On the use of *i** for Architecting Hybrid Systems: A Method and an Evaluation Report[†]

Juan Pablo Carvallo[1], Xavier Franch[2]

[1]Universidad Del Pacifico
Carlos Arizaga Toral S/N y Luis Moscoso, Cuenca, Ecuador
jpcarvallo@upacifico.edu.ec

[2]Universitat Politècnica de Catalunya (UPC)
c/Jordi Girona, 1-3, E-08034 Barcelona, Spain
franch@lsi.upc.edu

Abstract. The architectural definition of hybrid software systems is a challenging problem that demands to reconcile stakeholders' strategic needs and components marketplace, whilst defining an appropriate set of services. We have defined a method called DHARMA based on the *i** framework. The goal of this paper is to present an experience report about the use of *i** in large-scale projects. We provide two different viewpoints: the viewpoint of the stakeholder and the viewpoint of the modeller. Apart from general lessons learned, we also provide some insights about the use of *i** in the specific context of architecting hybrid systems using DHARMA.

Keywords: hybrid systems, goal-oriented models, *i**, software architecture.

1 Introduction

Most of current software systems are built as the integration of software components of different nature and origins in which sometimes is referred to as *Hybrid Architecture Systems* [1]. The software components used in these systems include software packages developed by third parties, commonly known as Off-The-Shelf (OTS) components [2] (e.g., commercial OTS components or COTS; free components open source or FOSS [4]; and web services [5]), and also bespoke software and legacy systems.

In this development context, systems are built in an opportunistic manner [6], considering at the same time the environment and the strategy of the organization, the components available in the marketplace (e.g., OTS marketplace, FOSS community), their capacity for being integrated into a single system and interoperate in a transparent manner, and the resources required by their adoption and integration.

The specification of requirements, the selection of the required components, and their adaptation and integration into a single architecture, are some of the problems that have been extensively studied and documented in the literature [7, 8]. However,

[†] This work has been partially supported by the Spanish project TIN2007-64753.

there are some other problems that remain as challenges and demand more study from the scientific community. Among them, we mention: the identification of the strategic needs for which the system is required; the identification of the specific services (bound to these needs) that the system shall offer; and the grouping of the services into atomic domains, which structure the generic architecture of the system and describe the minimum functionality that must be covered for each of the components that will be part of the system.

This paper proposes the DHARMA method to identify the architecture of a component-based system. The generic components that form this architecture may be later substituted in an opportunistic manner (in the sense of [6]) by components of different nature and origins forming a hybrid system. Specifically, DHARMA is based on the use of the *i** framework [9], exploiting its ability to represent actors, dependencies and intentions. And in fact this use yields to the main goal of the paper, namely to provide an empirical assessment on the use of the *i** framework in large-scale projects, both from the point of view of stakeholders and modellers.

The rest of the paper is organized as follows. Section 2 briefly describes the two case studies that have provided most of the feedback for this evaluation report. Section 3 provides a summary of the DHARMA method. Sections 4 and 5 give the details about the use of *i** in the experiences described in Section 2. Finally, Section 6 presents conclusions and future work.

2 The Experience

The work described in this paper in based on two projects developed in Ecuador: the renovation of the IS inside the company ETAPATELECOM, and the elaboration of an IT strategic plan for the Cuenca Airport. We briefly describe both projects in this section.

2.1 The ETAPATELECOM case

ETAPATELECOM is a new entrant telecom company, based in Cuenca, Ecuador. Established in 2002, it currently provides nationwide internet access, data carrying and public and residential fixed telephone services.

To fulfil its deployment strategy, ETAPATELECOM had to face the selection and adoption of several technologies, including several COTS components required by the information system that supports its operation. During this process, the company has used quality models [10] under different forms, and modelling techniques based on *i** to support several activities linked with the adoption and development of information technologies, with more than satisfactory results. Finally, both techniques (quality models and the *i** framework) were combined by means of the COSTUME method aimed at construction quality models for composite systems [11].

2.2 The Cuenca Airport case

Due to the decentralization process conducted in Ecuador in the last few years, the administration of Cuenca's airport was handed from the national Civil Aviation Direction (DAC) to its local municipality. Although the airport was at that moment the 3rd largest in the country, it was severally underused, managing only few domestic flights during the day. The new administration decided to change this situation and developed a strategic plan, designed to increase the airport usage with additional national and international frequencies, as well as other services including the implementation of cargo transportation fleet, a convention center and shopping facilities.

An important part of the strategic plan was oriented to the implementation of the IT services required to support its operation. The *i** framework was used to define basic hardware (network and domotic services required) and the software system architecture. Once the architecture was outlined, several projects were defined to support is implementation. Projects were categorized regarding the hardware-software and generic-strategic dimensions, and prioritized base on the current criticality and time available before they become essential e.g., in relation to the approximate dates in which new services were to be implemented according to the strategic plan.

