Closing the gap between Organizational Modeling and Information System Modeling[1]

Alicia Martínez[1,2], Jaelson Castro[3], Oscar Pastor[1], Hugo Estrada[1,4]

[1] Technical University of Valencia
Avenida de los Naranjos s/n, Valencia, Spain
{alimartin, opastor, hestrada}@dsic.upv.es
[2] I.T. Zacatepec, Morelos, Mexico
[3] UFPE Recife, Pernanbuco, Brazil jbc@cin.ufpe.br
[4]CENIDET Cuernavaca, Mor. Mexico

Abstract. The creation of a conceptual schema is a critical feature of the software production process. The conceptual schema should represent the structure and behavior of the information system so that the users can perform their organizational tasks. For this reason, the organizational context needs to be the starting point for the generation of an initial conceptual schema. This approach allows us to assure that the functionality of the information system will be equivalent to the tasks that are executed in the business. However, only a few research studies offer a systematic approach for carrying out the equivalence between models. In this paper, a methodological approach for deriving conceptual schemas from TROPOS business models is presented. The resultant conceptual schema will be the input of the OO-Method Case Tool, which implements the automatic software production process. By doing this, we go a step further in the process of including business modeling as a key piece in the software production process.

1. Introduction

Software development is an activity that is increasingly complex and that requires powerful techniques of elicitation, specification and development. Software engineering has proposed several techniques, methodologies and tools to achieve the goal of developing information systems that comply with user needs. Nevertheless, today there still exists a real impedance mismatch between the software product and its operational environment. At present, it is possible to generate software systems that are implemented correctly; however, it is not yet possible to assure the construction of the appropriate software product. The non-correspondence between the information system and its operational environment make it impossible for the information system to have the necessary functionality to permit the organizational actors to perform their organizational tasks.

[1] This project is partially funded by the Asociación Nacional de Universidades e Instituciones de Educación Superior ANUIES, Mexico

The impedance mismatch between the user needs and the functionality of the information system can be reduced by extending software engineering with business engineering. Business engineering should consider the problems related to the understanding of the organizational environment, while software engineering should consider the problems related to improving the quality and reducing the costs in the software production process. Both approaches have been analyzed in depth separately.

In the field of software production processes, there are interesting proposals [And96], [Sch98], [Pas01] which try to guide the translation of the problem space to the solution space. Their goal is to precisely specify the mapping between the relevant concepts of an object-oriented conceptual schema into their corresponding software representations. These works focus on describing the aspects related to the implementation of the information systems. However, they do not take into account the organizational modeling to determine the correct requirements for developing the software product.

In the field of business modeling, there are also interesting proposals [Bub95], [Col00], [Kol03], [Kou00] which focus on specifying the semantic of a business process in a very precise way. However, they do not provide effective solutions to the problems of methodological and systematic translation of the business model into information system specifications. The lack of mechanisms for defining the traceability between models leads to a translation process which is costly in both time and effort.

At present, there are very few research studies which focus on the problem of the mapping process between the business model elements and the conceptual schema elements of the information system. This is a fundamental task in the software development process. It is necessary for the translation between models to be carried out in a methodological process to assure its application in real software development environments.

In this paper, we propose the use of the business model as a starting point in order to obtain a conceptual schema of the information system. To do this, we propose joining two well-known techniques: The Tropos Framework from the field of business modeling and the OO-Method Case Tool from the field of software production process.

The paper is structured as follows: Section 2 presents the methodologies used in this paper. Section 3 presents an overview of the proposed method. Section 4 presents the conceptual schema generation method. Finally, section 5 presents the conclusions and further work.

2. Methodologies

This section presents the methodologies used in this paper: The Tropos Framework and the OO-Method. Both approaches are combined to obtain the conceptual schemas of an information system.

2.1 The Tropos Methodology

Tropos [Cas02] proposes a software development methodology and a development framework which are based on concepts used to model early requirements. They are based on the premise that in order to build software that operates within a dynamic environment, it is necessary to analyze and explicitly model that environment in terms of actors, their goals and their dependencies on other actors.

To support modeling and analysis during the early requirements, Tropos adopts the concepts offered by i* [Yu95], a modeling framework defined in terms of concepts such as actors and social dependencies among actors, including goal, softgoal, task and resource dependencies.

In Tropos, we have the following concepts:

- Actor: This is an actor is an active entity that carries out actions to achieve goals by exercising its know-how.
- Dependency: This describes an intentional relationship between two actors. It is composed by: a) *Depender*: the actor who is dependent on another actor, b) *Dependee*: the actor on whom another actor depends, c) *Dependum*: the task, goal, resource or softgoal on which the relationship is focused.
- Goal dependency: This a dependency in which an actor depends on another actor to fulfill a goal, without prescribing the way in which it should be carried out.
- Resource dependency: This is a dependency in which an actor depends on another actor to deliver a resource that can be either material or informational.
- Task dependency: This is a dependency in which an actor depends on another actor to carry out a task, establishing the way in which it should be performed.
- Softgoal dependency: This is similar to the goal dependency, with the difference that the goal has not been precisely defined.

By using these elements, it is possible to create the i* Strategic Models: the Strategic Dependency Model and the Strategic Rationale Model.

