponding meta-models and using the model editor provided by EMF. These models were developed by the end users with the help of the software engineer, so some work sessions were arranged to allow them to work together. Fig. 10 shows the tree view for the domain meta-model. As the figure shows, the two identified concepts, *actors* and *use cases*, are now represented as domain entities (see *Entity* nodes in the figure). Each entity belongs to one of the two included categories, since in this domain there is no need to group more than one entity into the same category. On the other hand, five different types of relationships were included (see *Relationship* node in the figure): *use, include, extend, use case generalization* and *actor generalization*. Each relationship definition comes with two *Entity participation* elements in order to define the types of entities that can be interrelated. In the presentation section, three types of lines have been modeled, and they are later used by the different types of relationships that have been created. Also, two types of figures are included, each one associated with one of the two domain entities.

For the workspace model, which was also defined by the end user and the software engineer, it was decided to include only one workspace containing a modeling workspace. This workspace had a chat feature, a session panel and a tool for floor control. There was no parameterization workspace because it would make no sense to have one since the domain does not have any domain properties or, indeed, any other kind of workspace.

4.1.3. Production of the collaborative modeling tool

In this third phase, the automatic transformations of the method were executed, taking the specification obtained in the previous phases as input. As explained above, four archives to be used by CGMF were generated.

Fig. 11 shows, as an example, the file for the graphical elements definition. It includes a *Canvas* element which represents the

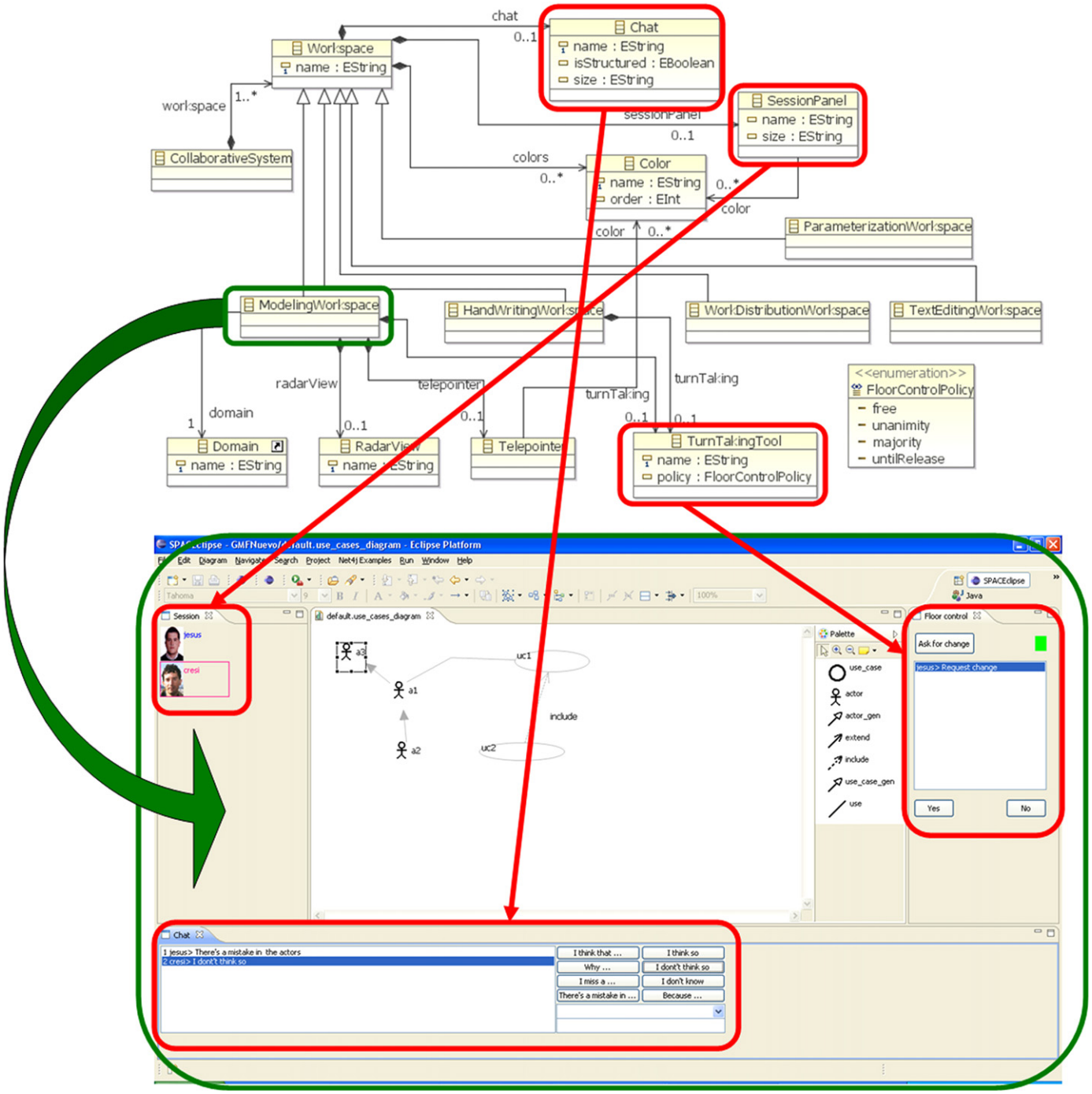

Fig. 9. Relationship between the meta-model and the workspace configuration.

Table 1
Comparison between our method and two different approaches for the modeling of collaborative systems.

	Our method	CIAM	AMENITIES
Kind of systems it supports	Collaborative modeling systems	Collaborative systems	Collaborative systems
Notations used	EMF, a DSL developed with GMF	CIAN notation, which integrates CTT (ConcurTaskTrees) and others	Ontologies, COMO-UML
Support tools used for modeling	Eclipse with certain plug-ins	Partial	COMO-TOOL
Support for the definition of requirements	No	No	Yes
Support for the definition of the application domain	Yes	No	Partial
Support for the definition of users, sessions, etc.	Work in progress	Yes	Yes
Support for the definition of tasks	Work in progress	Yes	Yes
Support for the configuration of workspaces	Yes	No	No
Support for the production of the groupware system	Yes	Partial	No

Table 2
Comparison between the tools generated by our method and two existing tools for collaborative modeling.

