A pattern language to join early and late requirements[1]

Alicia Martínez[1,2], Oscar Pastor[1], Hugo Estrada[1,3]
[1] *Valencia University of Technology*
Avenida de los Naranjos s/n, Valencia, España
[2] *I.T. Zacatepec, Morelos, Mexico*
[3] *CENIDET Cuernavaca, Mor. Mexico*
{alimartin, opastor, hestrada}@dsic.upv.es

Abstract. At present, the early phase of Requirements Engineering is a new research area in the Software Engineering field. This phase is concerned with the analysis of the organizational context in which a software system will be used. The models used in this phase allow us to describe an organizational environment using actors, goals, business processes and relationships. The late phase of Requirements Engineering, which is focused on representing the expected functionality of the software system, is more developed, so there are multiple techniques and tools to describe the software system that will be developed inside its operational environment. However, although there are methodologies which give separate support to each phase of requirements engineering, the development of methods to derive late requirements from the early requirements in a methodological way has been neglected in recent research works. This is due, in great measure, to the large difference between the abstraction levels of these two specification models. The objective of this paper is to propose a pattern language which allows us to reduce the abstraction level between early requirements and late requirements in a systematic way. This is done in an MDA-based approach.

Keywords: Organizational Model, Early Requirements, Late Requirements, pattern design.

1. Introduction

At present, several research efforts have been made to accurately represent an organizational model (early requirements) [4][12][2][5]. In these works, conceptual primitives represent business goals, organizational actors and dependencies among these actors. There are also several research works focused on the development of requirements models (late requirements) to represent the expected functionality of the information system [17][6][13] [18]. However, the problem of linking business models with requirements models in a methodological way has still not been resolved. One of the principal reasons for this is the different nature of their specifications. In the early requirements phase, the concepts are related to the organizational context,

[1] This work has been partially supported by the MEC project with ref. TIN2004-03534, the Valencia University of Technology, Spain, and the SUPERA project, Mexico.

while, in the late requirements phase, the concepts are related to the software system to be developed. There is thus a significant difference between the abstraction levels of both requirements specifications.
The lack of methods to generate the expected functionality of the software system from the relevant tasks of the organizational model have lead to severe limitations in the usefulness of these works in real software development environments.
The main objective and contribution of this paper is to propose a pattern language which permits us to reduce the abstraction level of a "pure" organizational model so that it is closer to the requirements model. The reduction process generates a new organizational model that is correctly adapted to systematically generate the requirements model. The proposed method complies with the MDA[8][11] approach, implementing the concept of PIM (platform independent model)-to-PIM transformations.
This paper is structured as follow: Section 2 presents the methodological background of the method. Section 3 presents an overview of the proposal as well as a case study. Section 4 shows the pattern language, and Section 5 presents conclusions and future work.

2. Methodological Background

The methods that give theoretical support to this research work are presented in this section. First, we present the Tropos Framework. This Framework is used to represent the early requirements. We then present a brief review of the Software Patterns. The patterns proposed in this work are used to reduce the abstraction level between early requirements and late requirements.

2.1 Organizational Model

In this paper, the Tropos Framework [4] is used to represent organizational contexts. Tropos proposes a software development methodology and a development framework which are both founded on concepts used to model early requirements. One of the key points of the Tropos Framework is the explicit representation of the social model that exists in an organizational context. In this social view of the organization, the actors depend on other actors to satisfy their goals, perform their tasks and obtain resources. The Tropos Framework provides two graphical models to describe these relationships: the Strategic Dependency Model and the Strategic Rationale Model (SR). The SR Model is used to carry out a deeper reasoning of the motives that exist behind each dependency relationship. In this model, the dependencies express intentional relationships that exist among actors in order to fulfill strategic objectives. A dependency describes an agreement between two actors, the *depender* and the *dependee*. The type of the dependency describes the type of agreement: a) Goal dependencies are used to represent the delegation of responsibility to fulfill a goal; b) Task dependencies represent situations where the *dependee* is required to perform a given activity; c) Resource dependencies are used to represent the delivery of resources between actors. The SR Model contains task decomposition links, which allow us to represent the combination of the necessary tasks to achieve a goal. It also contains mean-ends links, which allow us to represent the several alternatives that can be taken to fulfill a task or goal.