The defined projects were part of the IT strategic plan which also included the Function and Organization Manual (MOF, acronym for the Spanish term) and the outline of the process manual to be used by the IT staff in software and hardware acquisition, software development and systems operation.

3 The DHARMA method

The DHARMA method (Discovering Hybrid ARchitectures by Modelling Actors) aims at the definition of software architectures using the *i** framework. It has been defined as a result of the experiences reported in Section 2. The process resulting from the method is initiated by modelling the organizational context and ends with the identification of the generic architecture of the software system. By "generic architecture" we mean the identification of the actors that form part of the system, the services that must be covered by each of them and the relationships among them.

The concept of actor is therefore central to the DHARMA method and this is reason that makes the *i** framework highly convenient. System actors represent atomic domains for which OTS components may be identified. By "atomic domain" we mean a group of functions or services that bring some value to the user, such that not other proper subset of this group represents a different significant domain.

The objective of the DHARMA method is not the identification of the final architecture of the system, in which every actor represents a subsystem that may be directly mapped into an individual OTS component (although this may be a particular case). Instead, other cases are possible: an OTS component may cover the services of more than one actor; the services of an actor may be covered by more than one OTS component that altogether provide the required functionality; an actor may be covered by several OTS components that overlap for dependability purposes; or some services of an actor may not be covered by existing OTS components, requiring some bespoke development.

The method has been structured into four basic activities that may iterate or intertwine as needed (see Fig. 1):

- **Activity 1: *Modelling the organizational context.*** The organization and its business model are analysed in detail, in order to identify the role that it plays inside its environment. This analysis surfaces the different types of actors that exist in its contexts, and the strategic needs among them and the organization. The *i** SD diagrams are used to elicit and represent the actors and relationships.
- **Activity 2*: Modelling the environment of the system*.** In this activity, a new system is inserted into the organization and the impact that this system has over the context is analysed. The system may be a typical information system, or it may be a hybrid system including hardware components, maybe with some embedded software. The strategic dependencies identified in the former activity are analysed with the aim of determining which of them may be directly satisfied by the system, and which others are needed by the system providing its operational level. As a result, the dependencies are redirected inside the *i** SD diagram, and also new dependencies arise. The model includes also the organization itself as an actor in the system environment, in which its needs are modelled as strategic dependencies over the system.
- **Activity 3*: Decomposition of system goals*.** In this activity, the system is analysed and decomposed into a hierarchy of goals that are needed to satisfy the strategic dependencies stated by the environment actors. The goals represent the services that the system must provide, to interact with the actors in the environment. An SR diagram for the system is built, using decompositions means-end of type goal-goal (representing then a decomposition of objectives into subobjectives).

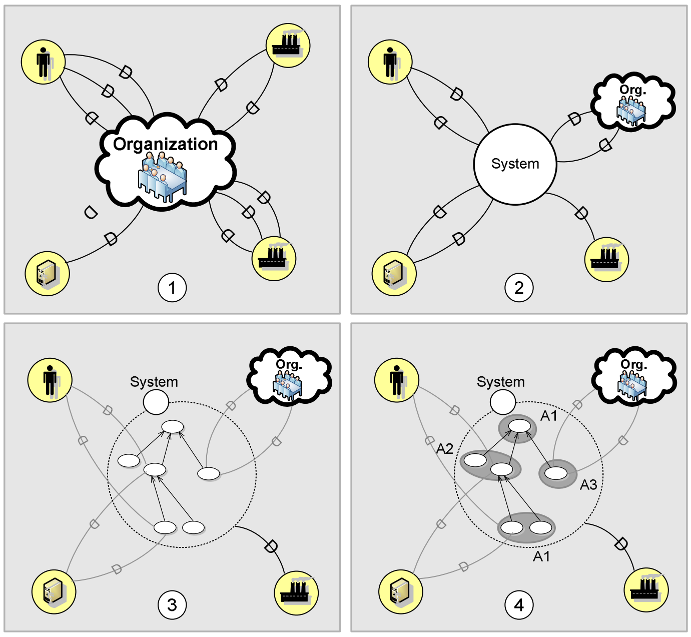

Figure 1. Activities of the DHARMA method.

- **Activity 4:** ***Identification of the system actors.*** The goals included in the SR model are analysed and systematically grouped into subactors that represent atomic domains. The objectives are grouped into services, according to an analysis of the strategic dependencies with the environment and an exploration of the existing OTS marketplace. The relationships between the different actors that form the basic structure of the system are described according to the direction of the means-end links that exist among the objectives included inside them.

4 The *i** Framework from the Stakeholder Point of View

In this section we outline the issues that we found when using *i** models in conjunction with the system stakeholders.