	Our tools	CoolModes	Synergo	Dawn
Domains supported	Any domain	A set of given domains (Petri Nets, Collaborative Calculator, etc.)	A set of given domains (concept maps, flowcharts, etc.)	Any domain
Workspaces included in the system	Modeling, Hand-writing, parameterization, work distribution	Modeling	Modeling, supervision	Modeling
Awareness and collaboration support elements	Tele-pointers, chat, Session panel, radar view, turn-taking tool	Radar view	Chat	None
Format in which models are stored	XMI	XML	Proprietary format	XMI

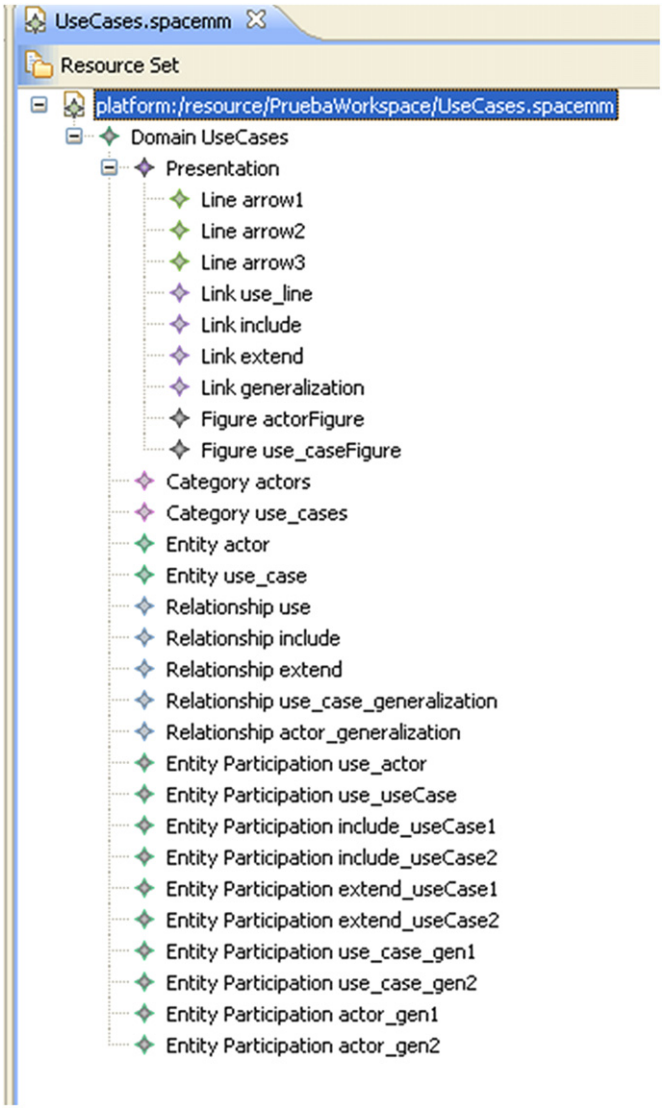

Fig. 10. Model for the Use Cases Diagram domain (developed by the user).

whiteboard where the diagram is created, and also a series of *Node*, *Connection* and *Label* elements which model the set of objects with which the work is done. These are, of course, the ones that were defined in the application domain model. *Nodes* refers to entities, *Connections* to relationships and *Labels* to labels in both types of elements. The model also has a *Figure Gallery* element which includes default graphical elements such as lines or arrow ends that are used to visually represent the different nodes or connections of the domain. The definitions of these elements are automatically created by the transformation process.

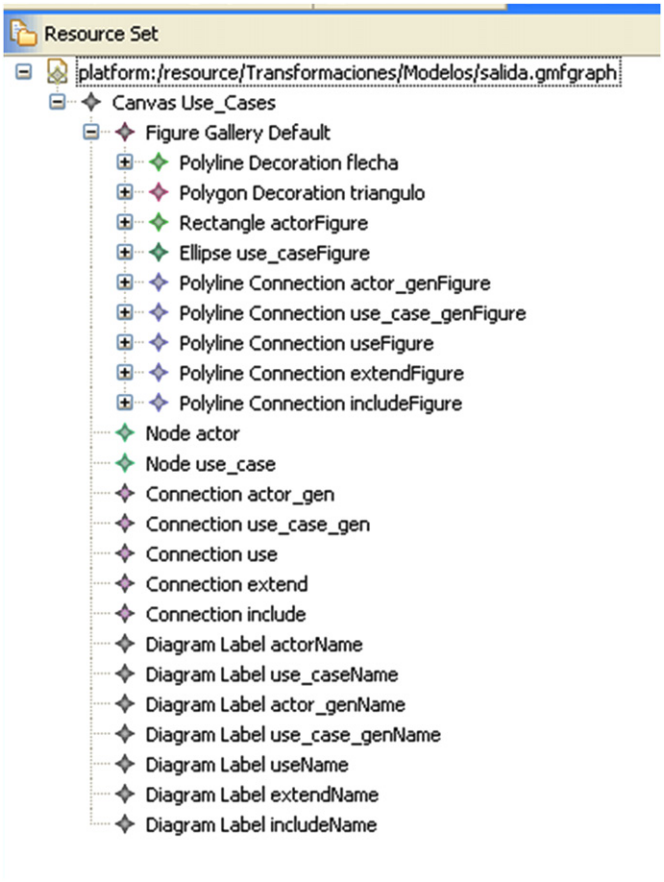

Fig. 11. Model for the definition of graphical elements (automatically generated).

4.1.4. Use of the tool

Finally, once the specifications have been obtained and the transformation process has been initiated by the end user, the collaborative modeling tool is obtained and the designers can begin to work collaboratively on Use Case Diagrams. Fig. 12 shows a screenshot of the collaborative modeling tool. Some instances of the *Use Case* entity are seen, as well as some instances of the *Actor* entity. We can also see how relationships between the entities are established. In addition, we can see how the graphical representation of each one of these elements matches what was initially specified in the domain definition. As such, *Use Cases* are represented as ovals and *Actors* as stick figures taken from an image file. Likewise, the relationships also have the line type and the label that was specified earlier: for example, a solid line without a label in the case of the *Use* relationship, and a discontinuous line with the label *include* in the case of the *Include* relationship.

Fig. 13 below clearly shows the correspondence between the domain specification, the toolbar of the generated tool and the elements on the shared whiteboard. From the specification of each

entity and each relationship (in the Figure, the *actor* entity and *actor generalization* relationship are highlighted), an entry is generated in the toolbar of the modeling tool. Afterwards, when using the tool, designers click on the toolbar and place the corresponding entities and relationships onto the shared whiteboard.

Finally, in the modeling tool, a tele-pointer can also be seen as an awareness support component. The tele-pointers allow each user to be aware of the movement of other designers' pointing devices. This means that the participants in the design session have information about what their partners are doing at all times. The other components for collaboration and awareness support that are present in this tool are the session panel, the chat feature and the turn-taking tool. In the session panel in the example (see Fig. 12), three users are working at the same time and each one has a specific associated color. In the chat feature, a conversation is underway between the users. And in the turn-taking tool, we can see how a user has asked the others for permission to edit and they have granted him that permission.

4.2. Other case studies

Another application domain to which this same development approach was applied is that of Digital Circuits. The steps of the methodological framework method were followed. The resulting domain model identified four entities and just one relationship. The entities were the input signal, the output signal and two logical gates chosen for the study. The workspace model developed for this application also had only one workspace (the modeling one). Afterwards, the automatic production took place and the tool for collaborative Digital Circuits modeling was produced. Fig. 14 shows a screenshot of the modeling tool after the method was applied. A circuit with two inputs, one output and three logical gates (two 'ands' and one 'or') was being developed when the screenshot was taken.

We have worked with some other application domains, developing tools made up of more than one workspace when it was necessary. For example, we have developed a system for designing task models using the CTT notation, which allows for the design of interactive application GUIs. This system begins with a hand-writing workspace so that the designers can do an initial sketch of the user interface. Next, the CTT model is produced by designers in a modeling workspace. Since there are two different workspaces, a coordination panel is included to allow users to decide when to move to another workspace. Fig. 15 shows the tool obtained for this domain. It is worth pointing out the different entities (e.g. abstract task, interactive task, etc.) and relationships (concurrency, choice, suspend/resume, etc.) that can be seen in the corresponding toolbar.