2.2 Pattern Languages

A pattern is a description of a common solution to a recurrent problem, which can be applied to a specific context [9]. The are several types of patterns such as architectural patterns [3] that show the high level architectures of a software system, design patterns [15][14] that are focused on the programming aspects, or patterns that are focused on project management [1].
In this paper, we propose a set of organizational patterns to allow us to reduce the abstraction level of an organizational model, bringing it closer to the requirements model of a software system. This is done by inserting the software system actor into the original organizational model and redirecting the dependencies of the original organizational actors towards this new actor. In this way, the proposed patterns allow us to analyze the organization elements, such as tasks, resources and goals, with the purpose of inserting a new organizational actor, which represents the software system to be developed. This new organizational actor should map the objectives and dependencies that exist among the organizational actors.

3. Proposal Overview

In this section, we present an overview of the proposed method to reduce the abstraction level between early requirements and late requirements. This specific research work is part of a project called “A methodological approach to generate requirements models from organizational models”. The complete method is composed of three phases (see Figure 1). However, for reason of brevity, in this paper we only describe the second phase, “the generation of the new organizational model”. The initial PIM of the proposed method is an organizational model represented in the Tropos Framework. This organizational model has been created using a goal-based approach [7]. In the second phase, we use transformational rules to generate a new PIM. In this paper, the transformation rules are defined by a Pattern Language called “FELRE” (From Early Requirements to Late Requirements). The new PIM represents an organizational model which integrates the software system actor (SSA). This phase allows us to reduce the abstraction level between a “pure” organizational model and a requirements model of the software system. Finally, in the third phase, the organizational model is systematically transformed into a new PIM that represents the requirements model of the information system. This third phase was analyzed in a previous initial version [16].

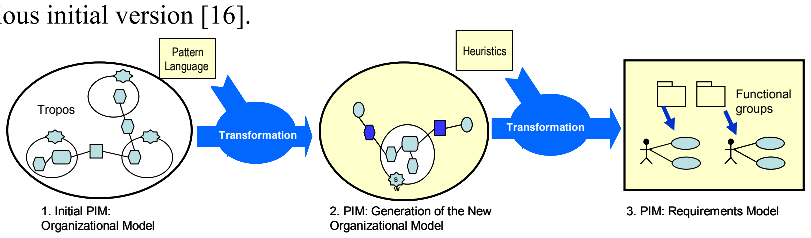

Pattern Language
Tropos
Transformation
Heuristics
Transformation
Functional groups
1. Initial PIM: Organizational Model
2. PIM: Generation of the New Organizational Model
3. PIM: Requirements Model

Figure 1. Schematic Representation of the methodological approach

3.1 The Case Study

In order to illustrate our approach, we analyze the “Golf Tournament Management” case study. The objective of this case study is to analyze the business processes of a company that manages golf tournaments.

The golf tournaments are validated by the Golf Federation, which ranks golfers in the golf championship. One of the main concerns of the company in question is to provide partial results for each game. To do this, there are controllers that register the results of the golfers for specific holes.

Figure 2 shows a fragment of the organizational model for this case study. This model represents the actors who perform tasks in the business: the Organization (the company), the Golfers, and the Controllers and the Golf Federation. There are several dependencies among the actors: the Organization depends on Golfers to obtain the registration information for each player. The Golfers depend on the Organization to obtain a card with the game information. The Organization depends on the Controllers to get the partial results of each game. The Organization also depends on the Federation to validate the results of the games. The shaded elements in Figure 2 will be used in section 4 to illustrate the translation patterns proposed in this paper.

