An Empirical Evaluation of the *i** Framework in a Model-Based Software Generation Environment*

Hugo Estrada[1,2], Alicia Martínez Rebollar[1,3], Oscar Pastor[1], and John Mylopoulos[4]

[1] Valencia University of Technology, Valencia, Spain
{hestrada, alimartin, opastor}@dsic.upv.es
[2] CENIDET, Cuernavaca, Mor. Mexico
[3] ITZ, Zacatepec, Mor. Mexico
[4] University of Trento, Italy
jm@cs.toronto.edu

Abstract. Organizational modelling has been found to be very effective in facilitating the elicitation of requirements for organizational information systems. In this context, the *i** modelling framework has been used widely in research and – some – industrial projects. However, no empirical evaluation exists to-date to identify areas of strength as well as weaknesses of the framework. This paper presents the results of an empirical evaluation of *i** using industrial case studies. These were conducted in collaboration with an industrial partner who employs an object-oriented and model-driven approach for software development. The evaluation of *i** uses a feature-based framework. The paper reports on lessons learned from this experience, both in terms of strengths and detected weaknesses. The results of this evaluation can play an important role in guiding extensions of the *i** framework.

1 Introduction

Organizational modelling is a promising approach for early requirements analysis during the development of organizational information systems. In this context, the *i** modelling framework [13] offers a well-founded and widely used set of concepts for describing organizational settings made up of social actors who have freedom of action, but also depend on other actors to achieve their goals.

The *i** framework and its methodological extensions (such as GRL [4] and Tropos [2]) have been used in a wide range of application domains, such as business modelling, object-oriented software development, software requirements elicitation, agent-oriented software development, modelling and analysis of non-functional requirements, security requirements, trust and privacy requirements, and more. In all these applications, *i** concepts have been used to capture social and intentional elements of each specific domain, thereby supporting software development. However, despite well-known theoretical advantages of *i**, there have been no empirical studies that confirm its usefulness and identify potential weak spots.

* This work has been partially supported by the MEC project with ref. TIN2004-03534, the Valencia University of Technology, Spain, Care Technologies Enterprise Inc. and the University of Trento, Italy.

E. Dubois and K. Pohl (Eds.): CAiSE 2006, LNCS 4001, pp. 513 – 527, 2006.

The purpose of this paper is exactly this: to present an empirical evaluation of *i**, based on industrial case studies. The case studies were conducted in collaboration with Care Technologies Inc. (http://www.care-t.com), a software company that has adopted the OO-Method for software development. OO-Method is a model transformation method that relies on a CASE tool ([7]) to automatically generate complete information systems from object-oriented conceptual models. The OO-Method can be viewed as a computer-aided requirements engineering (CARE) method where the focus is on properly capturing system requirements in order to manage the complete software production process. The resulting conceptual model specifies what the system is (problem space). Then, an abstract execution model is provided to guide the representation of these requirements in a specific software development environment that is focused on how the system will be implemented (solution space).

The transformation from a conceptual to an execution model (implementation) is effected by a Conceptual Model Compiler. The compiler exploits precise transformation rules from conceptual modelling constructs to corresponding software representations. The execution model is based on a component-based architecture in order to deal with the characteristics of component-based systems. The final software product's functionally is equivalent to the requirements specification. Figure 1 presents a graphical representation of the OO-Method.

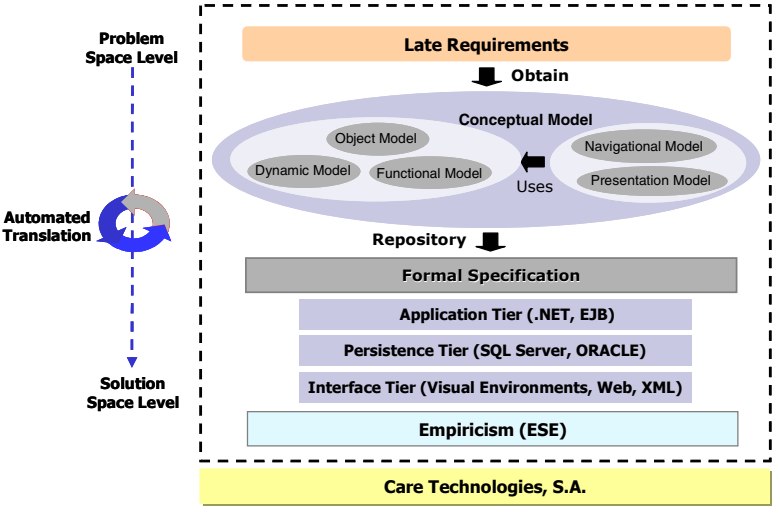

Fig. 1. The OO-Method approach for model-driven software development

Despite the major advantage of the OO-Method in automatically generating information systems, there are disadvantages as well. Specifically, there are currently no mechanisms for acquiring the requirements of an information system. Accordingly, the next step in developing further the OO-Method consists of adding a new phase of organizational modelling as a starting point to determine the correct requirements for the information system-to-be.

