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, Spain
[2] I.T. Zacatepec, Morelos, Mexico
[3] CENIDET Cuernavaca, Mor. Mexico

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) [2][3][6][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 [14][7][11][15]. However, the problem of linking Early Requirements (business models) with Late Requirements (requirements of the information system) in a methodological way has still not been resolved. One of the main reasons for this is the different nature of their specifications. In the early requirements phase, the concepts are related to the organizational context, 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 plans 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][10] 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 and late requirements of this proposal. 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 [2] 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 [16]. One of the key points of the Tropos Framework is the capacity for representing both the organizational context where the software system will operate, and also, the interactions between the software system and the human agents. In the early requirements phase of Tropos, the modeling activity is focused on describing the application domain and the intentions of the social actors that want to achieve their goals. In this phase, the social actors are identified and the network of dependencies between them is specified. In this way, it is possible represent the cases where an actor depends on other actors for goal to be achieved, for plans to be performed, and resources to be furnished. Some key concepts in Tropos are: a) Actor, that represents a physical, social or software agent as well as a role or position, b) Goal, which represent actor's strategic interest, c) Plan, which represents, at an abstract level, the way of doing something, A plan can contain AND/OR decomposition of a root plan into sub-plans, d) Resource, which represents a physical or an informational entity. e) Dependency, which indicates that one actor depends, for some reason, on the other in order to attain some goal,

[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.

execute some plan, or deliver a resource. It is composed by: *Depender*: the actor who is dependent on another actor, *Dependee*: the actor on whom another actor depends, and finally, *Dependum*: the task, goal, resource or softgoal on which the relationship is focused.

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]. There are several types of patterns such as architectural patterns [4] that show the high level architectures of a software system, design patterns [13] 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 plans, 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 conceptual schemas from organizational models".

The complete method is composed of several phases which allow us to use the organizational context for generating, in a systematic way, the specification of the conceptual schema of an information system. This conceptual schema is the input for the OO-Method Case Tool, which produces the source code for the information system in a target language.

However, for reason of brevity, in this paper we only describe the process for joining the early requirement phase with the late requirement phase. In the MDA Modeling context, the initial model (Tropos´s Goal Diagram) could be considered as the first PIM model of the process. Later, we use transformational rules to generate a new PIM Model from the Goal Diagram. The new PIM represents an organizational model which integrates the software system actor (SSA). In this paper, the transformation rules are defined by a Pattern Language called "FELRE" (From Early Requirements to Late Requirements). Finally, the organizational model is systematically transformed into a new PIM that represents the conceptual model of the information system. This last phase was analyzed in a previous initial version [12].

3.1 The Case Study

In order to illustrate our approach, we analyze the "Golf Tournaments Management (GTM)" 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 1 shows a fragment of the organizational model for this case study.