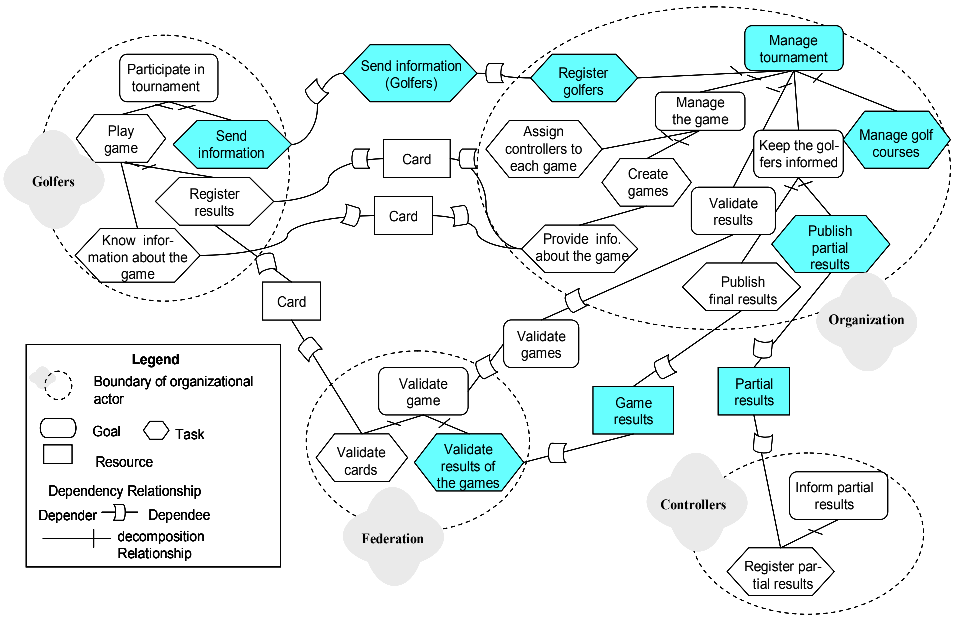


Figure 2. SR Model of the “Golf Tournament Management” case study

4. The FELRE Pattern Language

At present, pattern languages have been widely used as tools to define methodological frameworks that guide the modeling and design processes.

Following this line of research, we have developed a pattern language called “FELRE” (From Early Requirements to Late Requirements).

Table 1 shows a brief description of the FELRE patterns, this pattern language allows us to reduce the abstraction level that exists between early requirements and late requirements. A new intermediate model between the organizational and requirements is created to do this. The new organizational model (SS-BM) generated by the application of the pattern language will integrate the software system (SSA) as an actor of the model.

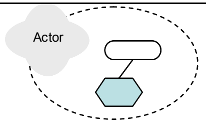
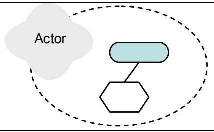
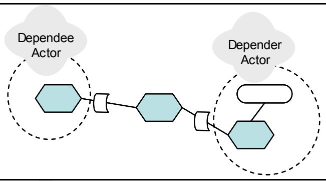
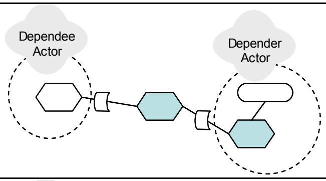
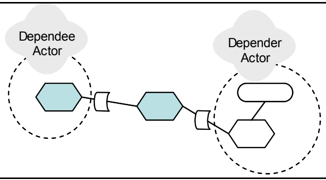
Name of pattern	**Use**	🟦 Task or goal to be automated
The *Final Task without dependencies* Automation Pattern	To be used when a *final task without dependencies* needs to be automated. The *final tasks without dependencies* are those final tasks that do not require the intervention of another actor.	
The *General Task or General* Automation Pattern	To be used when a *General Task or General Goal* needs to be automated.	
The *Depender-Dependee Actor Tasks* Automation Pattern	To be used when the tasks to be automated are both the *depender* actor task and the *dependee* actor task.	
The *Depender Actor Task* Automation Pattern	To be used when the *depender* actor task must be automated.	
The *Dependee Actor Task* Automation Pattern.	To be used when the *depender* actor task must be automated.	

Table 1 A brief description of the FELRE patterns

This new organizational actor represents the information system to be constructed, and in this context, this actor contains all the organizational tasks selected to be automated using a software system.

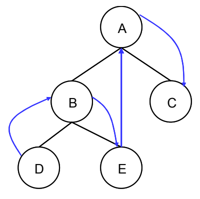
To do this, the original dependencies, goals, resources and tasks of the organizational actor need to be redirected towards the software system actor. In this way, the goals and tasks of the business are not modified; only the actor responsible for satisfying them is modified. Besides the elements that are redirected towards the SSA, this new organizational actor also contains the new dependencies that have been created during

the insertion of the SSA. They allow the SSA to obtain resources from the organizational actor as well as the execution of tasks using the information system.
The inclusion of the software system as an actor in the organizational model allows us to have a high-level description of the task that must be supported by the information system. This high-level description permits focus only on the relevant aspects to be automated, thereby reducing the complexity of the analysis task. Therefore, this model is correctly adapted to start the process of finding the requirements for the information system.
The generation process of the SSA could be done in a systematic way using the patterns presented in this paper. Without the patterns, the process of insertion of the software system actor could be very costly in time and in effort.