The *Strategic Dependency Model* (SD) shows the dependencies that exist between actors in a business process. The model is represented by a graph where nodes represent actors, and where the links represent the dependencies that exist between these actors to achieve their goals, carry out tasks and provide or request resources.

The *Strategic Rationale Model* (SR) carries out a deeper reasoning of the motives that exist behind each dependency relationship. This is useful for representing tasks that have to be carried out by the actors to achieve the goals which are expected of them, as well as for rethinking new ways of working. This model is based on the elements of the dependency model, adding a) task decomposition links which allow us to represent the combination of necessary tasks to achieve a goal, and b) mean-ends links whose objective is to present the diverse alternatives that can be taken to fulfill a task or goal.

The Tropos methodology has been used in several application areas, including requirements engineering, software processes and business process reengineering. However, in Tropos Methodology, there is still no method for using the business models to produce information system in an automatic way.

2.2 The OO-Method Case Tool

The OO-Method is a Case Tool to automate the software production process. It was created on the formal basis of OASIS, an object-oriented formal specification language for Information Systems. It is possible to distinguish two modeling components in OO-Method: the conceptual model and the execution model. The conceptual model specifies what the system is (problem space) in a notation based on UML, and the execution model guides the representation of these requirements in a specific software development environment (solution space), which is centered on how the system will be implemented.

The abstract execution model is based on the concept of the Conceptual Modeling Construct. The OO-Method provides a well-defined software representation of these constructs in the solution space. A concrete execution model based on a component-based architecture has been introduced to deal with the peculiarities of the component-based system. The implementation of this mapping from problem space concept to solution space representations opens the door to the automatic generation of executable software components. These software components together constitute a software product that is functionally equivalent to the requirements specification collected in the conceptual modeling step. In this proposal, the OO-Method is used to generate information systems from the conceptual schemas derived from the business model.

3. Overview of the proposed method

The strategy of our conceptual schema generation method consists in isolating the relevant information from the business model and using it to generate the elements of the information system. To do this, the software system actor must be inserted inside the original business model. Then, it is necessary to determine the relevant business goals[2] to be automated by the software system. Afterwards, the tasks and resources needed to accomplish the goal are then redirected toward the system actor. Therefore, the satisfaction of the goals will not be altered; only the actor responsible for its fulfillment is modified. The internal tasks in the software system actor must be defined in order to satisfy its goals. Finally, the information contained in the software system actor is used to create the elements of an object-oriented conceptual schema for the OO-Method Case Tool. The OO-Method tool translates the elements of the conceptual schema into elements of an imperative program in a specified target language.

In order to illustrate our approach, we used the *Conference Review Process* case study. The purpose is to model the business process of this type of review system in order to obtain a software system that handles the process of submission, assignment, evaluation and selection of papers for a conference.

[2] A goal dependency is relevant if its fulfillment depends on actions carried out by the system actor.

4. Conceptual Schema Generation Method based on Organizational Models

In this section, we describe in detail the steps presented in the previous section. Section 4.1 shows how to use the organizational model to obtain the relevant information to be automated. Section 4.2 shows the specification process of organizational models. Section 4.3 shows the method to translate the organizational model specification into a conceptual model of the information system. The case study is used to show the application of the method.

4.1 Selecting the relevant information to be automated

The first step of the method is the selection of a business model represented in the Tropos Framework. In our case study, the following events are presented: the Chair of the Program Committee (*PcChair*) determines the topics of interest and selects the members of the program committee (*PcMember*). The members can delegate the responsibility of review to additional reviewers (*Reviewers*). Finally, the *PcMembers* and Reviewers send the evaluations to the *PcChair* indicating acceptance or rejection. Figure 1 shows the Strategic Rationale Model for the case study.