4.1 Initial Modeling

The DHARMA method requires at its first step the construction of an SD diagram modeling the organization environment. Instead of the classical elicitation approach in which the RE expert elicits requirements from stakeholders and represents them using *i**, we opted for a different approach: stakeholders received some training in *i** and were committed to develop their own partial vision of the organization in a SD model.

A first consideration was needed: were the stakeholders going to learn the whole *i** language? Some authors have reported about the difficulty of using the full expressive power of *i** with stakeholders that are not skilled in advanced requirements engineer techniques [12, 13]. After a careful consideration and some feedback, we took several decisions that are reported below and described in the metamodel of Fig. 2, which shows some simplifications with respect to the one defined by Ayala et al. [14]:

- Actors. We treat all actors in a generic manner, without distinguishing roles, positions and agents. The barrier between these concepts is sometime fuzzy, especially when considering the combination of these types and links like is-a, and may provoke some confusion to the *i** novice. Instead, we considered useful to distinguish among four types of actors: human, software, hardware and organizations. Although we didn't bring the distinction into the model itself graphically, we kept traceability of the type through comments.
- Actor links. We kept the two types of main actor links, i.e. is-a and is-part-of. Especially the is-a specialization link became very useful when declaring hierarchies of human roles represented by actors of human type. Note that the actors' type may be used here for correctness conditions, e.g. the specialization of a human actor must also be human.
- Dependencies: contrary to what was expected beforehand, stakeholders very intuitively grasped the difference between goal and soft goal. The concept of subjectiveness was crucial to understanding this difference. Therefore, we kept both types of dependencies. Also resource dependencies had a very clear meaning, namely informational need. On the contrary, task dependencies were considered too much low level, stakeholders found easier

to focus on the level of goals (what the task is going to provide) than on the task itself. We avoid this fourth type of dependency (that may appear later when the expert takes the lead).

– Intentional elements. The most significant difference between the standard *i** and the way we used it was the type of intentional elements inside actors' boundaries. We just supported goals and then, as intentional elements' links, goal decomposition. This decision reduced complexity a lot (sometimes the distinction among goal, task and resource depends on the point of view or the emphasis) and aligned with most stakeholders' way of thinking, where goals play a central role.

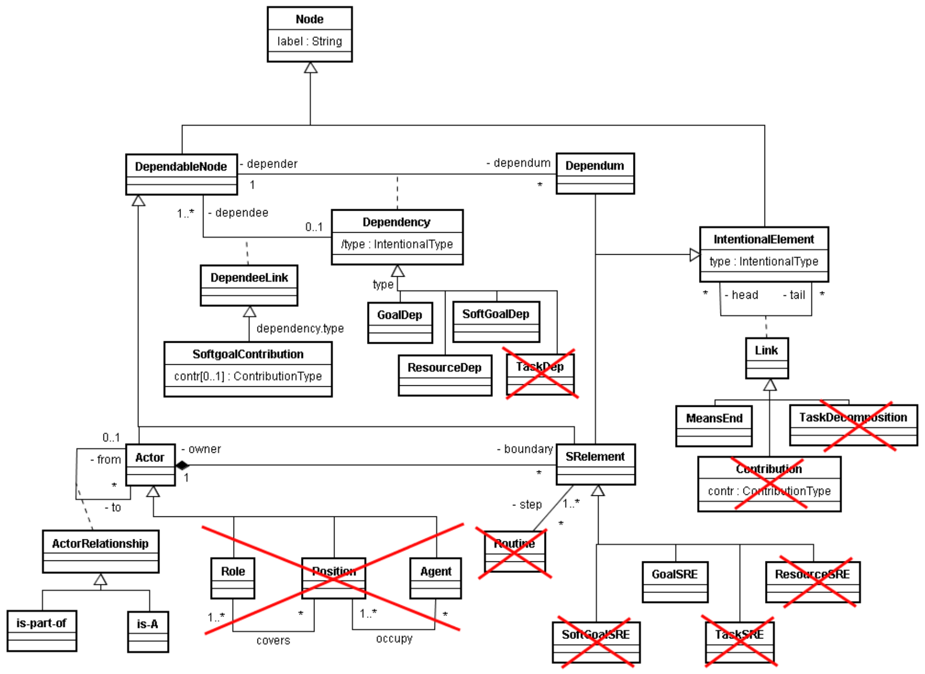

Figure 2. The *i** metamodel as defined in the DHARMA method.