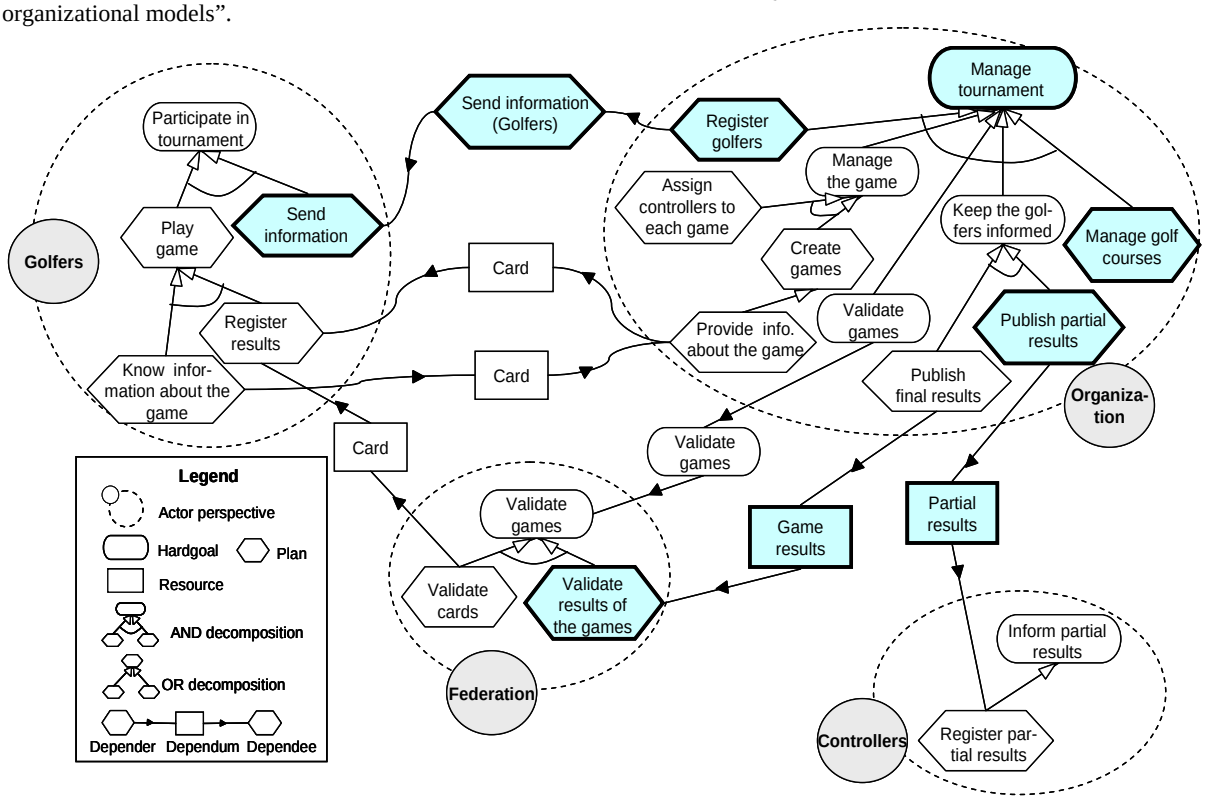


Figure 1. A fragment of the Goal Diagram for the GTM case study

This model represents the actors who perform plans 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 1 will be used in section 4 to illustrate the translation patterns proposed in this paper

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 research line, we have developed a pattern language called "FELRE" (From Early Requirements to Late Requirements), which allows us to reduce the gap that exists between early requirements and late requirements. As result of the application of the patterns, an intermediate model is created between the organizational model and the software requirements. The new organizational model called SS-BM (Software System-Business Model) will integrate the Software System Actor (SSA) as an actor of the organizational model. Table 1 shows a brief description of the FELRE pattern language.
The SS-BM represents the information system to be constructed, and in this context, this actor contains all the organizational plans selected to be automated using a software system.
To do this, the original dependencies, goals, resources and plans of the organizational actor need to be redirected towards the SSA. In this way, the goals and plans 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 plans 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 plan 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 plan. 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 systematically guide the analyst to obtain the new SS-BM. The proposed patterns are used as transformational rules to transform the initial PIM Model (Tropos´s Goal Diagram) into a new PIM Model (Organizational Model that includes the Software System Actor). The process to implement the pattern language is as follows:
Step 1. Identify the relevant plans to be automated. The relevant elements of the organizational model are shown in Figure 1 (elements with thick border). 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 plans, 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.

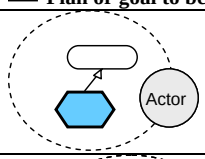
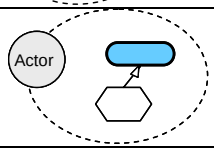
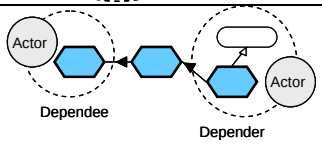
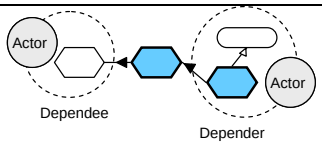
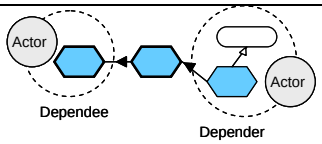
Pattern Name	Use	Plan or goal to be automated
The *Final Plan without dependencies* Automation Pattern	To be used when a *final plan without dependencies* needs to be automated. The *final plans without dependencies* are those final plans that do not require the intervention of another actor.	
The *General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

Table 1 A short description of the FELRE patterns

Step 3. Transfer the plans 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 plans of the actors. An actor can be composed of several goals and plans, which, in turn, can be subdivided into goals or plans. This subdivision leads to a tree structure. An infix traversing must be performed in the internal plans trees of each one of the organizational actors of the organizational model.