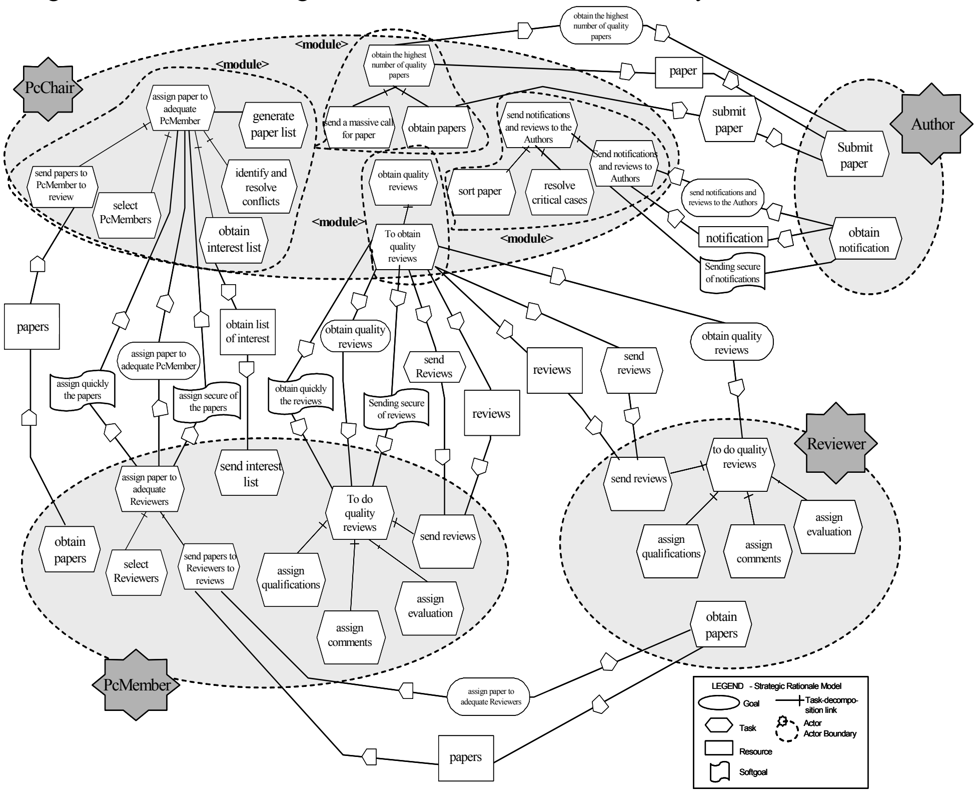
PcChair
Author
Reviewer
PcMember
<module>
obtain the highest number of quality papers
paper
submit paper
Submit paper
assign paper to adequate PcMember
generate paper list
identify and resolve conflicts
send papers to PcMember to review
select PcMembers
obtain interest list
send a massive call for paper
obtain papers
send notifications and reviews to the Authors
sort paper
resolve critical cases
obtain quality reviews
To obtain quality reviews
notification
obtain notification
Sending secure of notifications
papers
obtain list of interest
assign quickly the papers
assign secure of the papers
obtain quickly the reviews
Sending secure of reviews
send Reviews
reviews
send reviews
send interest list
assign paper to adequate Reviewers
To do quality reviews
to do quality reviews
select Reviewers
send papers to Reviewers to reviews
assign qualifications
assign comments
assign evaluation
LEGEND - Strategic Rationale Model
Goal
Task
Resource
Softgoal
Task-decomposition link
Actor
Actor Boundary

Figure 1. Strategic Rationale Model for the case study *Conference Review Process*

The next step consists of determining the type of interaction of each organizational actor model with the software system actor. This is called *Conference Review system* in our case study. An important concept used in this process is the concept of "module". A module represents the set of tasks performed by the actor to satisfy its goals with another actor. The *modules* are represented by internal task-refinement trees in the actors of the Strategic Rationale Model. An actor may have more than one *module*. This indicates that the actor should fulfill more than one goal in the organizational model. In our case study, the *PcChair* has the *modules*: *assign paper to adequate PcMembers, obtain the highest number of quality papers, obtain quality reviews and send notifications and reviews to the Authors* (Figure 1).

We present guidelines to insert the actor system into the organizational model. Figure 2 shows the result of the application of these guidelines to our case study.

Guideline 1. To insert the actor system into the organizational model and to identify the modules that need to be delegated to the information system. To identify the modules that need to be redirected towards the software system actor, it is necessary to do a softgoal analysis. In i* notation, the softgoal represents the desired qualities to satisfy a goal (speed, security, performance) and therefore to perform the module. It is then possible to use the softgoal associated to each of the goal dependencies to determine whether they should be automated. For example, in our case study, one of the modules of the *PcChair* is *assign paper to adequate PcMembers.* The goal dependency associated to this module is "*assign paper to adequate PcMembers*". At the same time, the softgoals associated to this goal are "*assign the papers quickly*", "*safe assignment of pa*pers". Based on these requirements, these goals need to be automated using the information system.

Guideline 2. To move the modules selected from the organizational actors to the system actor. This is only necessary when the main task (module root) needs to be automated. In the case in which only one subtask of the module needs to be automated, the task is considered to be the root of the module, and the necessary subtasks for accomplishing the goal must be defined.

To move each module, there are two steps. The first step is to create a copy of the module root in the system actor. The second step is to create a task dependency (with the same name as the module) between the organizational actor and the system actor. This task dependency indicates that the software system actor is now responsible for completing the task. There may be manual operations in the modules, where the system can only be used to send or receive information. In these cases, it is necessary to leave these manual operations in the modules of the organizational actor.

To move the modules from the organizational actor to the software system actor, the name of the actor that was the container of the original module must be included in the module.

Guideline 3. There are tasks that require information from the organizational actors when these tasks are transferred to modules in the system actor. In this case, it is necessary to create new resource dependencies between the system actor and the organizational actors. This is the case with the task *Generate PcMember list*, which requires information from the *PcChair* (when transferred to the system actor) as the system cannot generate the *PcMember List* by itself.

Guideline 4. When resource dependencies are created, it is necessary to create new tasks for sending and receiving resources. We recommend creating only task and resource dependencies between the organizational actor and the system actor. This avoids putting goal dependencies that would later be derived in task or resource dependencies.

As a result of the process of selection of relevant information, a new organizational model is created (Figure 2). The following are placed in this new model: a) the actors with dependencies with the software system actor, b) the resources and task dependencies between the organizational actor and the software system actor, and finally c) the goal dependencies that have been derived in task and resource dependencies between the organizational actors and the software system actor,

In this way, this model represents the interaction between the information system and the final users. The softgoal dependencies, which are used to select the relevant information to be automated, are not shown in this new organizational model.