A different example of a domain that we have modeled is that of Concept Maps (Fig. 16). Here, the entities are the concepts and there is just one kind of relationship, although it can be labeled with the name of the relationship between the linked concepts.

5. Empirical study

In order to test the suitability of some features of the development method that we have proposed, an empirical study of use of the method has been carried out with students of Computer Science and

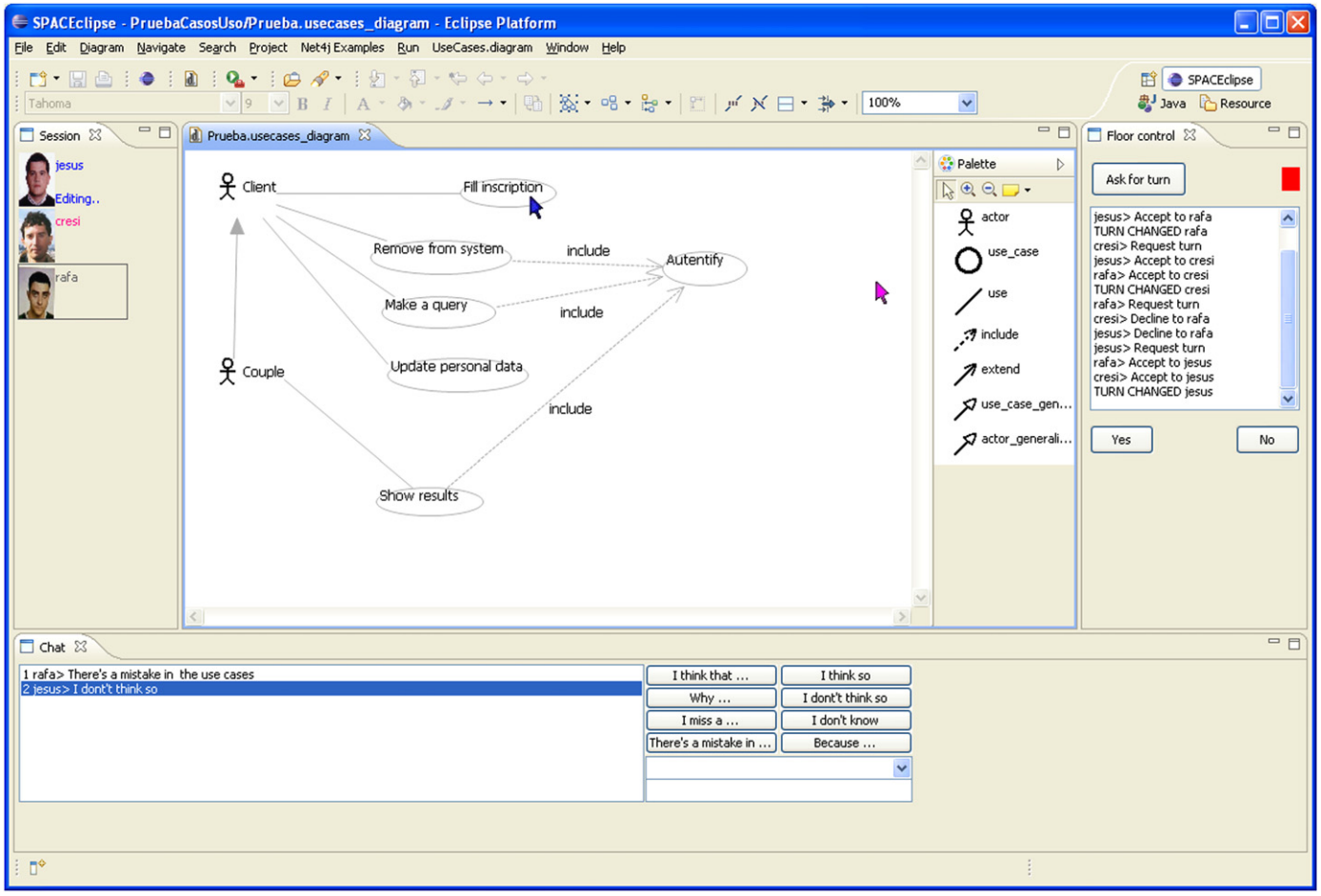

Fig. 12. Collaborative modeling tool working with the Use Case Diagram domain.

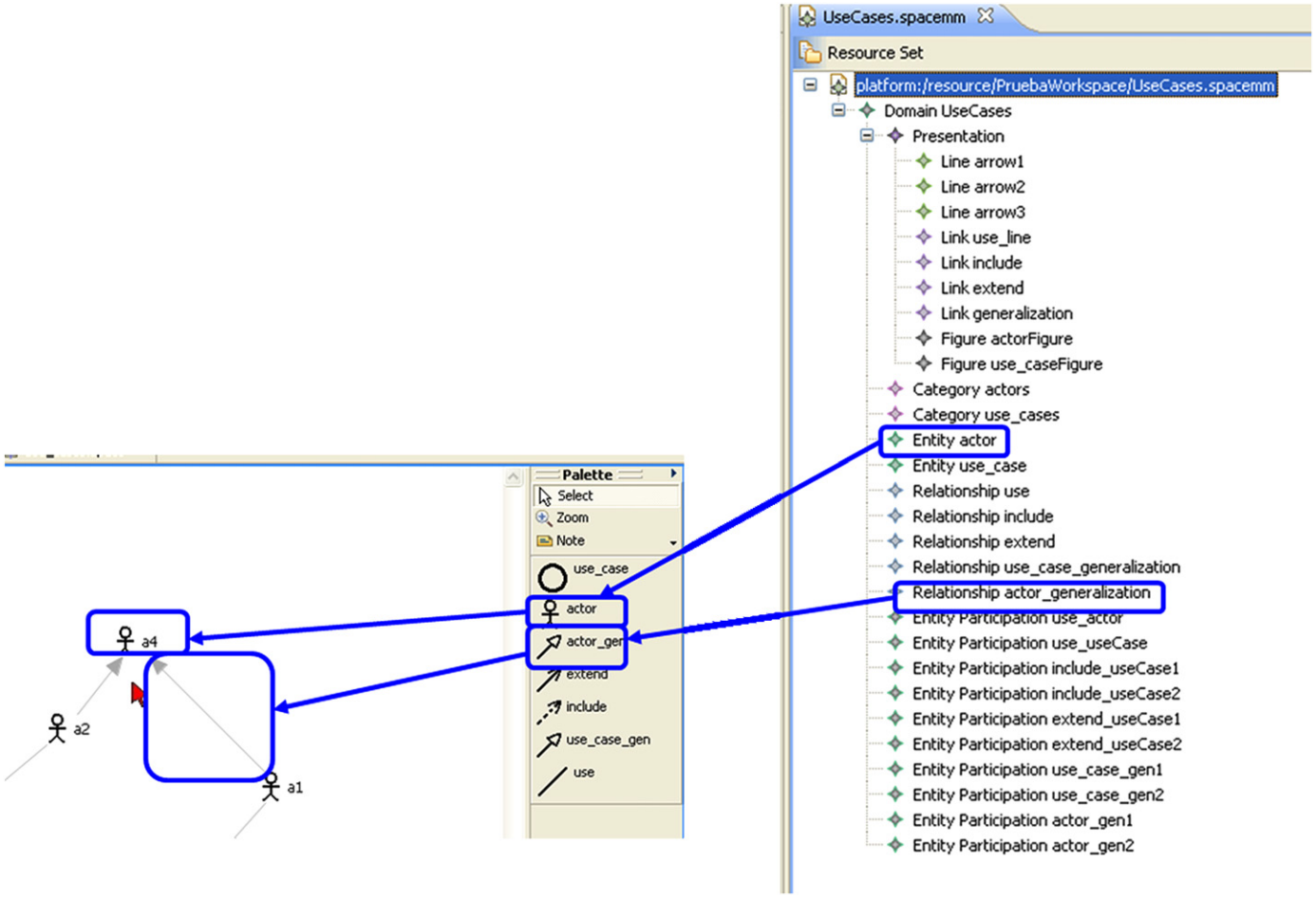

Fig. 13. Correspondence between the domain specification and the generated tool.