4.1 Implementing the FELRE pattern language

FELRE is composed of five patterns which allow us to systematically guide the analyst to obtain the new SS-BM. The process to implement the pattern language is as follows:

- **Step 1.** Identify the relevant tasks to be automated. The relevant elements of the organizational model are shown in Figure 2 (shaded areas). These shaded areas are used to indicate each one of the proposed patterns.
- **Step 2.** Place the SSA into the SS-BM. Include the actors that have some tasks, goals or dependencies relationship to be automated. In the case study analyzed, the actors with these characteristics are: the Golfers, the Organization, the Federation, and the Controllers.
- **Step 3.** Transfer the tasks or goals to be automated to the SSA. To perform this step, the following substeps must be carried out:
  - **Step 3.1** Analyze the internal tasks of the actors. An actor can be composed of several goals and tasks, which, in turn, can be subdivided into goals or tasks. This subdivision leads to a tree structure. An infix traversing must be performed in the internal task trees of each one of the organizational actors of the organizational model (see Figure 1). An example of this infix traversing is shown in Figure 3. In our case study "Golf Tournaments Management", for example, if the infix traversing of the actor Organization in Figure 2 is performed, the first task analyzed would be *Register Golfers*. The next goal to be analyzed would be *Manage Tournament*, followed by the task *Assign Controllers to each game*. This process continues until all the internal tasks of the Organization actor have been traversed.
  - **Step 3.2** Analyze each one of the internal elements obtained in the infix traversing. If one of these elements has a dependency relationship associated to it, all the elements of this dependency (*depender*, *dependee* and *dependum*) must be analyzed. This analysis allows us to identify the automation patterns that exist in the organizational model. The automation



Infix Traversing: D B E A C

Figure 3. Example of infix traversing

patterns proposed in this paper are the following: The *Final Task without dependencies* Automation Pattern*,* the *General Task or General* Automation Pattern*,* the *Depender-Dependee Actor Tasks* Automation Pattern*,* the *Depender Actor Task* Automation Pattern*,* and the *Dependee Actor Task* Automation Pattern. In our case study, to satisfy the task *Register Golfers*, the Organization *(depender* actor) depends on the Golfers (*dependee* actor) to obtain their personal data. This is represented with the task dependency "send information" between the Organization and the Golfers. In this example, the registration process as well as the process of sending this information have been selected for automation. For this reason, the Pattern identified in this example is the *Depender-Dependee Actor Tasks* Automation Pattern.

- **Step 3.3** Use the appropriate pattern to transfer the tasks or goals to be automated to the SSA. Once this pattern has been identified, the steps indicated in the pattern description must be followed.

4.2 Automation pattern descriptions

In this section, each one of the five automation patterns of FELRE pattern are explained.

1) **The *Final Task without dependencies* Automation Pattern**

Context: The organizations are composed of different types of tasks. One of these types is the *final task*. This sort of task is not decomposed into other goals or into other tasks. The *final tasks without dependencies* are those final tasks that do not require the intervention of another actor. This *final task* only needs the actor that contains it. For this reason, this task does not have associated any dependency with any other actor associated to it.

Problem: When analysts start analyzing the company to determine the expected functionality of the software system, they must analyze the organizational tasks that will be automated as well as the effects of this automation on the organizational actors and on the dependencies that already exist among these actors. If the task to be automated is a *final task*, which does not have any dependencies with any other actors, then other situations should be analyzed. Three forces are associated to this problem:

- The *final task* to be automated needs the intervention of the actor that performed it previously.
- The *final task* to be automated needs the intervention of the actor that performed it previously as well as the intervention of other organizational actors.
- The *final task* to be automated doesn't need the intervention of any organizational actor.

Solution: When a *final task without dependencies* needs to be automated, the first step is to transfer this task to the SSA of the SS-BM. The next step is to determine if the original owner of the task must perform it, or if the SSA could perform the task itself.