4.2 Creating the specification of organizational model

In this paper, we propose using a KAOS-like specification language to represent the elements of the new organizational model. KAOS [Dar93] is a formal framework based on temporary logic to elicit and represent the goals that the system software should achieve.

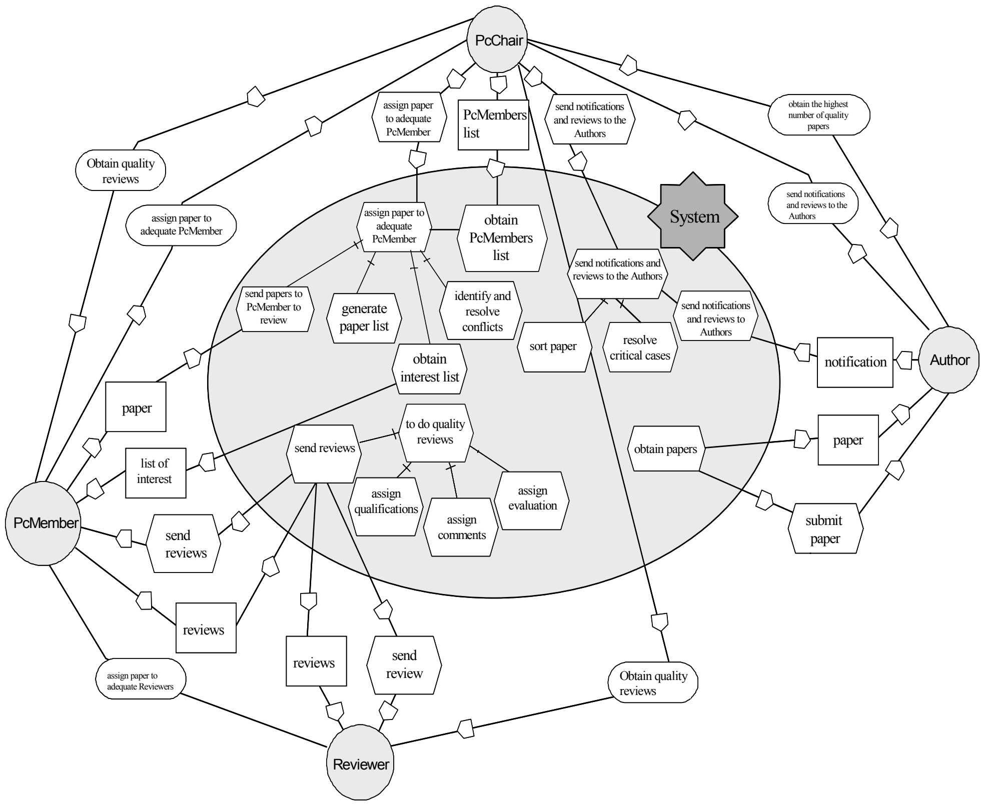
PcChair
assign paper to adequate PcMember
PcMembers list
send notifications and reviews to the Authors
obtain the highest number of quality papers
Obtain quality reviews
assign paper to adequate PcMember
send notifications and reviews to the Authors
System
assign paper to adequate PcMember
obtain PcMembers list
send notifications and reviews to the Authors
send papers to PcMember to review
generate paper list
identify and resolve conflicts
sort paper
resolve critical cases
send notifications and reviews to Authors
notification
Author
obtain interest list
paper
to do quality reviews
send reviews
obtain papers
paper
list of interest
assign qualifications
assign comments
assign evaluation
submit paper
PcMember
send reviews
reviews
reviews
send review
assign paper to adequate Reviewers
Obtain quality reviews
Reviewer

Figure 2 Insertion of the software system actor into the organizational model

In spite of the fact that KAOS was developed to represent information system goals, its syntax and semantics turn out to be especially appropriate for formally representing the conceptual primitives of the strategic models of the framework i*. This is due to the capacity of KAOS to represent modeling concepts at diverse abstraction levels.

In this paper, we define a set of specification rules to specify each one of the elements of the Tropos organizational model in the KAOS-like language. The specification rules are explained in the following subsections.

4.2.1 Actor Specification

The actors of the organizational model are represented using the concept of *Agent* of the KAOS language. An agent is an object which is a processor for some actions [Dar93]. The definition of an agent is composed of the agent name as well as a list of attributes. We have included, by default, an attribute with the same name as the actor to indicate the identification attribute of the actor.

In our case study, there are four actors with dependency relations with the software system actor: *PcChair*, *PcMember*, *Reviewer* and *Author*. Table 1 shows the specification of the actors *PcMember* and *Author* in KAOS language.

Agent PcMember Has PcMember, name, email, affiliation: String End PcMember
Agent Author Has Author, name, email, affiliation: String End Author

Table 1 Specification of the actors *PcMember* and *Author* in KAOS language

4.2.2 Goal Specification

The goal dependencies are represented using the concept of *SystemGoal* of the KAOS language. *SystemGoals* are application-specific goals that must be achieved by the composite system [Dar93]. The definition of the *SystemGoal* is composed of:

a) The type of the goal (*Achieve, Cease, Maintain, Prevent and Optimize* [Dar93]). In our proposal, we use *Achieve Goals* to express the fact that the goal needs to be satisfied in a current or a future state.
b) The specific category of the domain-level goal (*SatisfactionGoal, InformationGoal, SafetyGoal, ConsistencyGoal and RobustnessGoal*) declared in the clause *IntanceOf*. In this case, we use *SatisfactionGoals* to express the satisfaction of the agent request.
c) The clause *Concerns,* which links the goal with the objects. In this case, we use the clause *Concerns* to indicate the *depender* and *dependee* actors. Table 2 shows the specification of the goal "*obtain the highest number of quality papers*" in KAOS language.