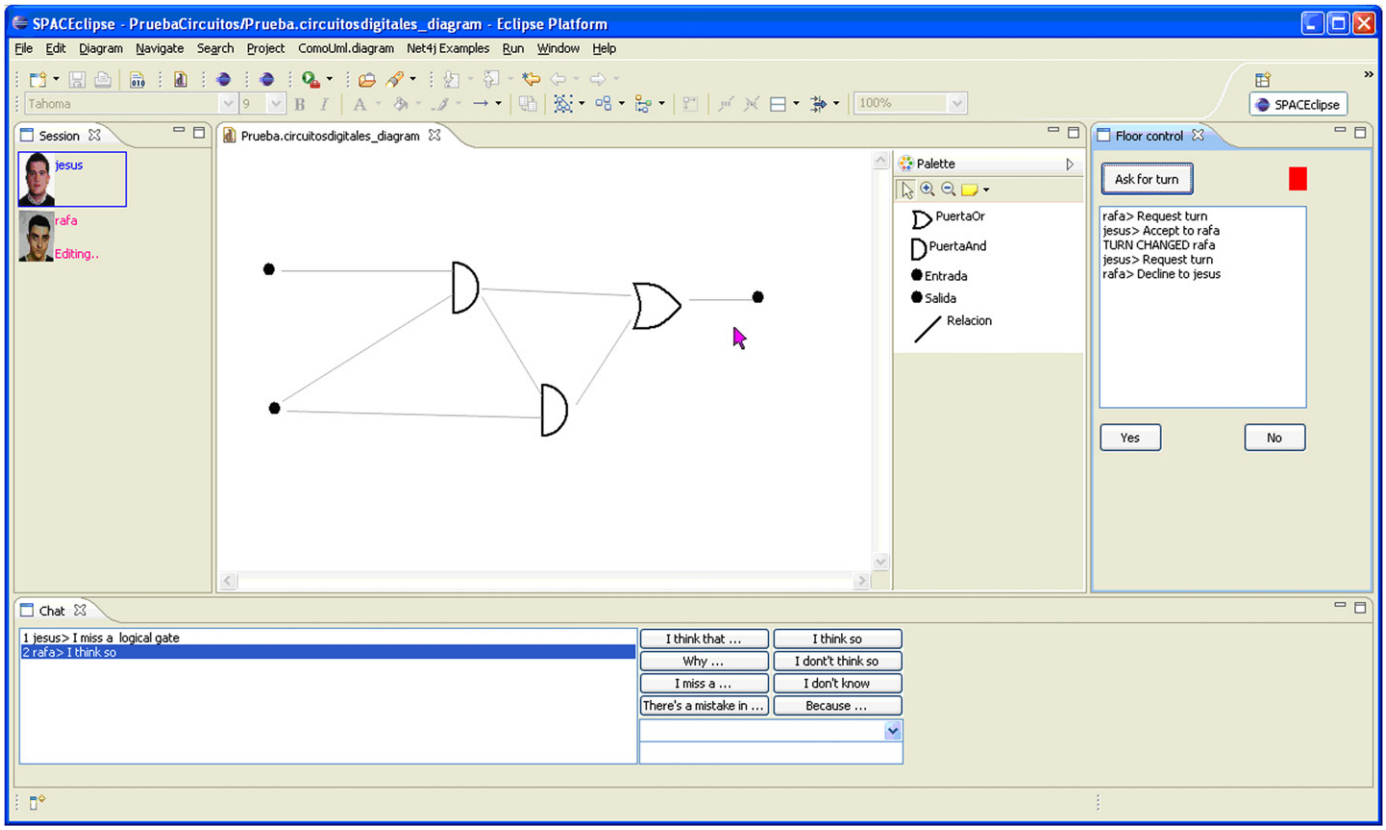

Fig. 14. Tool for the modeling of Digital Circuits generated using our development method (some elements of the user interface are in Spanish).

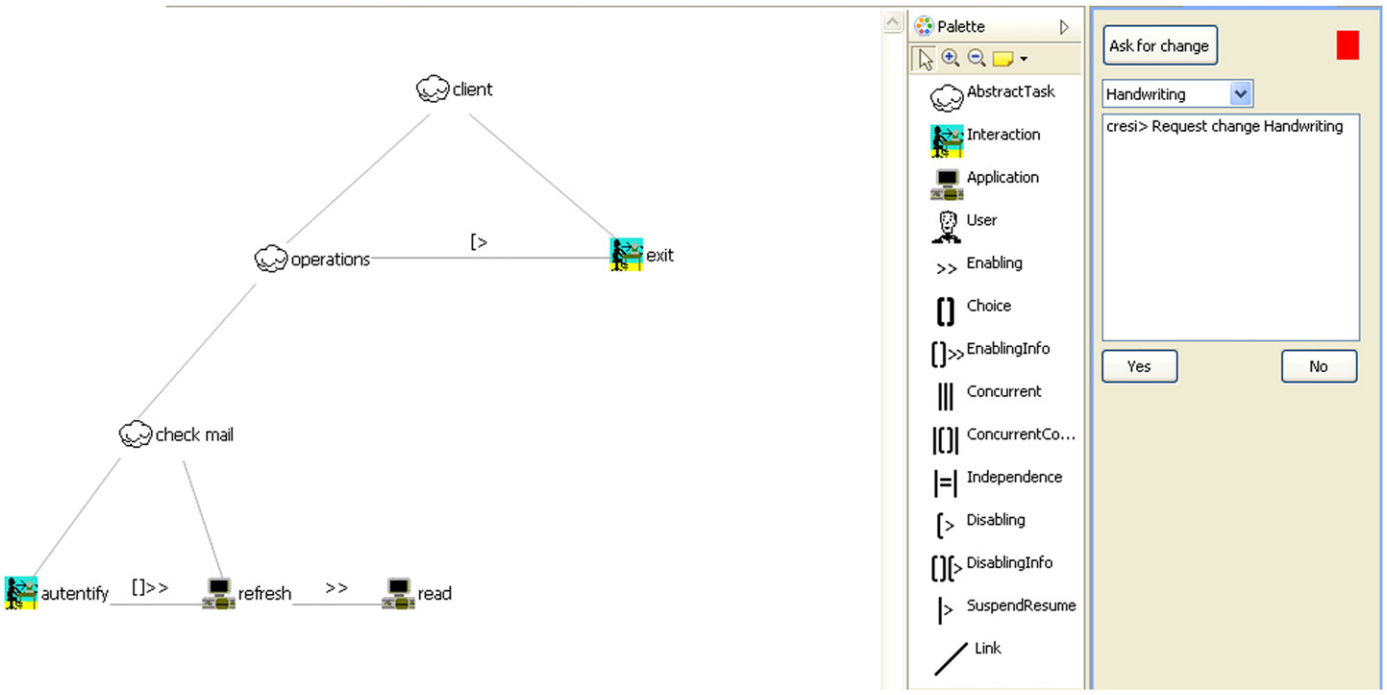

Fig. 15. Modeling of user tasks using the CTT notation with a tool generated using our development method.