In performing the empirical evaluation, our objective was to determine possible extensions to *i** that would make it suitable for inclusion in the OO-Method modelling and methodological framework. Consequently, the features selected for measurement in this empirical evaluation are inspired by model-driven approaches.

The rest of the paper is structured as follows: Section 2 presents an overview of the *i** framework. Section 3 presents related works. Section 4 describes the evaluation framework we used to assess *i**, while Section 5 presents the results of the evaluation conducted during the case studies. Section 6 discusses issues to be addressed in future versions of *i**. Finally, Section 7 concludes and briefly discusses future work.

2 An Overview of the *i** Framework

The *i** modelling framework [13] views organizational models as networks of social actors that have freedom of action, and depend on each other to achieve their objectives and goals, carry out their tasks, and obtain needed resources.

The *i** framework is made up of two models that complement each other: the *strategic dependency model* for describing the network of inter-dependencies among actors, as well as the *strategic rationale model* for describing and supporting the reasoning that each actor goes through concerning its dependencies on other actors. These models have been formalized using intentional concepts from Artificial Intelligence, such as *goal*, *belief*, *ability*, and *commitment*.

A strategic dependency model (SD) is a graph involving *actors* who have strategic dependencies among each other. A dependency describes an "agreement" (called *dependum*) between two actors: the *depender* and the *dependee*. The *depender* is the depending actor, while the *dependee* is the actor who is depended upon. The type of the dependency describes the nature of the agreement. Goal dependencies represent delegation of responsibility for fulfilling a goal; softgoal dependencies are similar, but their fulfilment cannot be defined precisely (e.g., because it is subjective and/or partial). Task dependencies require the *dependee* to perform a given activity, and resource dependencies require the *dependee* to provide a resource. In *i** diagrams, actors are represented as circles; goals, softgoals, tasks and resources are respectively represented as ovals, clouds, hexagons and rectangles. Dependencies have the form *depender* → *dependum* → *dependee*. In the SD model, the internal goals, plans, and resources of an actor are not explicitly modelled. The focus in such models is on external relationships among actors.

The strategic rationale model (SR) represents through means-ends relationships how stakeholder's goals and softgoals can actually be fulfilled through the contributions of other actors. A strategic rationale model is a graph with four types of nodes -- goal, task, resource, and softgoal -- and two types of links. Means-ends links represent alternative sub-goals/tasks for fulfilling a goal/task, while decomposition links represent necessary sub-goals/tasks for fulfilling a goal/task. A strategic rationale graph explains and accounts for each actor's dependencies on other actors.

3 Related Work

There have been several studies that compare the Tropos agent-oriented software development methodology with others in the same family. Shehory and Sturm [8] propose a feature-based framework for evaluating and comparing agent-oriented methodologies. The framework examines various aspects of each methodology: concepts and properties, notations and modelling techniques, processes, and pragmatics. More recently, the same authors [11] performed an empirical evaluation based on case studies for several agent-oriented methodologies including Tropos (Gaia, Tropos, MaSe, and OPM/MAS). The case studies employed students taking a computer science course. An important contribution of this work is the use of a framework for evaluating and comparing agent-oriented methodologies that is based on a set of pre-defined criteria (features).

Dam and Winikoff [3] also performed an evaluation of agent-oriented software development methodologies (MaSe, Prometheus, and Tropos) using an attribute-based evaluation framework. In this evaluation, a set of summer students developed the same case study using different methodologies. The students then filled out a questionnaire to give feedback about their experience in understanding and using the methodologies based on the selected features. The authors of this evaluation also collected comments from authors of the methodologies using the same questionnaire that the summer students had completed. One of the interesting elements of this work is the attempt to eliminate misconceptions by taking into account comments from the authors of each methodology.

Along similar lines, Sudeikat et al [12] present an evaluation framework for agent-oriented methodologies that takes platform-specific criteria into account. The specific objective of this study was to determine how the methodologies under evaluation (Mase, Tropos and Prometeus) match up with the Jadex agent platform.

Our empirical evaluation is somewhat different than all of the above. Firstly, our evaluation focuses on a modelling framework rather than a software development methodology. Secondly, the object of our study is a specific modelling framework, rather than a comparison of several. Moreover, our evaluation studies how well *i** matches a specific software development context (model-based software generation) in practice, rather than analyze *i** in the abstract. Finally, all the studies mentioned used students working on toy problems. This represents a major limitation of these studies and a major point of difference from our work.

There are also reported studies that use *i** for some application. In most of these studies, the modellers were well-acquainted with *i** concepts and their use. We are only aware of one study [5] where *i** was evaluated (along with other modelling techniques) in practice by modellers who are not researchers working with *i**. In our study, the modellers were practitioners performing organizational modelling in their daily work.

4 A Feature-Based Evaluation Framework

The empirical study of *i** was based on a feature-based framework. Such a framework consists of a set of features that can be properties, qualities, attributes, or characteristics. An evaluation was conducted by *evaluators* who assigned a judgment (*value*) of how

well each *feature* was supported by the *subject* of the evaluation. For our study, the features were selected on the basis of their relevance to model-driven software generation.