  In our case study GTM, for example, if the infix traversing of the actor Organization in Figure 1 is performed, the first plan analyzed would be *Register Golfers*. The next goal to be analyzed would be *Manage Tournament*, followed by the plans *Assign Controllers to each game*. This process continues until all the internal plans 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. In our case study, to satisfy the plan *Register Golfers*, the Organization *(depender* actor) depends on the Golfers (*dependee* actor) to obtain their personal data. This is represented with the plan dependency *send information* between the Organization and the Golfers. In this example, the registration processes 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 Plans* Automation Pattern.
- **Step 3.3** Use the appropriate pattern to transfer the plans 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 Plan without Dependencies Automation Pattern

Context: The organizations are composed of different types of plans. One of these types is the *final plan*. This sort of plan is not decomposed into other goals or into other plans. The *final plans without dependencies* are those final plans that do not require the intervention of another actor. This *final plan* only needs the actor that contains it. For this reason, this plan 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 plans 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 plan to be automated is a *final plan*, 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 plan* to be automated needs the intervention of the actor that performed it previously.
- The *final plan* to be automated needs the intervention of the actor that performed it previously as well as the intervention of other organizational actors.
- The *final plan* to be automated doesn't need the intervention of any organizational actor.

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

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

- ***Plan 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 plan and to place them as parameters in the dependency plans.
- ***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 1). The plan *Manage Golf Courses* of this actor complied with the characteristics of the *Final Plan without Dependencies* Automation Pattern. When the pattern was applied, this plan was transferred to the SSA and a new plan dependency 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, the *Golf Courses* element was set as a parameter of the plan dependency. The results of the application of this pattern are shown in Figure 2.

2) The General Goal or General Plan Automation Pattern

Context: One of the elements of the Goal Model of Tropos is the AND/OR decomposition link, which provides AND/OR decompositions of a root plan into sub-plans. These elements indicate that a main plan (named *General Plan* or *Parent Node*) needs to be executed by performing each one of the sub-plans (named *Child Nodes*).

Problem: When a *General Plan 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 plan 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 Plan or General Goal has a dependency with another organizational actor.

Solution: To determine if the *General Plan*/*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 Plan/Goal* to the SSA. To do this, the following two steps must be applied: In the first step, the *General Plan/Goal* is transferred to the SSA of the SS-BM. In the second step, all the plans and goals that were previously transferred to the SSA must be associated to their corresponding General Plan/Goal.

If the *General Plan/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 Plans Automation Pattern or the *Depender Actor Plan* Automation Pattern or the *Dependee Actor Plan* Automation Pattern.

Example: In our case study, this pattern was found by performing the infix traversing of the Organization actor (Figure 1). The goal *Golf Tournament Management* of this actor complied with the characteristics of the *General Plan or General Goal* Automation Pattern. In this example, the plans *Register Golfers*, *Publish partial results* and *Manage Golf Courses*; which are sub-plans 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 2.

3) The Depender-Dependee Actor Plans Automation Pattern

Context: In the Tropos framework, the organizational actors are related to each other through dependencies. Each dependency is composed by [2]: The dependency direction, which identifies who the *depender* and the *dependee* are, and the dependency type (resource, goal or plan).

Problem: One of the main problems of inserting the SSA into the organizational model is to transfer the plans to be automated to this new organizational actor. If the plan 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 plans to be automated are both the *depender* actor plan and the *dependee* actor plan, the *dependum* object is the guide for the steps to be followed. Two forces are associated with this problem:

- The plans (*depender*/*dependee*) to be automated are linked to a resource dependency (*dependum*).
- The plans (*depender*/*dependee*) to be automated are linked to a plan dependency (*dependum*).