SyemGoal Achieve [Obtain the highest number of quality papers] InstanceOf SatisfactionGoal Concerns PcMember, Author

Table 2 Specification of the goal *Obtain the highest number of quality papers* in KAOS language

4.2.3 Resource Dependency Specification

The resource dependencies of the organizational model are represented using the concept of *Entity* of the KAOS language. An *Entity* in KAOS is an autonomous object; its instances may exist independently from the other instances [Dar93]. The *Entity* definition is composed of the entity name as well as a list of attributes. Generally, the first of these attributes is the identification attribute of the entity.

In our case study, there are four resource dependencies between the actor and the software system actor: *Interest list, paper*, *review* and *notification*.

In the organizational model, there are resources that depend on other resources or actors to be created. In our case study, for example, a *Notification* is always linked with a *Paper,* and the existence of the *Notification* is conditioned by the action of generation a notification executed by the *PcChair*. These types of resources have a constant attribute (the name of the resource or actor to which they are linked) in their KAOS specification. The rest of the resources of the organizational model do not include a constant attribute.

Table 3 shows the specification of the resources *Paper* and *Notification* in KAOS language.

Entity **Paper** Has paperId: Long, title: String, coauthors: SetOf[Author], abstract: String, topics: String, status: String End Paper
Entity **Notification** Attribute Constant: Paper Has NotificationId: Long, PaperId: Long, sendingaddress: String, comments: SetOf[String], evaluation: String End Notification

Table 3 Specification of the resources *Paper* and *Notification* in KAOS language

4.2.4 Specification of links in resource dependencies

In the i* framework, the relationships between actors are represented by dependency links, in which there are no direct links between the actors. The dependency links connect the resources with the *depender* and *dependee* actors. The links are represented using the concept of *Relationship* of the KAOS language. A *Relationship* is a subordinate object. The existence of its instances depends upon the existence of the corresponding object instances linked by the relationship [Dar93]. The KAOS specification of a resource contains a) the resource name, b) the relation type (*Send*, *Sentto* or *Receive*) and c) the cardinalities of the relationships among the resource and the actors that send and receive it. In this way, the relationship is specified as Actor-Resource-Actor. One of the actors is obtained directly from the resource dependency with the system actor (Actor-Resource-System), as well as the resource. The other actor is obtained from the name of the actor included in the specification of the module, as was mentioned in 4.1. For example, in the resource dependency *Paper* of the case study shown in Figure 2, the *Author* sends the resource *Paper* to the Software System Actor. From the module *Obtain Papers*, we obtain the *PcChair* Actor as the original actor of the dependency. Therefore, the elements of this resource dependency are: *Author*, *Paper* and *PcChair*.

The cardinalities of the resource dependency are used to indicate that: a) an *Author* can send 1 or more *Papers*, b) a *Paper* could be sent by 1 or more *Author*s, c) a *Paper* is sent only to a *PcChair*, and finally, d) a *PcChair* can receive 1 or more *Papers*.

Table 4 presents the KAOS representation of the links in the *Paper* resource dependency between the *Author* and the *PcChair*.

Relationship
Links Author {role send, Card 1..*}
Links Paper {role sentto, Card 1..*}
	Relationship
	Links PcChair {role receive, Card 1..*}
	Links Paper {role sentto, Card 1..1}

Table 4 KAOS specification of links in the *Paper* resource dependency

4.2.5 Task Dependency Specification

The task dependencies are represented using the concept of *Action* in the KAOS language. The action specification in KAOS is composed of input parameters that permit the execution of the actions. The output parameters represent the resources generated as a result of these actions. For example, in the case study, it is possible to determine that the action *Submit Paper* generates the resource *Paper* as output of the action. The specification of the actions also contains the preconditions and postconditions of the action.

In the proposal presented in this paper, the clause *RelatedGoal* was included in the Action definition to link it with the goal that is satisfied. It also allows us to determine the *depender* and the *dependee* actors of the task dependency. For example, the task *Submit Paper* allows us to satisfy the goal *Obtain the highest number of quality papers.* From this goal, it is possible to obtain the *PcChair* and the *Author* as actors of the goal dependency, and in the same way, as actors of the task dependency. Table 5 presents the KAOS specification of the task *Submit Paper*.

```
Action SubmitPaper
Input String {Arg title}, String {Arg topics}, SetOf[string] {Arg abstract}, String {Arg status},
SetOf[String]{Arg author}, SetOf[String]{Arg coauthors}
Output Paper {Arg paper}
Precondition  RegisterAuthor (author)
Postcondition paper.title = title and paper.topics = topics and  paper.abstract = abstract and paper.status =
"sending" and paper.author = authors  and paper.coauthors = coauthors
RelatedGoal  Obtain the highest number of quality papers
End SubmitPaper
```