The three tasks that were undertaken during the first activity of DHARMA were then:

– Initial training of stakeholders. Initial Stakeholders' training was conducted more in a workshop-brainstorm formatted session than in formal teacher-students session. After a quick explanation of the basic *i** concepts, conducted by the moderator (a expert in *i**), the concepts were used to create the initial models of the organizational environment. With the guidance of the moderator a first set of environmental actors was brainstormed and then some basic dependencies were proposed and analyzed by participants. The session was about three hours long, and included stakeholders of several areas of the organizations (e.g., financial, administrative, legal, and technical). Blackboards and projectors were used as tools to support the process.

– Individual models built by stakeholders. With the first models constructed, stakeholders were given a week to carefully study them and to propose changes or new versions of the models. Once the resulting models were handled, they were reviewed by an expert in *i** which helped stakeholders to validate the correct usage of the different types of dependencies. It was interesting to find that some of the reviewed models included dependencies among environmental actors and third party actors, even if they didn't have a direct relation with the organization. In some cases they were seen by stakeholders as relevant to complement their understanding of the environment (e.g. the dependency among telephony service regulators and radio and TV services regulators, which were perceived as potential environmental actors, in the case of future joint ventures with that kind of service providers). This confirmed us that even if they were not technical staff, they got a good understanding of the basic *i** modeling skills.
– Consolidation of the different models into one. Once the individual models were validated, the team of *i** experts created a consolidated version including all the identified actors and the proposed dependencies. Redundancies were eliminated and similarities were marked in order to validate if different stakeholders were referring to the same concepts. After the consolidated models were completed, final workshops ware conducted in order for stakeholders to validate the resulting models and to align their views on the problem. At this point it was obvious that stakeholders were already very familiar with more abstract concepts such as soft-goals. This made easy the communication among technical and non technical staff and helped to conduct the workshops in a very proactive way.

Another point worth to mention is tool support. There are several *i** modelling tools available in the community (see [15] for a survey) and even recently an XML model interchange format named iStarML [16] based on the i* metamodel proposed in [14] has been defined and is being adopted by several tools. But of course, using these tools implies learning a new technology. And it must be remarked that the use of *i** in these projects was limited to modeling, no further treatments were required. As a consequence, the functionalities needed from these tools were quite limited. To sum up, we decided to use a generic drawing tool like MS Visio instead of using a new technology. This decision reduced the stakeholders' learning curve and allowed to take use of some facilities of MS Visio that became useful:

– The use of connection links to easily and permanently link actors and intentional elements.
– The use of the grouping by layers to control the visibility of the model. We assigned each stakeholder partial model to one layer, therefore during the analysis if a part of the model (developed by a stakeholder) was not relevant, it was easily hidden. Of course this was possible because of the particular characteristics of our SD models, which are radial (dependencies always stem/go from/to the system to/from a context actor).
– We took the chance to change the graphical representation of dependencies from the standard definition (use of oriented "D") by a standard directed arrow (this change is also recommended by [17] in a recent work).

– Some diagrammatic advices were issued. For instance, use of straight lines instead of curved lines for representing dependencies, making easier manual reallocation and the preliminary drawing of quadrants, as a mean to delimit the areas of the diagram to be filled by each actor and their particular dependencies, proved useful to support this activity.

4.2 The Model as a Communication Mean

In projects involving people with different background and skills, it is quite normal to find that many of them have their own view of the problem and goals on the project. *i** has proven to be a good way to align the different views and make people work together towards the achievement of the project, with the same concepts in mind.

During the workshops, the organization and its goals were discussed among participants. The produced environmental draft models were used as framework to drive the discussion. In the process several mismatches were identified; among them we can mention the following cases (illustrated with some examples from the ETAPATELECOM case):

– Addition of actors: Some actors were not originally included in the model, but after some discussion they became obviously required. This was the case of the *Prepaid Services Vendor* actor, proposed by the commercial staff. It was required by the organization to satisfy the goal *Prepaid Services Sold*, whilst it required from the organization the *Services Activation Cards* as a resource and the *Prepaid Services Consumption Controlled* as a goal.

– Elimination of actors: Some of the participants proposed the incorporation of new actors at some stage of the process, but after a more detailed review it became clear that they were not relevant. This is the case of the *Technology Provider* actor; it was originally introduced because of the concern of the financial staff, in relation to the criticality of the provision of several components required by the organization to construct its operations platform. After some discussion it was removed because it was perceived as an incidental actor, for which no permanent dependencies existed.

– Refinement of dependencies: During the workshops, it was quite normal to identify new dependencies or to remove some of them in order to refine the model. In addition, some dependencies were redefined as other kinds of dependencies, e.g. the *Provide Quality of Service* soft-goal originally proposed by the technical staff was later changed to a goal; in order to maintain the operation license it is required as a non negotiable goal by the *Regulation Authority* actor.

5 The *i** Framework from the Modeler Point of View

In this section we report our experience as requirements engineering experts about the use of *i** in industrial projects.