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

- ***The dependum is a resource:*** the plans or goals linked to the resource should be analyzed by completing the following steps:
  - **Step 1.** Depending on the automation, either the resource generation plan (of the *depedeee* actor) or the resource reception plan (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 organizational 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 plan or the generation plan has been selected) loses its dependency with the other organizational actor. In this case, it is necessary to create a new plan or resource dependency between this first actor and the SSA.
  - **Step 4.** The dependency relationships generated during the automation process are labeled to indicate that these dependency relationships are associated between them.
- **The *dependum* is a plan:** the plans or goals linked to the plan should be analyzed by completing the following steps:
  - **Step 1.** The *depender* actor plan is transferred to the SSA.
  - **Step 2.** A new dependency 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 plan (of the original dependency relationship) is transferred to the SSA as a plan decomposition that is linked to the plan transferred in Step 1.
  - **Step 4.** Either a plan/goal dependency 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 labeled to indicate that these dependency relationships are associated to each other.

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

4) The *Depender Actor Plan* Automation Pattern

Context: In a dependency relationship, the *depender* actor depends on the *dependee* actor to achieve its goals, to perform its plans or to deliver resources [2].

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

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

Solution: When the *depender* actor plan 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 plan is transferred to the SSA from the SS-BM.
  - **Step 2.** The next step consists of determining if the original owner of the plan (the *depender* actor of the original dependency) must perform it, or if the

SSA could perform the plan itself. If the intervention of the actor is required, a new plan 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 remains the same between the organizational actors.

- ***The dependum is a Plan.*** The following steps should be carried out.
  - **Step 1.** The *depender* actor plan is transferred to the SSA from the SS-BM.
  - **Step 2.** The next step consists of determining if the original owner of the plan (the *depender* actor of the original dependency) must perform it, or if the SSA could perform the plan itself. If the intervention of the actor is required, a new plan 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 plan 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 1). The plan *Publishing Partial Results* of this actor complied with the characteristics of the *Depender Actor Plan* Automation Pattern. In this case, only the *depender* actor plan 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 2. The dependencies that were modified or generated in this example are labeled with the number 2.

5) The *Dependee Actor Plan* Automation Pattern

Context: The *dependee* actor is the actor on which another actor depends to satisfy the dependency relationship [2]. It is the responsible for satisfying the dependency.

Problem: Generally, the internal plans 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 plan to be automated is the *depender* actor plan, the *dependum* object is the guide for the steps to be followed. Two forces are associated with this problem:

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

Solution: When the *dependee* actor plan 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 plan 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 *dependee* 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, then the original resource dependency is redefined in a resource dependency between the *depender* actor (of the original dependency) and the SSA. A dependency relationship between the *depender* actor of the original dependency and the SSA is also created.

- ***The dependum is a Plan.*** The following steps should be carried out.
  - **Step 1.** The *dependee* actor plan is transferred to the SSA.
  - **Step 2.** The dependency plan 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 plan (the *dependee* actor of the original dependency) must perform it, or if the SSA could perform the plan itself. If the intervention of the actor is required, a new plan 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 1). The plan Validate results of the games of this actor complied with the characteristics of the *Dependee* Actor Plan Automation Pattern. In this case, only the *dependee* actor plan 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 2. The dependencies that were modified or generated in this example are labeled with the number 3.

5. CONCLUSIONS AND FUTURE WORK

One of the main problems of current research works on organizational modeling 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 modeling 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 plans, 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 plans 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 plan 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 plan. 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 conceptual model for the OO-Method Case Tool from the new organizational model presented in this paper.