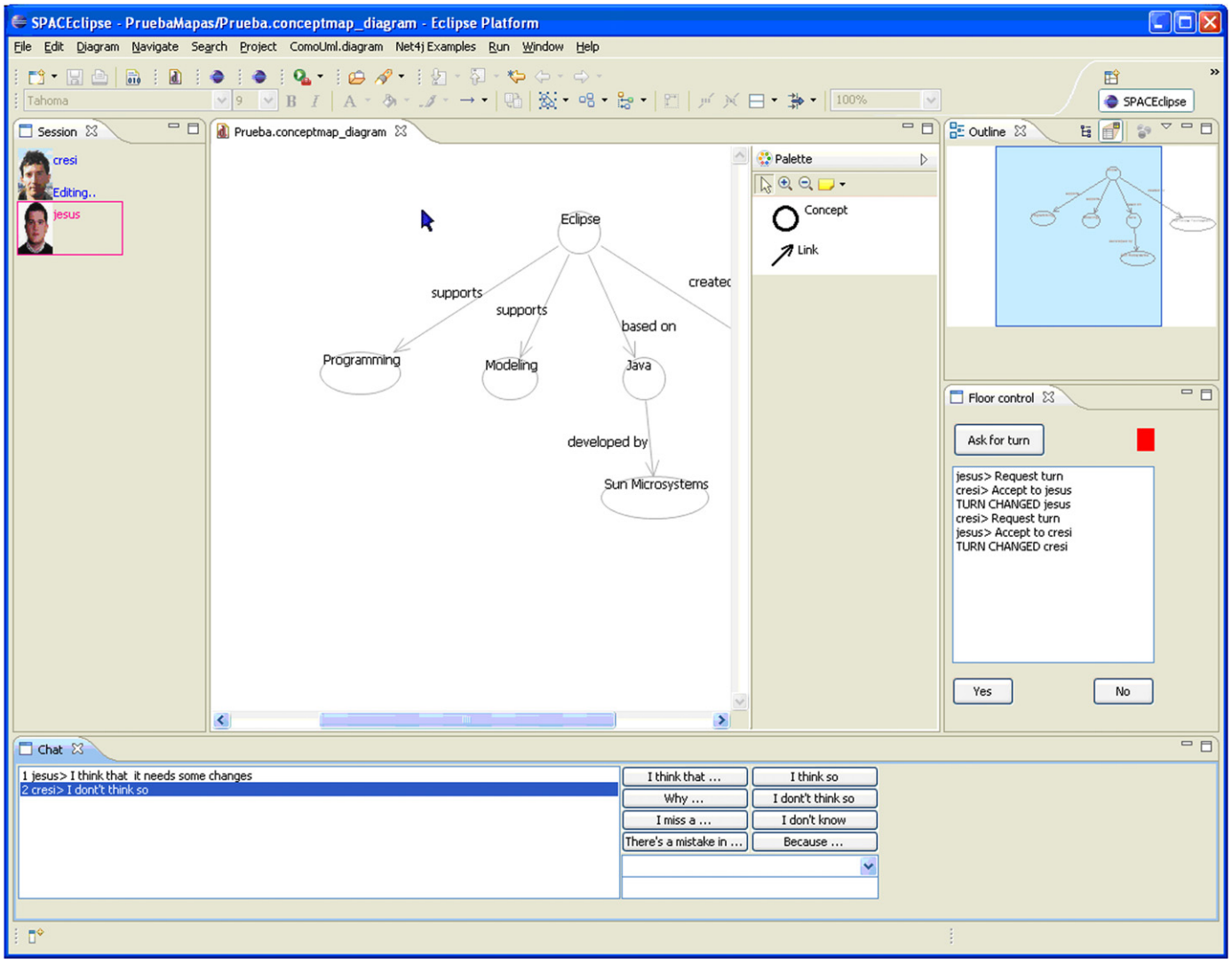

Fig. 16. Tool for the modeling of Concept Maps generated using our development method.

Engineering at the University of Castilla-La Mancha. To be specific, fourteen students took part in these studies. The studies were carried out in the scope of a subject named *Collaboration Systems*. Thus, students counted on some background in the fields of such systems, and also on some knowledge of Software Engineering and Computer–Human Interaction. This made it easier for them to get to know the goals that were desired from the studies.

Two activities were carried out within this study. The first one consisted of the modeling of an application domain starting from our application domain meta-model and using the EMF built-in tree editor. The second one implied the use of a tool generated by the development method. For both activities, the way of carrying them out implied: (i) introducing users with the technology to be used, the method it was framed within and the specific activity; (ii) carrying out the activity in one of the computer laboratories of the Computer Engineering School of the University of Castilla-La Mancha; and (iii) filling out some questionnaires in which users expressed their impressions after the activity had been carried out. Thus, after the studies were completed, the questionnaires were analyzed so that we could test the suitability of the method according to the opinions of the participants.

5.1. Application domain modeling

The first step in this evaluation activity was to introduce the concepts of *collaborative modeling system* and *application domain* to users, so that they acquired the knowledge needed to carry out the activity properly. Then, the different concepts that are included in our application domain meta-model were explained in detail, as students had to instantiate them later. As a final introductory issue, an example of an application domain model was shown to the students so that they got to know what kind of model they had to produce.

After this preliminary work was finished, students were prepared to do the modeling of a specific application domain. The chosen one was the CTT notation for task modeling (Paternò, 2004), which we talked about previously in this paper. Students had 45 min to do the modeling of this domain using a tree editor generated by EMF. Fig. 17 shows an example of an application model domain created during the study.

Once the study had taken place, participants answered a questionnaire in which they had to give their opinion about the activity they had completed. The questionnaire consisted of eight items and each one was given a value from 1 to 5, with 1 being the lowest value and 5 the highest. Table 3 shows some statistics data from the questionnaires.