5.1 Drawing of the Diagram

Although it may seem strange that we start this section by the issue of drawing, in fact *i** is a visual notation that heavily relies on the graphical representation of its models. As explained in the former section, stakeholders build their partial vision of the system using MS Visio and producing an *i** SD model. These models have to be merged into one after some consolidation conducted by the requirements engineer expert. As a result, we get a big single *i** model. This model is:

– Difficult to build. The different partial SD models have to be integrated into one. This integration must be done by hand (copy&paste plus manual reallocation of elements). Diagrammatic tools in the *i** community do not support this functionality neither. Therefore, this task becomes cumbersome.

– Difficult to modify. After the SD model is consolidated, it is modified in the next steps. These modifications are addition and removal of actors and intentional elements, and reallocation of links. Also these tasks are cumbersome.

We may say that there is a lot of work to do with *i** diagrammatic tools until they can be considered satisfactory for large-scale projects. As an alternative, we have started to represent *i** SD models as tables with the same rows and columns, and cells represent links between them. This representation solves the problems above, although the model is more difficult to be comprehended as a whole. Probably, a model-view-controller architecture supporting these two views altogether (and even some other, like the directory-like structure promoted by the J-PR*i*M tool [18]), and the addition of features like the layered control mentioned in Section 4, are the key to overcome the inherent difficulty of representing *i** models.

5.2 Reusability

We may consider three types of reusability:

– **Intra-process reusability.** SD Environmental Models describe the dependencies among the organization (or the system) and the actors on their environment. Thus, when describing the dependencies with respect to a particular environmental actor, we are implicitly describing the dependencies in the environment of the given actor with respect to the organization (or the system). This intra-process reusability became evident from the beginning when performing our first industrial experiences (prior to the ones described in Section 2). Whilst studying the *e-Mail Systems* domain, *Mail Clients* where included as actors in their environment (see Fig. 3, Top, for an excerpt of the e-Mail Systems environmental mode). When studding the *Mail Clients* domain in a latter process, the *e-Mail Systems* actor was included as environmental actor together with all the dependencies already identified.

– **Inter-process reusability.** Different organizations may share sets of elements in their environment. This is a well-known fact not only for organizations sharing the same vertical segment, but also for those in different market segments. Thus, regarding this issue two kinds of reusability exist:

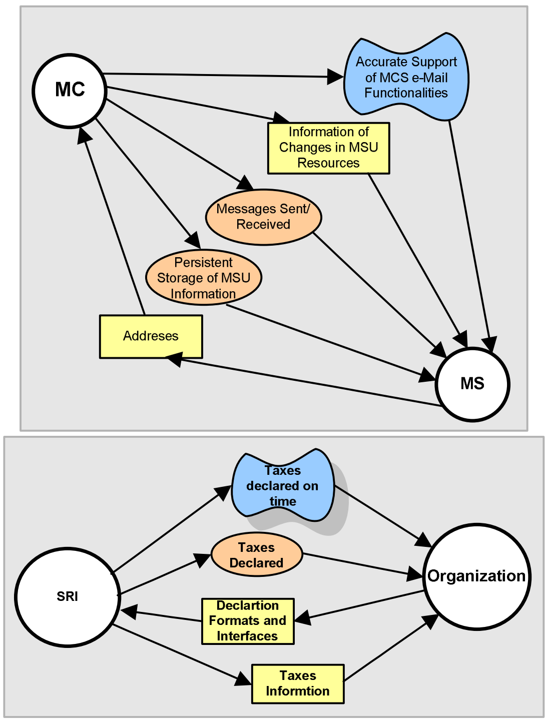
MC
Accurate Support of MCS e-Mail Functionalities
Information of Changes in MSU Resources
Messages Sent/ Received
Persistent Storage of MSU Information
Addreses
MS
Taxes declared on time
Taxes Declared
SRI
Organization
Declartion Formats and Interfaces
Taxes Informtion

Figure 3: Top: Excerpt of the Mail Client (MC) and Mail Server (MS) SD model; Bottom: Excerpt of environmental model showing the dependencies among the Ecuadorian Tax Agency and the Organization (ETAPATELECOM / Cuenca's Airport).

- Vertical reusability. When performing different DHARMA processes in organizations sharing the same vertical market segment. In these cases, most of the elements in the environmental model of one organization (or system) can be reused in the environmental models of others, e.g., telecommunications companies sharing the same regulators, users, interconnection providers, dealers, etc.
- Horizontal reusability: When performing different DHARMA processes in organizations with different vertical market segments. In these cases some commonalities can be found and model elements reused. For instance, both ETAPATELECOM and the Cuenca Airport shall periodically report about their income and expenses to the Ecuadorian Taxes Agency (SRI). Thus, the area of the model describing this environmental actor that was first constructed for the ETAPATELECOM case (see Fig. 3, Bottom), was latter reused in the airport experience.