The empirical evaluation was implemented using three real-life projects that were developed in parallel by three different development teams. The composition of the development teams was as follows: (i) Team 1 consisted of three expert analysts in the use of advanced tools for generating conceptual schemas from requirements models[1]; (ii) Team 2 included three expert analysts in the use of the CASE tool for automatically generating information systems from conceptual models[2]; (iii) Team 3 included two expert analysts in the use of *i** for business modelling.

The three case studies were conducted in isolation, i.e., with no exchange of information among participant teams. This was done in order to avoid the empirical analysis being affected by the different levels of knowledge about *i** by the teams involved.

The evaluation was conducted in five steps. The first step was devoted to the determination of a set of features (in the context of a model-based transformational approaches) to be measured. The second step consisted of training the three teams, where details about the concepts and proper use of *i** were given out, using original *i** sources and basic teaching support. In the third step, *i** was used to develop the three case studies. The fourth step consisted of evaluating the results of each team. To accomplish this, each participating team evaluated *i** for each relevant feature. The final step consisted of analyzing the results and drawing conclusions about the strengths and weaknesses of *i**.

As indicated, the case studies were real industrial projects. The goal of the development teams was to represent relevant business processes for each project using *i**. The domains of the three projects were: (i) Technical meeting management: model business processes for organizing technical meetings; (ii) Golf tournament management: model business processes for organizing golf tournaments; (iii) Car rental management: model business processes for a car rental company. For the Technical meeting management case study, the organizational environment involves a large number of interactions among participant actors, and a relatively small number of actors´ internal elements[3]. For the Golf tournament management case study, the organizational environment concerns a large number of actors´ internal activities and a small number of actor interactions. On the other hand, the Car rental management case study involves an organizational context with a large number of actors' internal activities and actors' interactions. As such, the case studies had rather different organizational characteristics and ensured that our study would be biased because of similarities in the case studies chosen.

The empirical evaluation of *i** was based on a set of features that have been considered highly relevant in the context of a model-based software development environment. In this specific context, the modelling primitives of a model must provide precise, bidirectional traceability with subsequent stages of the modelling

[1] At the beginning of the evaluation, this team had limited knowledge of *i**.
[2] At the beginning of the evaluation, this team had no knowledge of *i**.
[3] The internal elements are those goals, plans, softgoals, and resources (represented inside an actor´ boundary) that account for the actor's behavior.

process. It is important to note that the experiment was designed for practicing analysts who are used to dealing with software production concepts such as model-driven architectures, code generation, object-oriented analysis and late (conventional) software requirements specifications, rather than analysts who are familiar with early requirements. After all, we expect that this will be the normal scenario for *i** use in software production companies. Therefore the determination of relevant features for the study was perhaps the most critical step in the whole evaluation process.

The features chosen were based on three earlier studies comparing agent-oriented methodologies ([6], [10], [3]). By including features used in three different studies, we have tried to avoid biases that arise from using a single set of features that might be well suited for *i**.

The evaluation considered two main aspects of *i**: (i) Modelling Language (Refinement, Modularity, Repeatability, Complexity Management, Expressiveness, Traceability, and Reusability), and (ii) Pragmatics of the Modelling Method (Scalability and Domain Applicability). The features selected for these aspects are listed below.

- **Refinement:** This feature measures the capability of the modelling method to refine a model gradually through stages until the most detailed view is reached [1]. This is a relevant feature because it allows analysts to develop and fine-tune design artefacts at different levels of granularity during the development process [3].
- **Modularity:** the degree to which the modelling language offers well-defined building blocks for building model. The building blocks should allow the encapsulation of internal structures of the model in a concrete modelling construct. This characteristic ensures that changes in one part of the model won't have to be propagated to other parts.
- **Repeatability:** the degree to which the modelling technique generates the same output (i.e., same models), given the same problem. This is a very relevant feature in the context of model-driven approaches, where each modelling element during a specific step of the modelling process corresponds to a modelling element in subsequent steps. Repeatability ensures that a correct result is obtained when a transformation between models is applied. We use this feature to evaluate whether we obtain the same *i** model when the same domain is modelled by different modellers.
- **Complexity Management:** This feature measures the capability of the modelling method to provide a hierarchical structure for its models, constructs and concepts. Model management is a fundamental problem in industrial project settings.
- **Expressiveness:** the degree to which the application domain is represented precisely in terms of the concepts offered by the modelling technique. More concretely, this feature measures the degree to which the modelling technique allows us to represent static, dynamic, intentional and social elements of the application domain.
- **Traceability:** the capability to trace modelling elements through different stages of the modelling process. This feature is important because it allows the user to verify that all elements of one model (e.g., capturing requirements) have corresponding elements during the analysis and design stages, and vice versa. Traceability makes it possible for the analyst to move back and forth between models corresponding to different development stages [3].
- **Reusability:** the degree to which models can be reused. As with software code, this feature is causally related to modularity. If the modelling technique allows the definition of modules, general cases (patterns) can be defined for reuse.