By regarding the results of the questionnaires, some conclusions can be drawn. A positive one is the high evaluation received by the Eclipse-based approach and the ease of use of the tool, which leads us to conclude that it is not necessary to build another kind of tool to model application domains (e.g., a graphical tool). The elements included in the meta-model received a good evaluation. However, participants criticized the

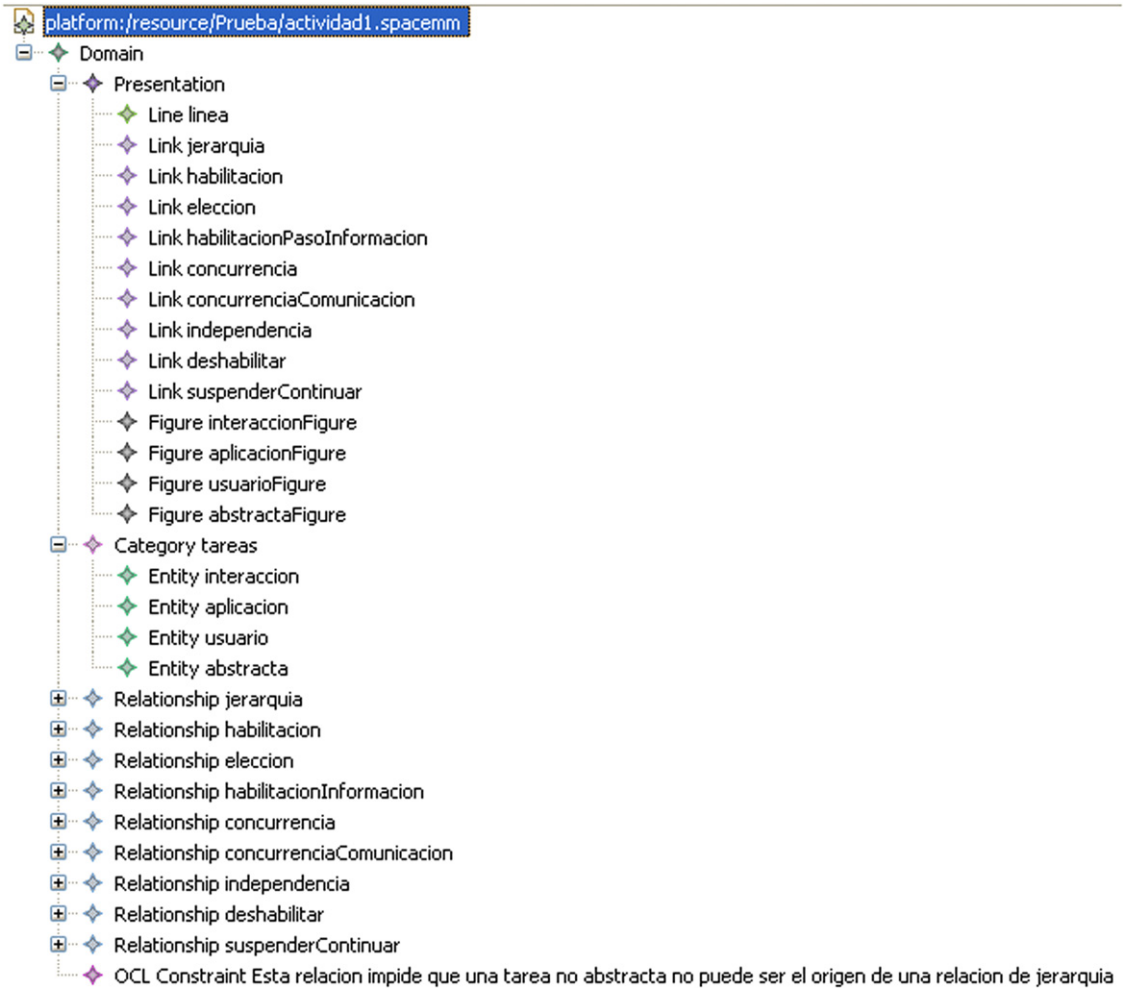

Fig. 17. Example of application domain model created by a student during the study.

hierarchy of the concepts in the model, so this could be a possible improvement in future versions of the meta-models. Finally, another interesting statistic is the low evaluation of the expressive power of the application domain modeling. This means that participants did not think that any application domain could be modeled by using the meta-model. Hence, future revisions of the meta-model should be provided with more expressive power so that more application domains can be modeled.

Table 3
Results of the questionnaries about the application domain modeling activity.

Question	Mean	Standard deviation	Mode
1.- Comprehension of the modeling	3000	1.155	3
2.- Ease of use of the tool	3538	1.198	4
3.- Expresive power of the application domain modeling	2615	0.870	2
4.- Suitability of the elements in the meta-model	3231	1.092	4
5.- Suitability of the concepts hierarchy	2769	1.301	3
6.- Advantage of the use of Eclipse	3769	0.832	4
7.- Possibility of creation of graphical tools	3154	0.987	3
8.- Enough time to carry out the activity	3231	1.235	4

5.2. Use of a collaborative tool generated by the method

The second activity included in the empirical study consisted of making the participants use a collaborative tool generated by the development method and solving a brief modeling problem with it. Thus, a collaborative tool to work with the COMO-UML notation defined in the AMENITIES approach (Garrido et al., 2007), which we already talked about in Sections 4.2 and 4.4, was generated using the method. This notation was selected because the activity was framed into the aforementioned *Collaborative Systems* subject. Fig. 18 shows a screenshot of the generated tool.

Thus, the initial considerations when introducing participants to this activity implied an explanation of how to use the tool, together with a description of its main components. Then, the specific activity to be solved by using the tool was explained. The 14 students that were taking part in the study were randomly organized in pairs to make up 7 work sessions. Students were forbidden to talk to each other so the chat room was the only method of communication. The length of the activity was one hour and, as it was done in the domain modeling activity, participants answered some questions so that their opinions about the tool and the study were reflected. Table 4 shows the statistical values for the questions that the participants answered.