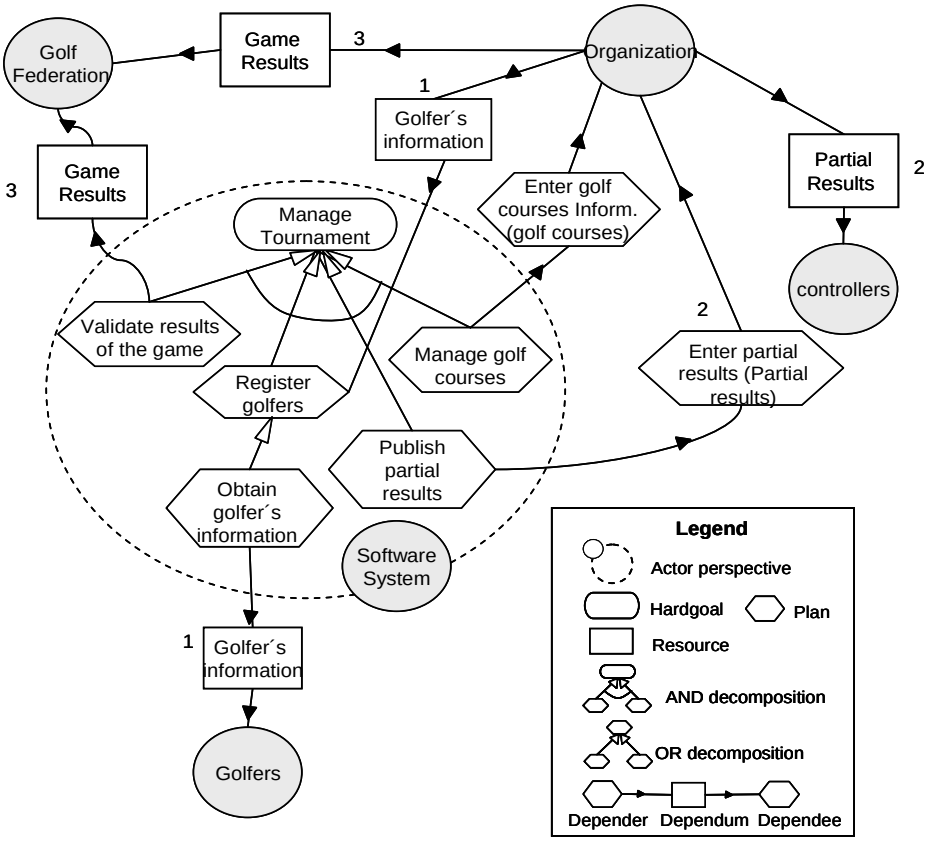


Figure 2 SS-BM generated by the application of the pattern language

6. REFERENCES

[1] Beedle Michael A. cOOherentBPR –A pattern language to built agile organizations, Plop-97, Technical Report #wucs-97-34, Washington University, 1997.

[2] Bresciani P., Giorgini P., Giunchiglia F., Mylopoulos J., and Perini A., *TROPOS:* An Agent-Oriented Software Development Methodology. In Journal of Autonomous Agents and Multi-Agent Systems, 8(3), Kluwer Academic Publishers, May 2004, pp. 203-236.

[3] Bubenko, J. A., Jr and M. Kirikova, Worlds in Requirements Acquisition and Modeling. In 4th European-Japanese Seminar on Information Modeling and Knowledge Bases, edited by K. Sweden, H. Kangassalo and B. Wangler, IOS Press, The Netherlands, 1994, pp. 159-174.

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

[5] Castro J. Kolp M. Mylopoulos J. Towards Requirements-Driven Information Systems Engineering: The Tropos Project. In Information System 27(2), Elsevier, 2002, pp. 365-389.

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

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

[8] Frankel S. David, Model driven Architecture, applying MDA to enterprise computing, John Wiley & Sons, USA, 2003.

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

[10] Kleppe A., Warmer J., Bast W. MDA Explained, the model driven architecture: practice and promise, Addison-Wesley, USA, 2003.

[11] Kulak Daryl Eamonn Guiney, Use Cases requirements in context, Addison-Wesley, USA, 2003.

[12] Martínez Alicia, Castro Jaelson, Pastor Oscar, Estrada Hugo, Closing the gap between Organizational Modeling and Information System Modeling, Proceedings of the VI Workshop on Requirements Engineering (WER 2003), Brazil, 2003, pp 93-108.

[13] Meszaros G. and J. Doble, A Pattern Language for Pattern Writing, in Pattern Languages of Program Design 3, edited by Robert Martin, D. Riehle and F. Buschmann, Addison-Wesley, USA, 1998, pp. 529-574.

[14] 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. In Information Systems 26(7), Elsevier, 2001, pp. 507-534.

[15] Rolland R., Souveyet, C., Plihon, V., Method Enhancement with Scenario Based Techniques, Proceedings of the 11th International Conference on Advanced Information System Engineering (CAISE'99), Germany, 1999, pp 14-18.

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