- **Scalability:** the degree to which the modelling framework can be used to handle applications of different sizes. Scalability also measures the degree to which the inclusion of new modelling elements leaves unaffected the understandability of models (also known as extensibility). This feature is causally related to refinement and modularity.
- **Domain Applicability:** the degree to which the modelling framework matches modelling requirements for a particular application domain.

It is true that, for some of the features chosen, one can evaluate *i** (or any other modelling framework, for that matter) on theoretical grounds alone. However, in our study of *i**, we wanted to include a practical evaluation as confirmation of any preliminary theoretical suppositions. Moreover, clearly the chosen features interact. For instance, better modularity management, obviously contributes to easier complexity management. Likewise, reusability contributes to scalability. We are studying such correlations and hope to integrate them in the evaluation framework for future studies. For this work, we focus on the application of the proposed set of features in evaluating *i** in practice.

5 Evaluation Results

The evaluation was conducted over a 9-month period. The average size of the models generated by the three teams had as follows: (i) Technical meeting management: 12 actors, 55 dependencies, 70 actors´ internal activities; (ii) Golf tournament management: 8 actors, 42 dependencies, 103 actors´ internal activities; (iii) Car rental management: 13 actors, 143 dependencies, 219 actors´ internal activities.

The evaluation assigned one of three possible values (*Well supported*, *Not well supported*, and *Not supported*) to each feature. Another output of the evaluation was a list of reasons given by the analysts for a judgement passed. In order to make the evaluation consensual, a meeting was held at the end of each case study. In these meetings, produced diagrams and personal evaluations were presented and discussed. The meetings included in-depth discussions for each feature in order to reach consensus and a final judgement.

One interesting result of the evaluation concerns the differences in the models produced by the participating teams. The members of team 1 were experienced in requirements modelling, although not used to modelling in terms of goals, actors and dependencies. They understood well the concepts underlying *i** (after all, requirements concepts match well i* modelling), and were enthusiastic about using *i** in practice. In this case, resulting models were partially compliant with i* philosophy. Moreover, the analysts of this team detected several areas where *i** lacked mechanisms to guarantee the usefulness of organizational models in generating system requirements.

In Team 2, the analysts were used to working with class diagrams, state and functional models as part of their on-going modelling activities. In this case, i* social and intentional concepts were rather unfamiliar and the analysts tried to use the concepts in the same way they used the concepts they were accustomed to. In this case, resultant models were less compliant with i* modelling philosophy. Moreover, these analysts had a lot to say about the lack of precise definitions for *i** concepts, and guidelines for generating *i** models.

The analysts for Team 3 were experienced *i** modellers. In this case, resulting models were completely compliant with *i** modelling philosophy. However, these models were often too abstract for generating software requirements.

Table 1 presents a summary of the results obtained from the evaluation. The first column indicates the type of each feature, the second column lists the feature itself, while the third column indicates the judgement passed on each feature.

Table 1. Results of the empirical evaluation

Evaluation Criteria	Evaluated issue		Evaluation
Modelling Language	1	Refinement	Not Well Supported
	2	Modularity	Not Supported
	3	Repeatability	Not Well Supported
	4	Complexity management	Not well Supported
	5	Expressiveness	Well Supported
	6	Traceability	Not Well Supported
	7	Reusability	Not supported
Pragmatics	8	Scalability	Not supported
	9	Domain applicability	Well Supported

In the following, we present the evaluation and justification for each feature.

1) Feature: **Refinement.** Evaluation: **Not Well Supported**

Explanation: There are two types of refinement supported by *i**: (i) refinement of strategic dependency models in terms of a more detailed strategic rationale model, where one can see why actors depend on each other; (ii) 2) refinement of actor goals into more concrete subgoals. However, the literature using *i** includes many examples where a rationale model is not the result of a refinement of a dependency model. This kind of refinement can be performed in the boundaries of an actor model.

These types of refinement are useful when analyzing small case studies. However, they have severe limitations when the model grows in size and complexity. The dependency model is too concrete to serve as starting point for the analysis of a large enterprise. In such cases, it may contain many actors with a large number of dependencies corresponding to different business processes, whose union constitutes a very complicated model to manage.

The current version of *i** does not include modelling primitives that allow one to start the modelling process of an enterprise with abstract concepts. These concepts would allow us to incrementally add more detail -- using other, more specific, modelling primitives -- until we reach concrete models of business processes and their actor dependencies. There are also no concepts to structure the different functional units of a complex organization. As a consequence of this absence of high-level refinement facilities, the modelling of complex systems that involve a large number of dependencies among many different actors is problematic for *i**.

2) Feature: **Modularity.** Evaluation: **Not Supported**

Explanation: Based on the empirical evaluation, it was concluded that modularity is not supported in *i**. This is the case because *i** doesn't have mechanisms for using building blocks that can be logically composed to represent different organizational fragments (e.g., business processes). In this context, if a new organizational process is added, this may affect all models constructed so far.