Table 5 KAOS specification of the *Submit Paper* task

4.2.6 Specification of internal tasks in the software system actor

The specification of internal tasks is the same as the task dependency but the clause *RelatedTasks* is included. This clause was included in our proposal to define the internal tasks in the software system actor. The internal tasks of the system actor arise as a result of moving *modules* from the organizational actors. For this reason, the internal tasks are linked to more general tasks using task decomposition links or mean-ends links. The clause *RelatedTasks* allows us to differentiate the task dependency from the task decomposition. Table 6 presents the KAOS specification of the task *Assign evaluation*. This task is linked to the "*to do quality review*" module in the *PcMember* Actor (Figure 2).

```
Action Assign evaluation
Input …
Output ..
Preconditions …
Postconditions …
RelatedTask  to do quality reviews
End SubmitPaper
```

Table 6 KAOS specification of the *Assign evaluation* task

The application of these specification rules allows us to obtain the complete specification of the organizational model. This specification will be used to generate the conceptual schema of the information system.

4.3 Generation of the conceptual schema for the information system

The generation of a conceptual schema from the organizational model (with the software system actor as part of the model) allows us to accurately and precisely determine the structure and behavior of the information system expected. In this approach, the conceptual schema elements are directly derived from the organizational elements. In this way, it is possible to assure that the functionality of the information system will be equivalent to the tasks executed in the business.

The process of conceptual model construction uses the KAOS-like specification of the organizational models as input. The KAOS-like specification is used to generate a conceptual schema specified in OASIS language, the support language of the OO-Method Case Tool. The translation process is presented in the following sections.

4.3.1 Translation of Actors

The actors with dependence relationships with the system actor are represented as classes in the conceptual schema in OASIS language.

The identification attribute of the actors of the organizational model is used to create the identification attribute of the OASIS classes, which represent these actors. The rest of the actors´ attributes are defined as variable attributes in the specification of the classes. This is a consequence of the lack of information of the organizational model to determine the stability of its attributes. For this reason, it is not possible to carry out a distinction between constant attributes and variables. In our case study, for example, the attributes *PcMemberId* and *AuthorId* are used to create the identification attribute of the classes *PcMember* and *Author*.

The mechanisms of creation and destruction of instances as well as the mechanism of modification of variable attributes are placed by default in the OASIS specification. Table 7 shows the OASIS specification for the classes *PcMember* and *Author*.

class Author
identification
    AuthorId: (AuthorId) ;
constant_attributes
    AuthorId : nat ;
variable attributes
    name : string; email : string; affiliation: string;
  ...
end_class
	Class PcMember
	identification
	    PcMemberId: (PcMemberId);
	constant_attributes
	    PcMemberId: nat;
	variable attributes:
	    name: string; email: string; affiliation: string;
	  ...
	end_class

Table 7 Specification of the actors *Author* and *PcMember* in OASIS Language

The actors´ classes of the information system are extended during the translation process of the rest of the elements of the organizational model.

4.3.2 Translation of resource dependencies

The resources of the organizational model are translated into classes of the OASIS conceptual schema. Their attributes are used to create the attributes of their corresponding classes in the conceptual model.

The existence of a constant attribute in the KAOS description of a resource allows us to create a relationship between the class of the resource described and the class of actor or resource which is specified as a constant attribute. To determine the relationship type, it is necessary to determine whether the class placed as a constant attribute is "part of" the class which contains it. In this case, the relationship is an aggregation. In other cases, the relationship could be an association relationship.

In our case study, for example, the *Notification* appears as a constant attribute of the resource *Paper* (as was commented in 4.2.3). In this case, the *Notification* is part of the *Paper*. Therefore, an aggregation between the classes *Notification* and *Paper* is created. It must be pointed out that there is no information that allows us to identify the type of aggregation or association obtained from the resource dependencies. Therefore, default values (inclusive, dynamic, univalued, disjoint, strict, notnull) are used to define the relationships, however, it is possible to redefine these values in the application of the following translation steps. The values for the associations and aggregations are shown in the next subsection (4.3.3).

Table 8 shows the OASIS specification for the classes *Paper* and *Notification*.

Complex class Paper aggregation of
Notification(inclusive,dynamic,univalued,disjoint,strict, notnull);
identification
    PaperId: (PaperId);
constant_attributes
    PaperId: Nat;
end_class
	Class Notification
	identification
	    NotificationId: (NotificationId);
	    PaperId: (PaperId);
	constant_attributes
	    NotificationId: Nat; PaperId: Nat;
	end_class

Table 8 Specification of the resource *Notification* and *Paper* in OASIS Language

4.3.3 Translation of links between actors in resource dependencies

The links and their corresponding cardinalities are used to specify relationships (according to the OO-Method Methodology [Alb03]) between the classes obtained from the actors and resources. The cardinality between the classes allows us to determine the type of the relationship: inclusive/relational, static/dynamic, univalued/multivalued, nodisjoint/disjoint, flexible/strict, null/not-null [Alb03]. The aggregations are indicated by inclusive relationships, and the associations are indicated by relational relationships. For example, using the KAOS specification shown in Table 4, it is possible to do the analysis of cardinalities shown in Table 9.