If the intervention of another actor is required to perform the task, either to execute tasks or to obtain resources, new dependency relationships should be created among these actors and the SSA. These new dependencies could be:

- Task Dependencies, which indicate the introduction of information to the software system from the organizational actor. In this case, it will be necessary to identify the entities modified in each task and to place them as parameters in the dependency tasks.
- Resource Dependencies, which indicate the delivery of resources to/from the organizational actor.

Example: In our case study, this pattern was found by performing the infix traversing of the Organization actor (Figure 2). The task *Manage Golf Courses* of this actor complied with the characteristics of the *Final Task without dependencies* Automation Pattern. When the pattern was applied, this task was transferred to the SSA and a new dependency task between the Organization actor and the SSA was generated. This dependency allowed us to indicate that the Organization would provide the information about the golf courses to the SSA. For this reason, *Golf Courses* was set as a parameter of the task dependency. The results of the application of this pattern are shown in Figure 4.

2) The *General Goal or General Task* Automation Pattern

Context: The elements of the SR Model of the Tropos Framework are: the task decomposition links and the mean-ends links. The task decomposition links indicate that a main task (named *General Task* or Parent Node) needs to be executed by performing each one of the subtasks (named Child Nodes). The mean-ends links indicate that the main task needs to be executed by performing any one of the different alternatives indicated by its subtasks.

Problem: When a *General Task or General Goal* is analyzed for automation, its associated children nodes must also be analyzed. This is due to the fact that the transfer of this task or goal to the SSA will depend on the decision to automate any of the child nodes. Two forces are associated with this problem:

- At least one children node needs to be automated.
- The General Task or General Goal has a dependency with another organizational actor.

Solution: To determine if the General Task/Goal should be transferred to the SSA, the child nodes must be analyzed. If at least one of them needs to be automated, then it will probably be necessary to transfer the General Task/Goal to the SSA. To do this, the following two steps must be applied: In the first step, the General Task/Goal is transferred to the SSA of the SS-BM. In the second step, all the tasks and goals that were previously transferred to the SSA must be associated to their corresponding General Task/Goal.

If the General Task/Goal has a dependency with another actor, an appropriate pattern to perform the transfer must be selected. The potential patterns to be applied are: The *Depender-Dependee Actor Tasks* Automation Pattern, the *Depender Actor Task* Automation Pattern, and the *Dependee Actor Task* Automation Pattern.

Example: In our case study, this pattern was found by performing the infix traversing of the Organization actor (Figure 2). The goal *Golf Tournament Management* of this actor complied with the characteristics of the *General Task or General Goal* Automation Pattern. In this example, the tasks *Register Golfers*, *Publish partial results* and *Manage Golf Courses*; which are subtasks of the Golf Tournament Management Goal, had already been transferred to the SSA. For this reason, this goal was also transferred to the SSA. The results of the application of this pattern are shown in Figure 4.

3) The *Depender-Dependee Actor Tasks* Automation Pattern

Context: In the Tropos framework, the organizational actors are related to each other through dependencies. Each dependency is composed by [19]: The dependency direction, which identifies who the *depender* and the *dependee* are, and the dependency type (resource, goal or task).

Problem: One of the main problems of inserting the SSA into the organizational model is to transfer the tasks to be automated to this new organizational actor. If the task that is being analyzed has a dependency relationship associated to it, all the elements of that dependency (*depender*, *dependee* and *dependum*) must be analyzed. When the tasks to be automated are both the *depender* actor task and the *dependee* actor task, the *dependum* object is the guide for the steps to be followed. Two forces are associated with this problem:

- **The dependum is a resource***:* Generally, in a dependency relationship, one of the involved actors has the internal task of generating the resource (*dependee* actor), and the other actor has the task of obtaining the resource (*depender* actor). In this case, it must be determined if one or both of these tasks are to be automated.
- **The dependum is a task or goal***:* Generally, in a task dependency, the *depender* actor depends on the *dependee* actor to carry out an internal task to satisfy the dependency task/goal. In this case, it is necessary to determine if the automation of the *dependee* actor task will also allow the *depender* actor to perform the task or to obtain the results of the task performed by the *dependee*.

Solution: When both the *depender* and the *dependee* task must be automated, the *dependum* object must be analyzed before following the instructions for each case.

- **The *dependum* is a resource:** the tasks or goals linked to the resource sould be analyzed by completing the following steps:

  Step 1. Depending on the automation, either the resource generation task (of the *depedeee* actor) or the resource reception task (of the *depender* actor) will be transferred to the SSA.