The lack of modularity mechanisms in *i** can be viewed as a consequence of its focus on actor modelling rather than on business process modelling. The modelling mechanisms of *i** are oriented towards the definition of the behaviour of the organizational actors (to satisfy their goals and dependencies) rather than being oriented to the definition of high-level views of the organizational business processes.

Due to this the lack of modularity, rationale models represent a monolithic view where all elements of an enterprise are represented at the same abstraction level without considering any sort of hierarchy. Figure 2 shows an example for the Technical Meeting Management case study where the goal dependency "o*btain quality review*s" and other dependencies associated with this goal (the task dependency: "*send reviews on time*", and the resource dependency: "*review*") are represented at the same abstraction level. This makes it impossible to distinguish the hierarchical level of these concepts, which are represented as dependencies in the same diagram.

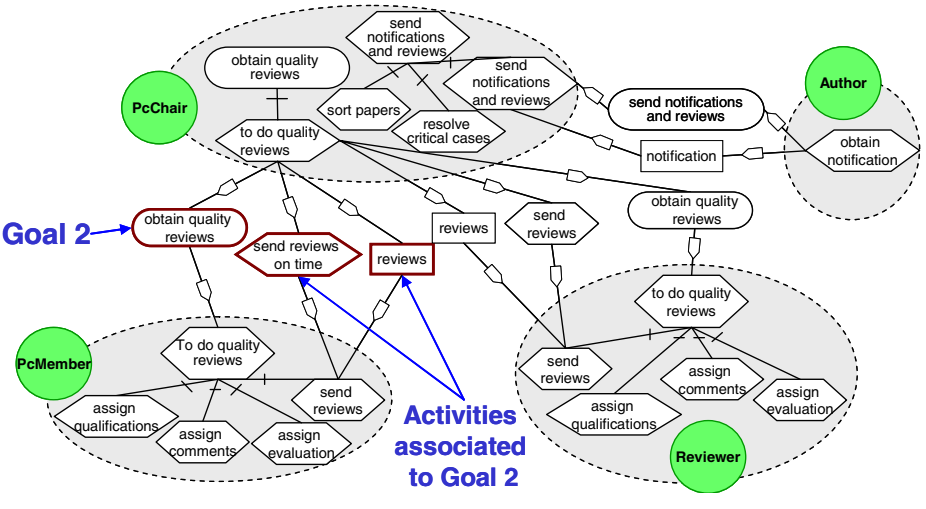

Fig. 2. Representing concepts at same abstraction level

3) Feature: **Repeatability.** Evaluation: **Not Well Supported**

Explanation: One of the key points for ensuring repeatability in a modelling method is the definition of a precise, formal semantics for the modelling constructs. In principle, the modelling constructs of *i** have been defined using formal descriptions and meta-modelling diagrams. These definitions are useful for expert analysts in early requirements. However, for those who are not experts in *i**, these definitions do not provide the necessary, precise support to determine which modelling construct to use when. This problem can also be noted in the *i** literature. There are several examples where very similar settings have been modelled using different primitives.

It is also possible to find in the literature examples of dependencies that do not satisfy the basic semantics of an actor dependency (vulnerable actor, actor who decides how to fulfil the dependency, type of *dependum*). For example, we found cases where the *dependee* of a dependency was incorrectly used as the vulnerable actor, instead of the *depender*. In another example, we found cases where the

dependee of a dependency was incorrectly treated as the actor who prescribes the actions to perform for a delegated task (task dependency), instead of following the guidelines of always placing the *depender* as the actor that prescribes a task dependency. As a consequence of these situations, it is difficult to ensure that a reasonable degree of repeatability is achievable with *i**.

Figure 3 shows an example of these repeatability problems. In this example, taken from the Golf tournament management case study, the process for "Pay for registration of in tournament" was represented in two different ways by the participating analysts: either as a task dependency, where the focus was placed on the activity to be executed; or as a resource dependency, where the focus was placed on the payment, which was viewed a concrete resource relating the actors involved.

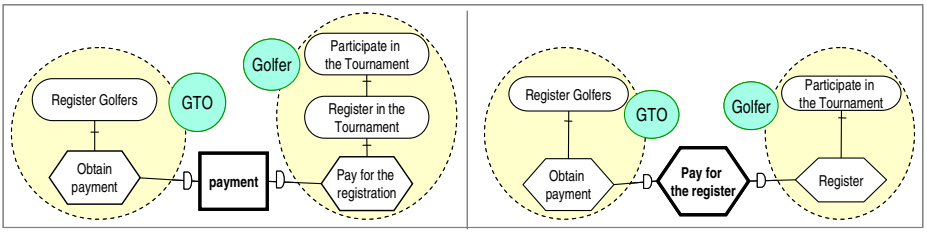

Fig. 3. Example of two different representations for a given single process

4) Feature: **Complexity Management.** Evaluation: **Not Well Supported**

Explanation: In the current version of *i**, it is possible to analyze an enterprise model using two different viewpoints: the strategic dependency model and the strategic rationale model. These viewpoints are useful for small cases, but they are not adequate for dealing with large and complex problems. There are no mechanisms for defining a high-level view of the whole process executed in the enterprise. This high-level view would be properly decomposed following a model-within-a-model strategy, where lower level descriptions are created separately, incorporating all relevant detail.