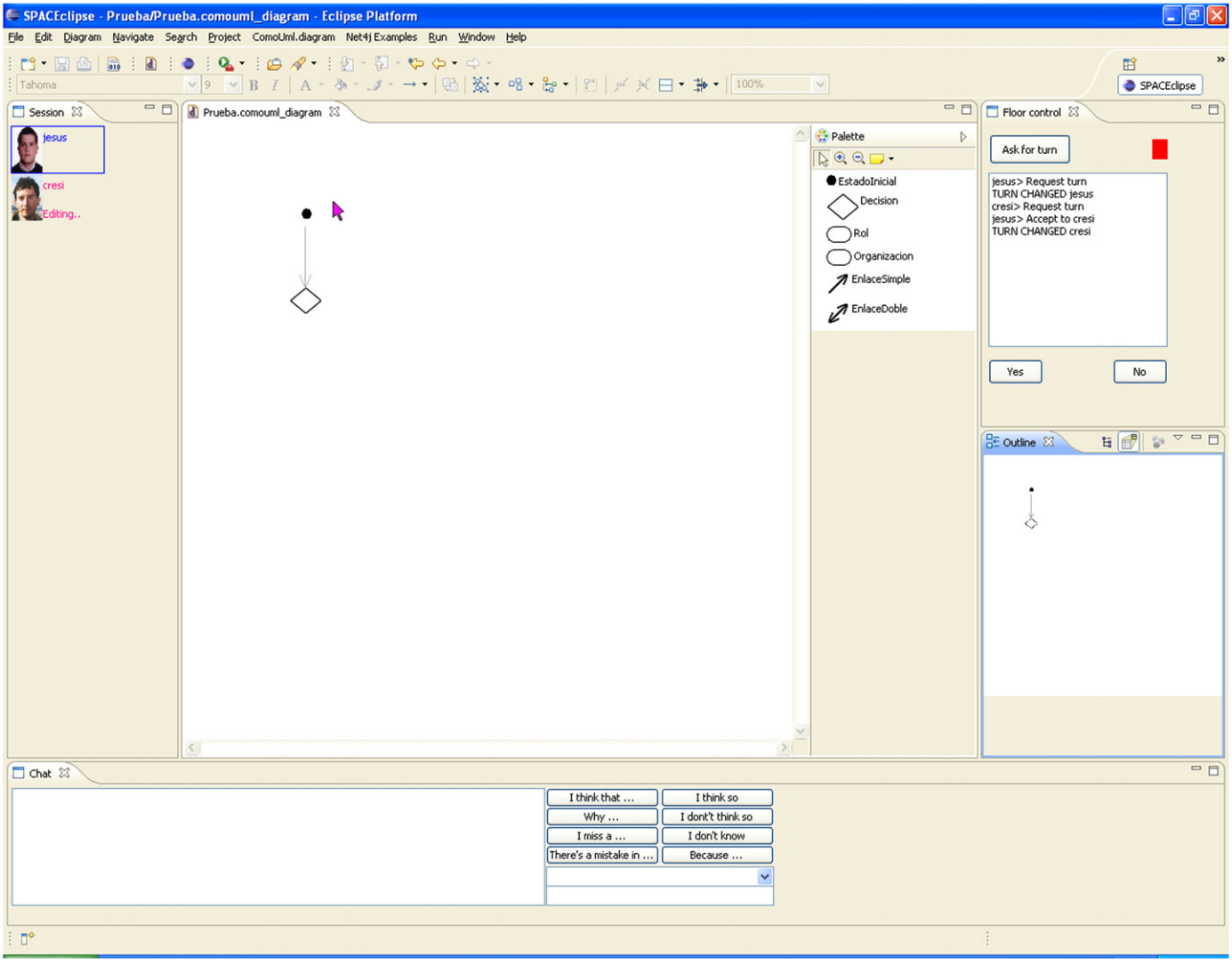

Fig. 18. Screenshot of the collaborative tool used in the study.

Table 4
Results of the questionnaires about the activity of use of the tool.

Question	Mean	Standard deviation	Mode
1.- Ease of use of the tool	3214	0.893	3
2.- Suitable object representation	4143	1.099	5
3.- Suitable relationship representation	3929	0.917	3
4.- Advantage of the use of Eclipse	3714	1.139	3
5.- Advantage of the integration of tools	3571	1.089	4
6.- Increase of difficulty if the group consisted of more than two people	2929	1.141	4
7.- Awareness information provided by the telepointers	4214	0.802	5
8.- Awareness information provided by the session panel	4357	0.745	5
9.- Awareness information provided by the structured chat	3643	1.598	5
10.- Awareness information provided by the radar view	4143	0.949	5
11.- Usefulness of the state label in the session panel	3929	1.207	5
12.- Usefulness if intentionality information was included	3643	0.929	3
13.- Self-explanatory user interface	2857	0.949	3

The results of the questionnaires are quite satisfactory as users evaluated most items with high values. One remarkable issue to take into account is that users regarded the representation of the objects as a good idea, but did not think the same about the representation of the relationships. Some other interesting results to consider are the ones referring to the awareness information provided by the different awareness support elements in the user interface. Users gave high values to all awareness support elements, so the election of these elements to give proper awareness support seems to be a good idea. Finally, two weak points can be identified. One is the possible difficulty when working in groups made up of more than two people; a possible solution for this could be to add a different floor control mechanism. The second weak point is the answer to question 13, in which users think that the user interface is not as self-explanatory as would be desirable. Thus, some changes in the user interface should be made in future versions of the technological framework to make the generated applications more usable.

6. Conclusions and future work

In the project that this article documents, we have studied how to systematize the development of groupware applications, specifically those applications that are used to support collaborative modeling tasks. For this purpose, we have proposed a method that allows users to develop groupware tools and that takes into consideration the specific characteristics of this type of software. The method is based on a model-driven software development approach and is organized into three frameworks: a conceptual framework, a methodological framework and a technological framework, which together provide a guide for the correct application of the method. Using the meta-models that make up the conceptual framework, a wide amount of application domains and their modeling tasks can be modeled. This is achieved through a well-defined set of steps in accordance with the methodological framework, and is supported by a technological framework based on the Eclipse platform. From the point of view of the user of the method, the key issue is the definition of a proper set of models that define the application domain and the structure of the tool to be generated. On an internal level, two relevant transformation processes are carried out: an M2M transformation process that we have developed to generate GMF models, and am M2T transformation process to generate the tool, which is the classical GMF process with some extensions which we have implemented.

The main contribution of this method is, at the conceptual level, the obtaining of a set of models for specifying both the application domain and the workspaces which make up the modeling tool. At the methodological level, our contribution is the set of steps which should be followed to create the collaborative modeling tools while making use of the completed models and technology. And at the technological level, we contribute to the integration and extension of different technologies from the Eclipse Modeling Project in order to obtain collaborative functionality in the generated tools and to make the tool specification process easier. We have also extended this technology to incorporate mechanisms for communication, coordination and the replication of actions, along with some awareness techniques that are typically found in groupware applications.

In order to check that our method can be successfully applied to a real-life situation, we have created tools for collaboratively developing Use Case diagrams, ConcurTaskTrees, Concept Maps, etc. These examples were explained in Section 5. Especially remarkable is the simple way in which new domains are approached, which consists of building a domain model with the participation of one or more domain experts. Then, on top of this model, some transformations that we have developed are applied in order to generate the collaborative modeling tool especially adapted for this particular domain. The method has also been used with real users in a study that has evaluated, on the one hand, the mechanism for defining application domains and, on the other hand, the applications that are generated by the method. The results of the study are satisfactory, although some weak points of the method have been identified. These weak points will be dealt with in future versions of the different frameworks of the method.

Therefore, we have shown how the development of collaborative modeling tools can be systematized by means of a model-driven approach which allows non-expert end users to benefit from it. In addition, we have also verified that the development of these kinds of applications using model-driven engineering techniques is possible. Finally, we have also shown how this entire process can be supported in terms of technology by a platform that is in widespread use in the field of software development, such as the Eclipse platform. Thus, the technological framework is dependent on the Eclipse technology but the remaining elements in the method can be applied if a different technology is used to implement technological issues.

A current limitation of our development method, and in particular of our domain modeling phase, is that not every application domain can be represented using the domain definition meta-model. As such, we are working to provide it with more semantic and syntactic power adding, for example, the possibility of having entities that are composed of other entities. We are also working on specifying design constraints: the maximum cost of a model, limiting the number of entities to be used, the time

available for building the design, or other constraints that could perhaps be considered when making the model.

Another limitation of the conceptual framework is the lack of any mechanism to express more semantically powerful collaboration protocols, i.e. protocols which allow for a collaborative system that is made up of a set of workspaces, some of which can be executed simultaneously. For example, in a software development project, the engineers who develop the activity diagrams may be divided into groups with each group developing one diagram at a time while the other groups develop the rest of them. An additional improvement would be to have more workspaces added to the workspace meta-model in order to make more powerful applications possible.