  Step 2. The original resource dependency between the organizational actors is redefined in a resource dependency between a software actor and the SSA.

  Step 3. When the redirection of the resource dependency has been done, the original resource dependency between the organizational actors disappears. Therefore, one of the actors of the original resource dependency (depending on whether the reception task or the generation task has been selected) loses its dependency with the other organizational actor. In this case, it is necessary to create an *interaction relationship* between this first actor and the SSA. An interaction relationship is represented graphically by an arrow between the organizational actor and the SSA. This represents the fact that an actor can activate a task of the SSA.

Step 4. The dependency relationships generated during the automation process are labelled to indicate that these dependency relationships are associated to each other.

Step 5. The internal tasks of the organizational actors that have been transferred to the SSA are labelled in order to indicate that they have already been analyzed.

- **The *dependum* is a task:** the tasks or goals linked to the task should be analyzed by completing the following steps:

  Step 1. The *depender* actor task is transferred to the SSA.

  Step 2. A new dependency relationship or interaction relationship is created between the *depender* of the original dependency and the SSA. The *depender* of the dependency will be the SSA. This new dependency relationship represents the introduction or the reception of information in the SSA by the organizational actor.

  Step 3. The *dependee* actor task (of the original dependency relationship) is transferred to the SSA as a task decomposition that is linked to the task transferred in Step 1.

  Step 4. Either a task/goal dependency or an interaction relationship between the *dependee* actor of the original dependency and the SSA is created. The *depender* actor of this new dependency will be the SSA. This new dependency relationship represents the introduction or reception of information in the SSA by the organizational actor.

  Step 5. The dependency relationships generated during the automation process of resource generation or resource reception are labelled to indicate that these dependency relationships are associated to each other.

  Step 6. The internal tasks of the organizational actors that have been transferred to the SSA are labelled to indicate that they have already been analyzed.

Example: In our case study, this pattern was found by performing the infix traversing of the Organization actor (Figure 2). The task *Register Golfers* of this actor complied with the characteristics of the *Depender-Dependee Actor tasks* Automation Pattern, because this task was linked to the task dependency *send information (Golfers)* which also had to be automated. Once the pattern was applied, a task decomposition SSA was created (the parent node was the *Register Golfers* task, and the child node was the *obtaining information*). Both a dependency relationship and an interaction relationship were also created. . The dependencies that were modified or generated in this example are labelled with the number 1. These results are shown in Figure 4.

4) **The *Depender Actor Task* Automation Pattern**

Context: In a dependency relationship, the *depender* actor depends on the *dependee* actor to achieve its goals, to perform its tasks or to deliver resources [19].

Problem: As mentioned before, one of the main problems in the insertion of the SSA is the determination of the tasks to be automated. If the task analyzed has a dependency relationship associated to it, all the elements of that dependency (*depender*, *dependee* and *dependum*) must be analyzed. When the task to be automated is the *depender* actor task, the *dependum* object is the guide for the steps to be followed. Two forces are associated with this problem:

- The task to be automated is linked to a resource dependency.
- The task to be automated is linked to another task dependency.

Solution: When the *depender* actor task must be automated, the *dependum* object must be analyzed before following the instructions for each case.

- **The dependum is a Resource**. The following steps should be carried out.

  Step 1. The *depender* actor task is transferred to the SSA from the SS-BM.

  Step 2. The next step consists of determining if the original owner of the task (the *depender* actor of the original dependency) must perform it, or if the SSA could perform the task itself. If the intervention of the actor is required, a new task or resource dependency should be created between this actor and the SSA. The *depender* actor of this new dependency relationship will be the SSA.

  Step 3. The resource dependency is placed between the organizational actors of the SS-BM.

- **The dependum is a Task**. The following steps should be carried out.

  Step 1. The *depender* actor task is transfered to the SSA from the SS-BM.

  Step 2. The next step consists of determining if the original owner of the task (the *depender* actor of the original dependency) must perform it, or if the SSA could perform the task itself. If the intervention of the actor is required, a new task or resource dependency should be created between this actor and the SSA. The actor *depender* of this new dependency relationship will be the SSA.

  Step 3. The task dependency is redirected by placing the SSA as the *depender* actor and placing the actor that was the *dependee* of the original dependency as the *dependee*.