In general, inter-process reusability increases as the explored domains are more similar. Regarding this issue, four levels of abstraction regarding similarity of their business strategy (e.g., service-oriented CRM, manufacture-oriented ERP, logistics- and transportation-oriented SCM, etc.) can be established. From the most similar to the most dissimilar: organizations in the same vertical market sharing the same business strategy; organizations in the same vertical market with different business strategies; organizations in different vertical markets but sharing business strategies; organizations in different vertical markets with different business strategies.

– **Knowledge reusability.** As stated in the previous paragraphs, organizations share commonalities at different levels. Therefore it is not an unusual fact to find parts of models that can be reused as detailed patterns in other experiences. For instance, let's consider again the e-Mail Systems case, which used the activities of the COSTUME method [11] to identify the system architecture and to build the artifacts required for the selection of its components. Some of the actors (with their respective SR models as goal-subgoals decompositions) identified in this case were reused both in the ETAPATELECOM and the Cuenca's airport cases, namely the ones corresponding to the Mail Servers and Directory Servers system actors (see Fig. 4).

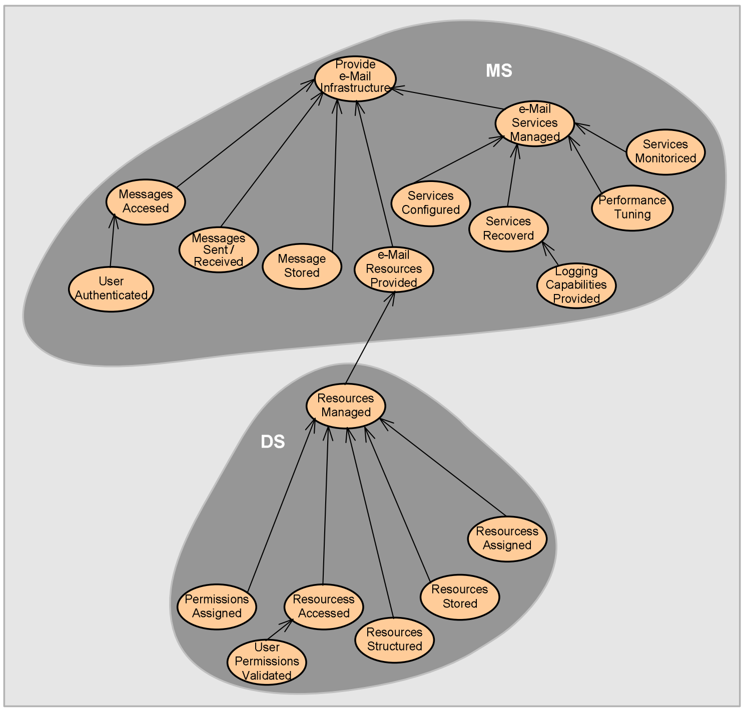

Figure 4: Mail Server (MS) and Directory Services (DS) system actors with their SR decomposition

5.3 DHARMA-related lessons learned

There are some additional lessons that emerged from the application of DHARMA:

– **Environmental models refinement.** Although the refinement of dependencies in environmental models was mainly driven by stakeholders' participation and understanding on the problem, there are some tips that help the modeler:

- Base the identification of environmental actors on several sources of information: use case diagrams; goal-oriented modeling techniques; identification of organizational roles supported by: the review of ontologies (e.g. OpenCyc), standards of professional bodies (e.g. SWEBOK), or organizational theory literature [19]; or the adoption of social patterns [20].
- To define environmental dependencies: first, identify which goals of the environmental actors depend on the organization (or the system) and vice versa, and represent them by goal dependencies. To simplify the process, omit the dependencies that do not involve the organization (or the system) as an actor. Environmental models shall be kept as simple as possible focusing only on the services required from the organization (or the system).
- Next, identify the resources needed to satisfy these goal dependencies and model them with resource dependencies. Note that resources may be physical or informational.
- Finally, analyze each goal dependency over the organization (or the system) with respect to catalogues of non-functional requirements e.g. the ISO/IEC 9126-1 standard, and include in the model a soft-goal for every subcharacteristic considered crucial to satisfy this goal.
- Tend to avoid task dependencies in the model, since they are rather prescriptive. A task dependency represents one particular way of attaining a goal; it can be considered as a detailed description of how to accomplish a goal.