In terms of work that is currently in progress, we are including in the conceptual framework the possibility of having roles associated with the users in each working session. Each role will be able to do different things during the session. For example, a role could be limited to only adding to the model entities belonging to certain categories. There could also be supervisory roles that, instead of modifying the model, simply watch the modeling process and participate in discussions via the chat feature. This could be used, for instance, in a CSCL (Computer-Supported Collaborative Learning) environment to implement a *teacher* role.

Also, the possibility of extending the method to work with 3D models is being considered. This will imply developing an extended application domain meta-model in which 3D objects can be modeled. However, the technological support given by GMF will not be enough to carry out such modeling tasks so a different technological framework should be developed for this purpose.

Finally, we are looking at carrying out further evaluations with experts to obtain better feedback about possible improvements to the conceptual and methodological framework.

Acknowledgments

This work has been partially supported by the mGUIDE (PBC08-0006-5212) and INTEGroup (PPII11-0013-1219) projects, funded by the Junta de Comunidades de Castilla-La Mancha (Spain).

References

Avouris N, Margaritis M, Komis V. Modelling interaction during small-groups synchronous problem-solving activities: the Synergo approach. In: Proceedings of the 2nd International Workshop on Designing Computational Models of Collaborative Learning Interaction. Maceio, Brazil; 2004.

Bézivin J. On the unification power of models. Software and System Modeling (SoSyM) 2005;4(2):171–88.

Bravo C, Gallardo J, García-Minguillán B, Redondo MA. Using specifications to build domain-independent collaborative design environments. In: Luo Y, editor. Cooperative Design, Visualization and Engineering, LNCS 3190. Berlin: Springer; 2004. p. 104–14.

Bravo C, Redondo MA, Ortega M, Verdejo MF. Collaborative distributed environments for learning design tasks by means of modelling and simulation. Journal of Network and Computer Applications 2006;29(4):321–42.

Cook S. Domain-specific modeling and model driven architecture. MDA Journal 2004;January.

Schmidt DC. Model-driven engineering. IEEE Computer 2006;39(2):25–31.

Dourish P, Bellotti V. Awareness and coordination in shared workspaces. In: Proceedings of the Conference on Computer Supported Cooperative Work CSCW92. New York: ACM Press; 1992.

Duque R, Bravo C. Analyzing work productivity and program quality in collaborative programming. In: Proceeding of The Third International Conference on Software Engineering Advances, 2008 (ICSEA '08). IEEE Computer Society, pp. 270–276; 2008.

Duque R, Bravo C, Ortega M. A model-based framework to automate the analysis of users' activity in collaborative systems. Journal of Network and Computer Applications 2011;34(4):1200–9.

Favre JM. Towards a basic theory to model modeldriven engineering, in: Proceedings of Workshop on Software Model Engineering, WISME 2004, joint event with UML2004; 2004.

Flügge M. Dawn Project, 〈http://wiki.eclipse.org/Dawn〉 (visited on November 15, 2010).

France R, Rumpe B. Model-driven development of complex software: a research roadmap, in: Proceedings of FOSE'07. IEEE Computer Society; 2007.

Gallardo J, Bravo C, Redondo MA. An ontological approach for developing domain-independent groupware, In: Proceedings of the 16th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE 2007). IEEE Computer Society, pp. 206–207; 2007.

Gallardo J, Molina AI, Bravo C, Redondo MA, Collazos CA. An ontological conceptualization approach for awareness in domain-independent collaborative modeling systems: application to a model-driven development method. Expert Systems With Applications 2011a;38(2):1099–118.

Gallardo J, Molina AI, Bravo C, Redondo MA, Collazos CA. Empirical and heuristic-based evaluation of collaborative modeling systems: an evaluation framework. Group Decision and Negotiation 2011b;20(5):535–62.

Garrido JL, Noguera M, González M, Hurtado MV, Rodríguez ML. Definition and use of computation independent models in an MDA-based groupware development process. Science of Computer Programming 2007;66:25–43.

Gerosa MA, Pimentel M, Raboso AB, Fuks H,de Lucena, CJP. Towards an engineering approach for groupware development: learning from the AulaNet LMS development. In: Proceeding of the 9th International Conference on CSCW in Design. IEEE Computer Society, pp. 329–333; 2005.

Giraldo WJ, Molina AI, Collazos CA, Ortega M, Redondo MA. Model based approach for GUI development in Groupware Systems. In: Briggs RO, Antunes P, de Vreede GJ, Read A, editors. Groupware: Design, Implementation, and Use, 14th International Workshop, CRIWG 2008, LNCS 5411. Berlin: Springer; 2008. p. 324–39.

Greenfield J. Bare Naked Languages or What not to Model, The Architecture Journal 9. Microsoft Corporation, Redmond, WA; 2005.

Grudin J. Groupware and cooperative work: problems and prospects. In: Baecker RM, editor. Readings in Groupware and Computer Supported Cooperative Work. San Francisco, CA: Morgan Kaufman Publishers; 1993. p. 97–105.

Kim K. A model-driven workflow fragmentation framework for collaborative workflow architectures and systems. Journal of Network and Computer Applications 2012;35(1):97–110.

Laviola JJ, Holden LS, Forsberg AS, Bhuphaibool DS, Zeleznik RC. Collaborative conceptual modeling using the SKETCH framework. In: Proceedings of the 1998 IASTED International Conference on Computer Graphics and Imaging, pp. 154–157; 1998.

Molina AI, Redondo MA, Ortega M. A methodological approach for user interface development of collaborative applications: a case study. Science of Computer Programming 2009;74:754–76.

Paternò F. ConcurTaskTrees: an engineered notation for task models. In: Diaper D, Stanton NA, editors. The Handbook Of Task Analysis For HCI. Mahwah, NJ: LEA; 2004. p. 483–501.

Pinkwart N, Hoppe U, Bollen L, Fuhlrott E. Group-oriented modelling tools with heterogeneous semantics. In: Cerri S, Gouardères G, Paraguaçu F, editors. Intelligent Tutoring Systems, LNCS 2363. Berlin: Springer; 2002.

Redondo MA, Bravo C, Ortega M, Verdejo MF. Providing adaptation and guidance for design learning by problem solving: the design planning approach in DomoSim-TPC environment. Computers & Education 2007;48(4):642–57.

Roseman M, Greenberg S. Building real-time groupware with group kit, a groupware toolkit. ACM Transactions on Computer–Human Interaction 1996;3(1):66–106.

Schuckmann C, Kirchner L, Schümmer J, Haake JM. Designing object-oriented synchronous groupware with COAST. in: Proceedings of the 1996 ACM conference on Computer supported cooperative work. ACM Press, pp. 30–38; 1996.

Snowdon D, Benford S, Greenhalgh C, Ingram R, Brown C, Lloyd D, Fahlén L, Stenius M. A 3D collaborative virtual environment for web browsing, In: Proceedings of the Virtual Reality WorldWide'97; 1997.

Trujillo S, Batory D, Díaz O. Feature oriented model driven development: a case study for Portlets. In: Proceedings of the 29th International Conference on Software Engineering, pp. 44–53; 2007.

van Joolingen WR, de Jong T, Lazonder AW, Savelsbergh ER, Manlove S. Co-Lab: research and development of an online learning environment for collaborative scientific discovery learning. Computers in Human Behavior 2005;21:671–88.