Example: In our case study, this pattern was found by performing the infix traversing of the Organization actor (Figure 2). The task *Publishing Partial Results* of this actor complied with the characteristics of the *Depender Actor Task* Automation Pattern. In this case, only the *depender* actor task was automated. We applied the steps indicated for *the dependum is a resource*. The results of the application of the pattern are shown in Figure 4. The dependencies that were modified or generated in this example are labelled with the number 2.

5) **The *Dependee Actor Task* Automation Pattern**

Context: The *dependee* actor is the actor on which another actor depends to satisfy the dependency relationship [19]. It is the responsible for satisfying the dependency.

Problem: Generally, the internal tasks of the organizational actors are linked to dependency relationships with other organizational actors. In this case, all the elements of that dependency (*depender*, *dependee* and *dependum*) must be analyzed. When the task to be automated is the *depender* actor task, the *dependum* object is the guide for the steps to be followed. Two forces are associated with this problem:

- The task to be automated is linked to a resource dependency.
- The task to be automated is linked to another task dependency.

Solution: When the *dependee* actor task must be automated, the *dependum* object must be analyzed before following the instructions for each case.

- **The *dependum* is a Resource.** The following steps should be carried out.

  Step 1. The *dependee* actor task is transferred to the SSA.

  Step 2. In this step, it is necessary to determine whether the *depender* actor could use the system to obtain the resource, or if the resource will be sent by the *dependee* actor.

  Step 2.1 If the *depender* actor does not have access to the system to obtain the resource, the resource dependency remains the same, and another resource dependency must be created between the SSA and the *depender* of the original resource dependency. The *depender* actor of this new dependency will be the SSA.

  Step 2.2 If the *depender* actor does have access to the system, a resource dependency between this actor and the SSA is created. An interaction relationship between the *dependee* actor of the original dependency and the SSA is also created.

- **The dependum is a Task.** The following steps should be carried out.

  Step 1. The *dependee* actor task is transferred to the SSA.

  Step 2. The dependency task is redirected between the *depender* actor of the original dependency and the SSA.

  Step 3. The next step consists of determining if the original owner of the task (the *dependee* actor of the original dependency) must perform it, or if the SSA could perform the task itself. If the intervention of the actor is required, a new task or resource dependency should be created between this actor and the SSA. The *depender* actor of this new dependency relationship will be the SSA.

Example: In our case study, this pattern was found by performing the infix traversing of the Federation actor (Figure 2). The task *Validate results of the games* of this actor complied with the characteristics of the *Dependee Actor task* Automation Pattern. In this case, only the *dependee* actor task was automated. We applied the steps indicated for *the dependum is a resource*. The results of the application of the pattern are shown in Figure 4. The dependencies that were modified or generated in this example are labelled with the number 3.