– **System models refinement**. We found the following guidelines useful to conduct this activity:
  - To construct the SR model of the system, first identify the main goal of the system and draw it as the root goal of the diagram.
  - Reduce this goal into sub-goals by means of goal-goal links, representing the main identifiable functional areas that the system is expected to provide and link external dependencies to them whenever appropriated. This first decomposition is achieved by exploring the dependencies that environmental actors have on the system.
  - Repeat the previous process for each of the sub-goals identified until the obtained sub-goals represent services atomic enough, such that it does not makes sense to further reduce them. A rule of the thumb to validate the decomposition is that all of the leaf goals of the hierarchy must be linked to at least one environmental dependency. If one leaf goal is not linked to any external dependency it can be removed, unless it is considered critical for the fulfillment of its predecessor.
  - The process is complete when all the environmental dependencies have been considered and linked to the appropriated sub-goals required for their fulfillment, in case of incoming dependencies, or to the ones which depend on them, in the case of outgoing dependencies.

– **System actors' identification.** We identify two kinds of system actors that can be present in system models:

- *Core system actors.* This kind of actors provides the core functionality of the system. Because of this, in many cases the system as a whole adopts their name. Most of the committed and critical dependencies of environmental actors are usually linked to them. Some examples of core system actors are the Mail Server in e-Mail Systems, the telecom billing system and the ERP system in the ETAPATELECOM case, or the airplane guidance and monitoring system in the Cuenca's airport case.
- *Supporting system actors.* Supporting actors do not provide the core functionality of the system. Instead they offer services required by the core actors in order to fulfill some of their external dependencies with environmental actors (e.g., the telecom billing services system relies on the platform mediation interfaces for services to be automatically activated / deactivated). All supporting actors have dependency links with core actors, but not necessarily among them. They may also have dependency links with environmental actors, but usually not in relation to the core functionality of the system.
- Systems may include more than one core actor. Regarding supporting system actors, they are not mandatory and some systems may not include them (although this is not the usual case). With these considerations in mind, in the extreme case, a system will include one core system actor and at least one additional actor.
- The identification of system actors is guided by the goals identified in the SR model of the system. These goals reveal services that are expected to be covered by system actors. Their assignment to system actors can be supported by reviewing several sources of information, such as online COTS components markets or COTS components taxonomies. Experience, Internet browsing and Google search for key words included in the defined goals, proved to be the most effective ways to conduct this activity

– **Components interoperability:** Decisions on system architecture rely in several aspects but mainly in the ability of components to interoperate and work together as whole system. To support the decision making process, we found very useful to create an enriched SD model of the system after system actors were identified. To obtain the model we follow the process below:
  - The set of goals and sub-goals assigned to a system actor (see Fig. 5, a) have to be abstracted to a circle representing the actor (see Fig. 5, b).
  - The circles representing the actors inherit all the environmental dependencies assigned to the goals that define their services (see Fig. 5, b).
  - The end links among the actors are replaced by goal dependency links. In these links the actor of the end goal is the depender, the actor of the means goal the dependee and the goal the dependum (see Fig. 5, c).
  - Internal goal dependencies among system actors can be refined with a process similar to the one proposed for environmental process refinement, for obtaining a detailed interoperability model (see Fig. 5, d).

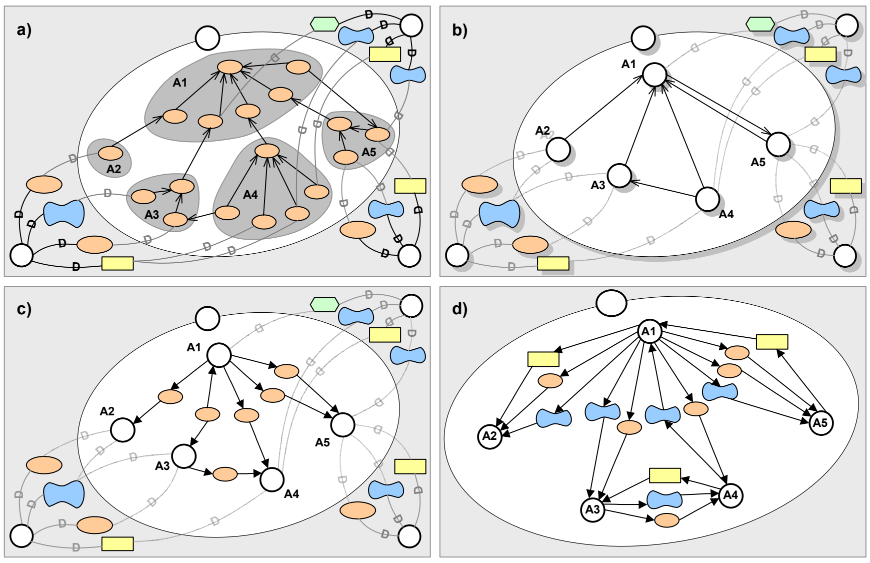

Figure 5: Process to obtain an interoperability model

6 Conclusions and Future Work

In this paper we have presented an experience report about the use of *i** in the particular case of architecting hybrid systems using the DHARMA method. In a few words, the framework has demonstrated to be useful both for stakeholders and modellers provided that some simplifications of the model are done, remarkably the conversion of the rich SR models into goal-subgoal decomposition graphs.