{paper}
Relationship SentBy
Links Author {role send Card 1..*} {an *Author* could have 1 or more *Papers*, then the cardinality of the class Author is 1..*}
Link Paper {role sentto Card 1..*} {a *Paper* could have 1 or more *Authors* }

Table 9 Analysis of relationships in KAOS language

From these cardinalities, it is possible to determine that the relationship is: a) relational (the *Paper* and the *Author* are not encapsulated), b) static (the *Author* is always linked with the *Paper*), c) multivalued (a *Paper* could have 1 or more *Authors*), d) nodisjoint (an *Author* could participate in several *Papers*), e) strict (an *Author* need to participate in the conference with at least a *Paper*) and, finally, f) notnull (A *Paper* cannot exist without an *Author*). Therefore, as the relationship is relational, an association is between the classes *Author* and *Paper* is created. In OASIS language, the associations are represented as relational aggregations:

```
complex class Paper aggregation of
Author (relational, static, multivalued, nodisjoint, strict, notnull);
```

4.3.4 Translation of task dependencies

The task dependencies are translated into events of the class of the actor that executes the action. The event type could be obtained from the KAOS specification following the guidelines below:

a) If a resource is generated or modified as a result of the task (the KAOS specification of the task dependency contains the name of the resource, as was mentioned in 4.2.5), then it will be necessary to determine if there is an aggregation or association between the classes of the resource and the actor. In this case, the task dependencies are translated into shared events in the class of the actor that executes the action and in the class of the resource generated by the task. This indicates that the operation invoked by the actor changes the values of the attributes of the resource class, which produces a change in the object state.
b) If a resource is not generated as result of the task, then the event will be specified as a Private event in the class of the actor.

The resource generation specification in the organizational model is translated into the event of creation of instances in the class that represents the resource in OASIS language. In our case study, we can establish that the action *Submit Paper* (executed by the *Author* actor) is the event that creates an instance of the class *Paper*. As a result, the *Author* will have the shared event *Submit Paper*, which leads to new papers in the system.

The postconditions of the actions of the organizational model are translated into valuations of the operations of the resource generated by the action. In the case study, the postconditions of the action *SubmitPaper* correspond to the valuations of the event *SubmitPaper* in the class *Paper*.

4.3.4.1 Translation of the internal task in the system actor

These tasks are translated as private events in the actor classes that execute the most general task. In our case study, the tasks *Generate PcMember List*, *Obtain Interest List*, *Obtain Paper* and *Send Paper* are translated as private operations of the *PcChair* class.

By applying the translation steps to all the organizational model elements, it is possible to obtain the specification of an OO-Method conceptual schema. Table 10 shows a fragment of the OO-Method specification for the *Paper* class.

```
complex class Paper aggregation of
 Author (relational, static, multivalued, nodisjoint, strict, notnull);
 Notification (inclusive, dynamic, univalued, disjoint, strict, notnull);
 Review (inclusive, dynamic, multivalued, disjoint, strict, notnull);
 PcMember (relational, dynamic, multivalued, nodisjoint, strict, notnull);
 Reviewer (relational, dynamic, multivalued, nodisjoint, strict, notnull);
 PcChair (relational, dynamic, univalued, disjoint, strict, notnull);
identification:
      PaperId: (PaperId);
constant_attributes
…
Variable_attributes
…
private events
     var
        ntitle: String;  nauthor: String; ncoauthors: String;  nabstract: String;  ntopics: String;  nstatus: String;
     end_var
     crear ()  new;
     eliminar (paperId) destroy;
shared_events
      SubmitPaper(ntitle, nauthor,ncoauthors,ntopics) with Author NEW;
      AssignPaperPcMember() with PcChair;
Valuation
      [modification_data(ntitle,nauthor,ncoauthors,nabstract,nstatus)] title=ntitle and author=nauthor and
       coauthors = ncoauthors and nabstrac = abstract and topics = ntopics;
      [SubmitPaper(ntitle, nauthor,ncoauthors,ntopics)] title= ntitle and author = nauthor and coauthors =
       ncoauthors and ntopics = topics;
      [AssignPaperPcMember ()] status = “assigned”
end_class
```

Table 10 Specification of the *Paper* Class in OASIS language.

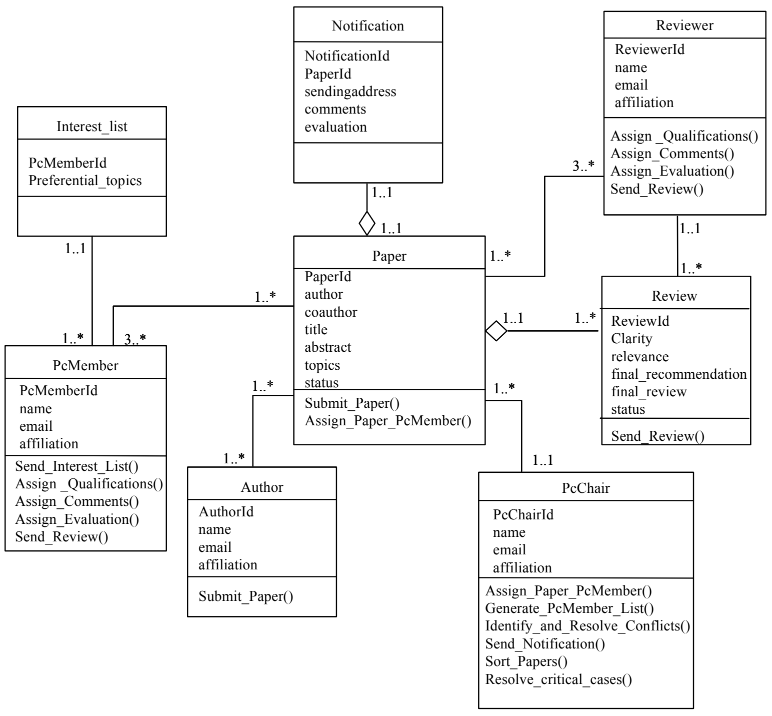

Figure 3 Conceptual Schema of the Conference Review System case study.