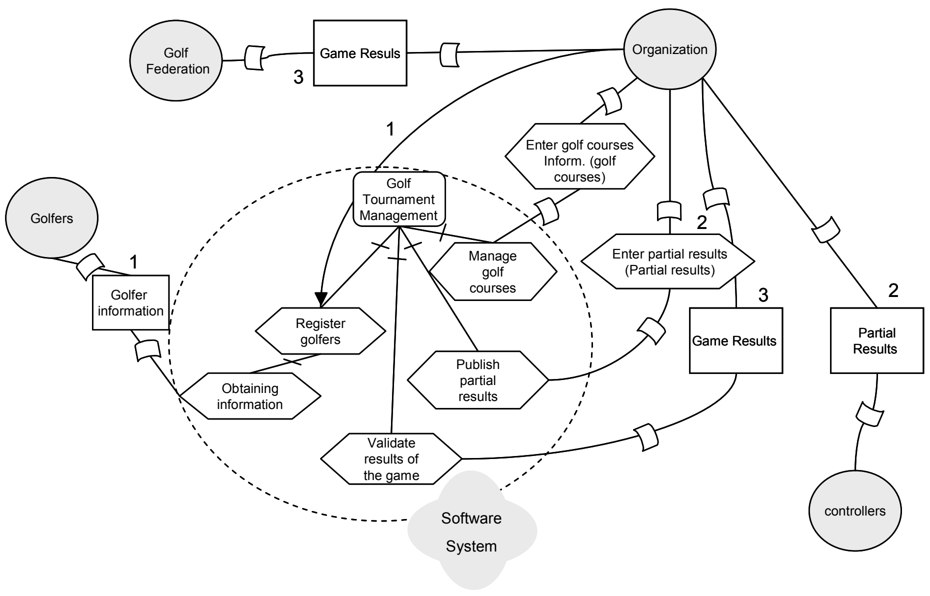


Figure 4 SS-BM generated by the application of the pattern language

5. Conclusions and future work

One of the main problems of current research works on organizational modelling is the lack of a methodological approach to map of the elements of an organizational model into the elements of a requirements model for a software system. Because of this lack, efforts in the organizational modelling phase have not yet provided practical application for software development environments.

In this work, we have proposed a pattern language which allows us to reduce the abstraction level of a "pure" organizational model so that it is closer to the requirements model. This process has been achieved by inserting the software system as an actor into the organizational model and redirecting the relevant tasks, goals and dependencies of the organizational actors to this new actor. In this way, there is a pattern for each situation that arises in the redirection of tasks or goals to the new organizational model. The new organizational model generated from the application of FELRE allows us to have a high-level description of the task that must be supported by the information system. This high-level description permits us to focus only on the relevant aspects to be automated, thereby reducing the complexity of the analysis task. The generated organizational model is therefore an intermediate model between the organizational model and the requirements model. The proposed method complies with the MDA approach because it implements the concept of PIM-to-PIM transformations.

We are currently developing a method to automatically obtain the requirements model for the RETO Tool [10] from the new organizational model presented in this paper.

6. References

[1] Beedle Michael A. COOherentBPR –A pattern language to built agile organizations, Plop-97 Conference, 1997.

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

[3] Buschmann, R. Meunier, H. Rohnert, P. Sommerland and M. Stal, Pattern - Oriented software Architecture: A system of Patterns. John Wiley & Sons, 1998.

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

[5] Cesare S. Mark Lycett, Business Modelling with UML, distilling directions for future research, Proceedings of the Information Systems Analysis and Specification Cd. Real, 2002, pp. 570-579.

[6] Cockburn Alistair, Writing Effective Use Cases, Addison-Wesley, 2001.

[7] Estrada Hugo, Martínez Alicia, Pastor Oscar, Goal-based business modeling oriented towards late requirements generation, 22nd International Conference on Conceptual Modeling ER2003, ISBN 3-540-20-299-4, Chicago Illinois, 2003, pp. 277-290.

[8] Frankel S. David, Model driven Architecture, applying MDA to enterprise computing, Wiley publishing, Inc. USA, 2003.

[9] Gamma] E., R. Helm, R. Johnson, and J. Vlissides. Design Patterns, Addison-Wesley, 1995.

[10] Insfrán Pelozo Emilio, A Requirements Engineering Approach for Object-Oriented Conceptual Modeling, PhD Thesis, Valencia University of Technical 2003.

[11] Kleppe Annere, Warmer Jos, Bast Wim, MDA EXPLAINED, the model driven architecture: practice and promise, Addison Wesley, ISBN: 0-321-19442-X, Boston, April 2003.

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

[13] Kulak Daryl Eamonn Guiney, Use Cases requirements in context, Addison-Wesley, ISBN 0-201-65767-8, 2000.

[14] Martin Robert, Dirk Riehle, Frank Buschmann, and John Vlissides (eds). Patterns Languages of Program Design 3, Addison-Wesley, 1998.

[15] Meszaros G. and J. Doble, A Pattern Language for Pattern Writing, pages 529-574. Pattern Languages of Program Design 3, Addison Wesley, Robert Martin, D. Riehle and F. Buschmann ISBN 0-201-310112, USA, 1998.

[16] Pastor Oscar, Alicia Martínez R., Hugo Estrada, Generación de Especificaciones de Requisitos de Software a partir de Modelos de Negocios, Informatics Technology Management Núm. 1, Vol. 1. 2002. ISSN 657-82364 Págs. 53-65.

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

[18] Rolland R., Souveyet, C., Plihon, V., *Method Enhancement with Scenario Based Techniques*, Proceedings CAISE 99, 11th Conference on Advanced Information System 1999.

[19] Yu Eric, Modelling Strategic Relationships for Process Reengineering, PhD Thesis, University of Toronto, Toronto, 1995.