The limitation in the mechanisms that are provided for managing the system complexity make modelling in *i** unnecessarily complicated. The lack of hierarchies leads to problems such as: a) it is difficult to determine where to start the analysis; b) it is difficult to determine the elements of the model that correspond to each organizational process and/or unit. The lack of hierarchies produces models where several business processes are represented and mixed all together in the same diagram, without any indication of the ownership of each low-level activity nor any information about the boundaries of each individual process (Figure 4).

5) Feature: **Expressiveness.** Evaluation: **Well Supported**

Explanation: There was unanimous agreement among all participants in this experiment that *i** indeed provides a very interesting set of conceptual primitives that make it possible to build pure organizational models on top of conventional requirements ones (mostly, use case-based models). Analysts also agreed on the importance of linking early requirements and late requirements, as a way of connecting software engineering practices with organizational design tasks that are too often performed in isolation by consultants.

The *i** framework was deemed adequate for capturing the relevant concepts of the enterprise, providing mechanisms for representing: a) the social structure of the enterprise, b) the intentional aspects of the organizational actors, c) the activities needed to satisfy the goals of the business actors, d) the relevant resources in the business processes, e) the ability to represent roles, positions and agents to describe the organizational actors, f) the architecture of the enterprise and g) the interaction between the system and external agents.

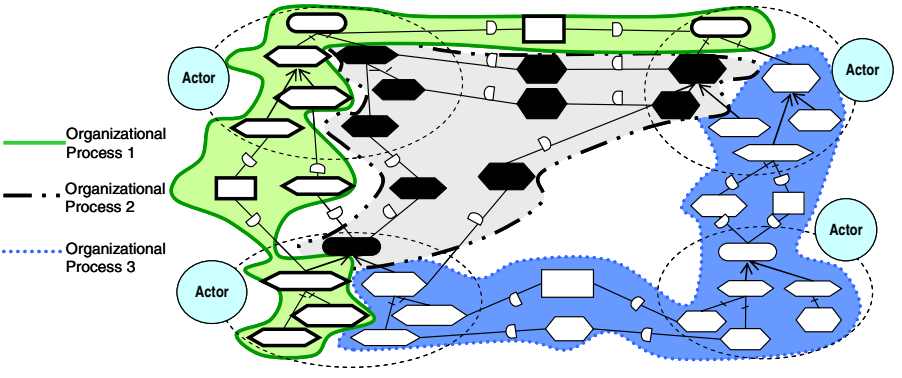

Fig. 4. Representation of different processes in the same diagram

These conclusions account for the difference between *i** and other modelling techniques, which are not as well equipped to represent the social and intentional reasons that underlie the operation of an enterprise. The empirical evaluation allowed us to demonstrate that building an *i** organizational model is very useful for detecting the following problems:

Bottlenecks: This is the case when an actor concentrates a large number of incoming dependencies from other organizational actors. In this case, a failure or delay in this organizational actor could cause a chain reaction in the entire enterprise. The bottleneck problem could be detected by analyzing the dependencies where an actor plays the role of *dependee* of several dependency relationships. We are not aware of other modelling frameworks that account for this kind of analysis.

Vulnerabilities: One of the key advantages of *i** is the explicit representation of vulnerabilities of organizational actors. In this case, if an actor participates in too many dependencies as *depender*, this actor could then become vulnerable if any of the *dependee* actors fail to deliver on their respective dependencies.

Critical Responsibilities: This is the case where an actor concentrates many goal dependencies, which indicate that the actor has many critical responsibilities in the business process. In this case, it may be that the actor has excessive responsibilities and needs help, or at least monitoring.

The explicit representation of these organizational situations is the basis for performing a useful business process reengineering analysis.

6) Feature: **Traceability.** Evaluation: **Not Well Supported**

Explanation: *i** provides modelling flexibility for adding elements to individual dependency and/or rational models. This means that new dependencies can be added to a rationale model that were not previously considered in the corresponding dependency model (Figure 5), and vice versa. This is sometimes useful with respect to modelling flexibility. However, it is also true that this could have negative effects for model-driven approaches, where the elements of a model must have counterparts in previous models. We conclude that *i** does not have precise guidelines for deriving each element of the dependency model from corresponding elements in the rationale model[4].