Figure 3 shows the graphical representation of the complete conceptual schema obtained from the translation process for the case study. The conceptual schema in OASIS language is the input of the OO-Method Case Tool, which automatically establishes the correspondence between the conceptual primitives of the OASIS conceptual schema and the corresponding elements in a target programming language.

5. Conclusions and further work

This paper presents a method for translating an organizational model into specifications of a conceptual schema for information systems. As a first step in the method, an organizational model which has been previously specified in the i* framework is selected. This Model is then enriched with the inclusion of the system actor. This facilitates the identification of the tasks to be automated using the information system. A method for translating each of the elements of the organizational model into specifications of a conceptual model described in OASIS language is also presented.

We are now working on developing a Case Tool which performs the methodological process described in this paper. This CASE Tool will assure the correspondence between the business model and the information system

References

[Alb03] Manoli Albert, Vicente Pelechano, Joan Fons, Marta Ruiz, Oscar Pastor: "Implementing UML Association, Aggregation, and Composition. A Particular Interpretation Based on a Multidimensional Framework", Proceedings of the CAISE 03, pp 143-158, Austria, 2003.

[And96] Andrade Luis, Amílcar Serdanas, "Banking and Management Information System Automation", proceedings of the 13th world congress IFAC, Volume L, 1996.

[Bub95] Bubenko, J. A., jr and M. Kirikova, "Worlds in Requirements Acquisition and Modelling", in: Information Modelling and Knowledge Bases VI. H.Kangassalo et al. (Eds.), IOS Press, pp. 159– 174, Amsterdam, 1995.

[Cas01] Jaelson Castro, John Mylopoulos, Fernanda M. R. Alencar, Gilberto A. Cysneiros Filho: Integrating Organizational Requirements and Object Oriented Modeling. Proceedings of the RE 2001, pp146-153, Canada, 2001.

[Cas02] Castro J. Kolp M. Mylopoulos J. "Towards Requirements-Driven Information Systems Engineering: The Tropos Project". Information System Journal, Elsevier, Vol 27, pp 365-389, 2002.

[Ces02] Cesare S. Mark Lycett, "Business Modelling with UML, distilling directions for future research", Proceedings of the Information Systems Analysis and Specification (ICEIS 2002), pp. 570-579, Ciudad-Real, Spain, 2002.

[Col00] Colette Rolland, Janis Stirna, Nikos Prekas, Pericles Loucopoulos, Anne Persson, Georges Grosz: Evaluating a Pattern Approach as an Aid for the Development of Organisational Knowledge: An Empirical Study. Proceedings of the CAiSE 00, pp176-191, Sweden, 2000.

[Dar93] Dardenne, A. Van Lamsweerde and S. Fickas, "Goal Directed Requirements Acquisition," Science of Computer Programming, vol. 20, pp. 3-50, North Holland, April 1993.

[Kol03] Manuel Kolp, Paolo Giorgini, John Mylopoulos: Organizational Patterns for Early Requirements Analysis. Proceedings of the CAiSE 03, pp 617-632, Austria, 2003.

[Kou00] Manolis Koubarakis, Dimitris Plexousakis: A Formal Model for Business Process Modeling and Design. Proceedings of the CAiSE 00, pp 142-156, Sweden, 2000.

[Lou95] Loucopoulos Pericles, Evangelioa Kayakli, "Enterprise Modelling and the Teleological Approach to Requirements Engineering", International Journal of Cooperative Information Systems (IJCIS), pp. 45-79 , 1995.

[Pas01] Oscar Pastor, Jaime Gómez, E. Infrán, V. Pelechano, The OO-Method approach for information systems modeling: from object-oriented conceptual modeling to automated prgramming, Information Systems 26, 2001.

[Pas95] Oscar Pastor, Isidro Ramos, OASIS 2.1.1: A Class-Definition Language to Model Information Systems Using an Object-Oriented Approach, 3rd Edition, Servicio de Publicaciones, Technical University of Valencia, Spain, 1995.

[San03] J. Sánchez; O. Pastor; H. Estrada; A. Martínez, Semi Automatic Generation of User Interfaces Prototypes from Early Requirements Models, Perspectives on Requirements Engineering. Editors: J. Leite; J. Doorn, Kluwer Academic Press, to appear in October 2003.

[Sch98] Schwabe Daniel and Gustavo Rossi, "An Object Oriented Approach to Web-Based Application Design", Proceedings of the Theory and Practice of Object Systems, pp 207-225, New York, 1998.

[Yu95] Yu, Eric, Modelling Strategic Relationships for Process Reengineering, Phd Thesis, University of Toronto, (1995).