We summarise in a sentence our view of each of the issues evaluated in [12]:

– **Refinement.** (1) SD: the three modeling steps, i.e. first joint workshop, then each stakeholder, last the modeler, seem to support stepwise refinement of the SD model; (2) SR: much easier than usual since decomposition is just goal-goal.
– **Modularity**. Somehow supported by the use of the MS Visio layer concept.
– **Repeatability**. Considering the sense given by [12], the use of a reduced *i** framework makes easier to use the framework in a uniform way.
– **Complexity management**. Again the use of a reduced framework supports this.
– **Expressiveness**. On the contrary, our proposal clearly damages the high expressiveness of *i**, although throughout the paper it has been argued that the concepts kept are the fundamental ones for stakeholders.
– **Traceability**. Not explicitly supported, although it has been said that comments are used to trace which stakeholder provided which part of the model.
– **Reusability**. Both intra- and inter-process reuse are supported.
– **Scalability**. The use of a reduced set of concepts and some diagrammatic conventions make the *i** models a bit more scalable than usual (trade-off with expressiveness). But it is not clear yet how much significant are the differences.
– **Domain Applicability**. It applies well to the hybrid systems architecting domain.

As future work, we are planning to extend a preliminary work in relation to hybrid systems evolution. In this work the modules of several legacy systems have been modeled as system actors and the dependencies among them have been stated, to make explicit the interoperability among them. In a second stage of the process, an ordering sequence has been established in relation to the priority in which some of the modules need to evolve to new versions. In this way the impact of the replacement of the modules in relation to other system components is made evident using a visual notation; as a consequence system evolution can be planned with more evidence of the effort required.

References

[1] *Proceedings of the 7th International Intl. Conference on Composition-Based Software Systems* (ICCBSS), IEEE, 2008.
[2] J. Li *et al.* “A State-of-the-Practice Survey of Risk Management in Development with Off-the-Shelf Software Components”. *IEEE TSE*, 34(2), 2008.
[3] A. Mohamed, G. Ruhe, A. Eberlein. “COTS Selection: Past, Present, and Future”. CBSE 2007.
[4] J. Feller, B. Fitzgerald. *Understanding Open Source Software Development*. Addison-Wesley, 2002.
[5] M.P. Papazoglou. *Web Services: Principles and Technology*. Prentice-Hall, 2008.
[6] G. Kotonya, S. Lock, J. Mariani. “Opportunistic Reuse: Lessons from Scrapheap Software Development”. CBSE 2007.
[7] C. Alves, F.M.R. Alencar, J. Castro. “Requirements Engineering for COTS Selection”. WER 2000.
[8] X. Burgués *et al*. “Combined Selection of COTS Components”. ICCBSS 2002, Springer.
[9] E. Yu. *Modelling Strategic Relationships for Process Reengineering*. PhD Dissertation, University of Toronto, 1995.
[10] X. Franch, J.P. Carvallo. “Using Quality Models in Software Package Selection”. *IEEE Software,* 20(1), 2003.
[11] J.P. Carvallo *et al*. “COSTUME: A Method for Building Quality Models for Composite COTS-Based Software Systems”. QSIC 2004, IEEE.
[12] H. Estrada *et al*. “An Empirical Evaluation of the *i** Framework in a Model-Based Software Generation Environment”. CAiSE 2006, Springer.
[13] M.C. Annosi *et al*. “Analyzing Knowledge Transfer in Software Maintenance Organizations using an Agent- and Goal-oriented Analysis Technique – an Experience Report”. iStar 2008, CEUR Workshop Proceedings.
[14] C. Ayala *et al*. “A Comparative Analysis of *i**-based Agent-Oriented Modeling Languages”. SEKE 2005.
[15] URL: http://istar.rwth-aachen.de/tiki-index.php?page_ref_id=21.
[16] C. Cares *et al*. “iStarML: an XML-based Model Interchange Format for *i**”. iStar 2008, CEUR Workshop Proceedings.
[17] D.L. Moody, P. Heymans, R. Matulevicius. “Improving the Effectiveness of Visual Representations in Requirements Engineering: An Evaluation of *i** Visual Syntax”. RE 2009, IEEE.
[18] G. Grau, X. Franch, S. Ávila. “J-PRiM: A Java Tool for a Process Reengineering *i** Methodology”. RE 2006, IEEE.
[19] R.L. Daft. *Organization Theory and Design*. Thomson, 1992.
[20] A. Fuxman *et al*. “Information Systems as Social Structures”. FOIS 2001.