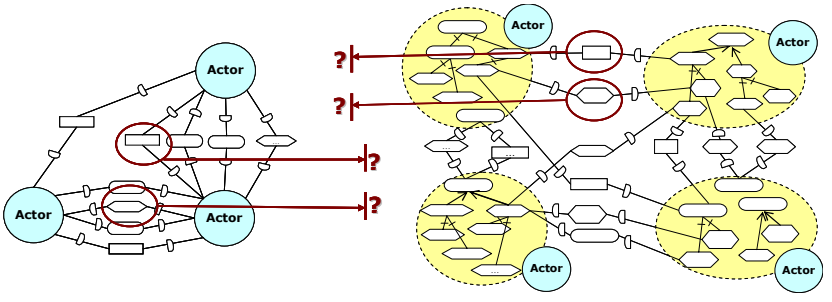

Fig. 5. Representation of problems of traceability

7) Feature: **Reusability.** Evaluation: **Not Supported**

Explanation: *i** does not offer clear mechanisms for properly managing reusability of parts of an organizational model. As mentioned earlier, the lack of good reusability capabilities is a consequence of the absence of mechanisms for modularization. The lack of conceptual building blocks with the required granularity makes it very complicated to reuse certain fragments of a model. Moreover, *i** lacks view definition mechanisms (in the sense of database views) for selecting parts of a monolithic model that capture new viewpoints.

As a consequence of this weakness, modelling projects using *i** must too often start from scratch, without taking advantage of previous projects for similar domains.

8) Feature: **Scalability.** Evaluation: **Not Supported**

Explanation: This is probably the best-known and widely acknowledged problem of *i**. There are simply no clear mechanisms for managing the scalability of strategic models in *i**.

For small problems *i** clearly works fine. However, when the modelling problem grows in size and complexity, the large number of elements represented in the same diagram makes their systematic use and analysis very complicated, when not completely impossible. The scalability problem is also a direct consequence of the lack of mechanisms for modularization, and the inability to put together an abstract view of the high-level business processes of an enterprise. Consequently, all modelling elements for representing the semantics of a specific business process must

[4] Tropos [2] supports such a process that ensures that each element of every dependency model has counterparts in some rational model, and vice versa.

be placed in the same diagram. Figure 6 shows an example of the high number of modelling elements in a diagram for only a fragment of a business process. And this is a very small fragment of the case study.

In summary, the lack of mechanisms for managing scalability is one of the greatest problems for the real applicability of *i** modelling.

9) Feature: **Domain applicability.** Evaluation: **Well Supported**

Explanation: *i** has an ontology and a corresponding notation that we found well suited for organizational modelling. It is also appropriate for the analysis of late requirements. The conceptual primitives are expressive enough to be applied in different domains, and they are appropriate for expressing properties that an organizational model must include. The semantics of the social concepts could also be applied, for example, to present dependencies within and between communities of systems, or even to represent the dependencies between an information system and its stakeholders.

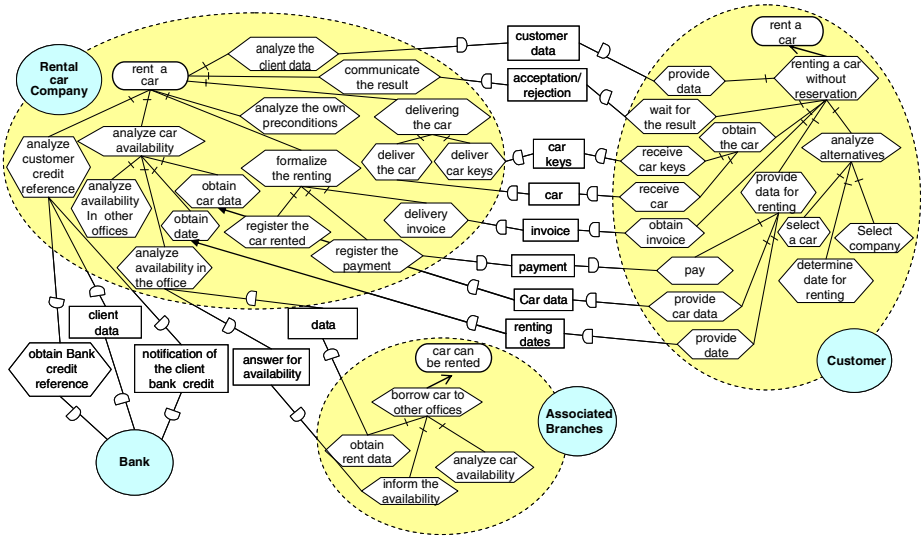

Fig. 6. Fragment of the car renting process in the Car Rental Management case study

6 Discussion

The main conclusion of this empirical evaluation is that *i** needs to be extended with mechanisms that manage granularity and refinement in models, as discussed below:

Granularity: Many of the negative results in the evaluation of *i** are related to the lack of mechanisms for defining granules of information at different abstraction levels, and composition mechanisms for composing these granules. This problem becomes evident when the modelling problem grows in size and complexity. In these cases, non-expert *i** users have difficulties with the scalability of their model. The result of this scenario is usually an overloaded monolithic model that contains all

relevant detail of a social and intentional setting. Any activity that tries to extend, analyze, adapt or reuse parts of such a model is bound to be complicated and error-prone. To avoid this problem, it is necessary to provide precise conceptual constructs representing building blocks that break the monolithic structure of *i** models, as well as composition mechanisms. Then, encapsulated model units could be created, analyzed and reused in an independent way. The practical implication of the granularity solution is the introduction of viewpoints that go beyond the actor viewpoint. For example, process viewpoints could give an orthogonal view for an organizational model. Note that for this extension, no modifications are needed to the original set of *i** modelling constructs.

Refinement: Apart from the definition of abstract primitives as building blocks, analysts must be provided with guidelines that allow them to structure a complete enterprise model. One way to achieve this consists of using concrete specification units to create the models following a refinement-based approach. In this way, the modelling process starts with a high-level view of the enterprise. Then, each element of this high-level view is refined into more concrete model. Viewpoint mechanisms are a very promising direction to help manage the complexity of modelling activities. A viewpoint on a system involves a perspective that focuses on specific concerns regarding the system, while suppressing irrelevant details [9]. A promising strategy towards this direction would be to guide the organizational modelling process using selected viewpoints. The refinement process enables us to join the advantages of social modelling with a compositional approach to create the organizational models in an incrementally way.

7 Conclusions

The *i** modelling framework is widely used for organizational modelling. The framework focuses on strategic relationships between actors in order to capture the social and intentional context of an enterprise. The main contribution of this paper consists of an empirical evaluation of *i**, using a feature-based evaluation framework and three industrial case studies. The evaluation has demonstrated that there is a set of issues that need to be addressed by the *i** modelling framework to ensure its successful application within industrial software development projects. These issues boil down to a lack of modularization mechanisms for creating and structuring organizational models.

We propose to extend *i** in order to address the weaknesses reported in this paper. Specifically, we are working on a solution for the problems of refinement, modularity, complexity management, reusability and scalability. Our solution is founded on the concept of a Business Service Architecture where organizational units can be encapsulated can only participate in actor dependency networks through well-defined interfaces. Along a different direction, we are developing a proposal to characterize *i** modelling primitives based on a multidimensional framework. This makes it possible to clearly differentiate the modelling primitives of *i**, so that modellers get better guidance on what primitives to use in different situations. With the proposed modifications, our intention is to overcome the current limitations that practitioners face when using i* in its current state. In fact, these modifications are intended to both solve the problems that were detected and to make the practical application of the

method easier. It certainly is necessary to evaluate whether these conclusions can be generalized in practice, and this is the direction of our current empirical work.

References

1. Bergenti, F., Gleizes., and Zambonelli, F. Methodologies and Software Engineering for Agent Systems. Kluwer Academic Publishing, 2004.
2. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., and Mylopoulos, J. TROPOS: an agent-oriented software development methodology. Journal of Autonomous Agents and Multiagent Systems, 8 (3): 203-236, July 2004.
3. Dam, K., and Winikoff, M. Comparing Agent-Oriented Methodologies. Proceedings of the Fifth International Bi-Conference Workshop on Agent-Oriented Information System (AOIS 2003), pages 78-93. Melbourne, Australia, July, 2003.
4. Liu, L., and Yu, E. Designing Information Systems in Social Context: A Goal and Scenario Modelling Approach. Information Systems Journal, 29(2): 87-203, 2003.
5. Mavin A., and Maiden N.A.M., 2003, Determining Socio-Technical Systems Requirements: Experiences with Generating and Walking Through Scenarios, Proceedings of the 11th International Conference on Requirements Engineering, pages 213-222, California, USA, September, 2003.
6. Padgham, L., Shehory, O., Sterling, L., and Sturm, A. "Methodologies for Agent-Oriented Software Engineering". Seventh European Agent System Summer School (EASSS 2005), Utrecht, the Netherlands, 2005.
7. Pastor, O., Gómez, J., Infrán, E., and Pelechado, V. The OO-Method approach for information systems modeling: from object-oriented conceptual modeling to automated programming. Information Systems, 26(7): 507-534, 2001.
8. Shehory, O., and Sturm, A. Evaluation of modeling techniques for agent-based systems. Proceedings of the Fifth International Conference on Autonomous Agents, pages 624-631. Montreal, Canada, May, 2001.
9. Sinan, S. Understanding the Model Driven Architecture (MDA). From http://home.comcast.net/~salhir/UnderstandingTheMDA.PDF. October, 2003.
10. Sturm, A., and Shehory, O. A Framework for Evaluating Agent-Oriented Methodologies, Proceedings of the Fifth International Bi-Conference Workshop on Agent-Oriented Information System (AOIS 2003). Melbourne, pages 94-109. Australia, July 2003.
11. Sturm, A., Dori, D., and Shehory, O. A Comparative Evaluation of Agent-Oriented Methodologies, to appear in Methodologies and Software Engineering for Agent Systems, Federico Bergenti, Marie-Pierre Gleizes, Franco Zambonelli (eds): Kluwer Academic Publishers.
12. Sudeikat, J., and Braubach, L., and Pokahr, A & Lamersdorf, W. "Evaluation of Agent-Oriented Software Methodologies Examination of the Gap between Modeling and Platform". Workshop on Agent-Oriented Software Engineering (AOSE-2004). New York, USA, pp. 126-141, July, 2004.
13. Yu, Eric. "Modelling Strategic Relationships for Process Reengineering". Published Doctoral dissertation, University of Toronto, Canada, 1995.