International Journal of Software Engineering
and Knowledge Engineering
Vol. 17, No. 1 (2007) 79–106

World Scientific
www.worldscientific.com

SYSTEMATIC CONSTRUCTION OF *i** STRATEGIC DEPENDENCY MODELS FOR SOCIO-TECHNICAL SYSTEMS

XAVIER FRANCH*, GEMMA GRAU†, ENRIC MAYOL‡, CARME QUER§,
CLAUDIA AYALA¶, CARLOS CARES‖, FREDY NAVARRETE**,
MARIELA HAYA†† and PERE BOTELLA‡‡

Universitat Politècnica de Catalunya-LSI, Campus Nord, Barcelona, Spain
*franch@lsi.upc.edu
†ggrau@lsi.upc.edu
‡mayol@lsi.upc.edu
§cquer@lsi.upc.edu
¶cayala@lsi.upc.edu
‖ccares@lsi.upc.edu
**fjnavarrete@lsi.upc.edu
††mhaya@lsi.upc.edu
‡‡botella@lsi.upc.edu
http://www.lsi.upc.es/∼webgessi/index.html

Goal- and agent-oriented models have become a consolidated type of artifact in various software and knowledge engineering activities. Several languages exist for representing such type of models but there is a lack of associated methodologies for guiding their construction up to the necessary level of detail. In this paper we present R*i*SD, a method for building Strategic Dependency (SD) models in the *i** notation. R*i*SD is defined in a prescriptive way to reduce uncertainness when constructing the model. R*i*SD tackles three fundamental issues: (1) it tends to reduce the average size of the resulting models; (2) it defines some traceability relationships among model elements; (3) it provides some lexical and syntactical conventions. As a result, we may say that R*i*SD supports the construction process of goal- and agent-oriented models whilst increasing their understanding.

Keywords: Goal-oriented modeling; agent-oriented modeling; agent-oriented methodologies.

1. Introduction

In the last few years, the use of *goals* for supporting a variety of processes and disciplines, such as business process reengineering, organizational impact analysis and requirements engineering, is increasingly gaining importance. Goals present several characteristics that make them attractive, e.g. expressiveness, stability and

*Corresponding author.

evolvability [1]. As a consequence of this tendency, the software engineering community is currently addressing the problems associated with the formulation of business goals, plans, processes, etc., in order to achieve organizational objectives in an ever-changing organizational environment [2].

From a methodological perspective, the intensive use of goals in these disciplines has resulted in many approaches that propose *goal-oriented modeling* in order to understand or describe problems associated with business structures, processes and their supporting systems [1, 3]. One of the most widespread goal-oriented modeling languages is the i^* notation proposed by Eric Yu in the first half of the 90s [4, 5]. i^* defines a repertory of intentional elements for building two types of models: the *Strategic Dependency* (SD) model and the *Strategic Rational* (SR) model, each one corresponding to a different abstraction level. Both models not only provide means for reasoning about goals but also about agents, therefore i^* is usually also considered an *agent-oriented modeling* language. In this paper, we are interested in both perspectives, goal-oriented and agent-oriented.

Since it was presented to the software engineering community, i^* has been used, and is being used, by a great deal of research teams with different purposes in the context of many projects, see [6] for a comprehensive view. There is also empirical evidence that i^* may be successfully used by practitioners in large-scale projects [7]. However, some weaknesses have been detected in other works. For instance in [8] noises, silences, contradictions and ambiguities of the i^* language have been identified. Also in [9] some drawbacks of the use of i^* are deduced from an empirical evaluation. Specifically, the authors report that modularity, reusability and scalability are not supported; and that refinement, repeatability, complex management and traceability are not well supported.

Some factors that are behind these weaknesses are the following:

Absence of prescriptive construction methods. There exists a consolidated method in the i^* framework, the Tropos method [10], aimed at supporting the conception of a global solution for the software development process, from early requirements engineering to implementation. But, on the other hand, there is currently a lack of guidance for supporting the prescriptive construction of i^* models [11]. One could argue that having a reasonable degree of freedom during model construction is precisely a property inherent to goal-oriented modeling [4], but the flexibility of the i^* model construction processes sometimes yields to different alternatives to choose among, whose selection is not always obvious.

Complexity of the models. Models for non-trivial systems grow very quickly and are plenty of intentional elements of many types which are not independent but interrelated [6]. Some types of relationships, like decomposition, synergy or means-end, can be represented in the language, especially in SR models, but other relationships are not modeled explicitly, and they need to be deduced from others or even that they cannot be stated at all. For instance, in the SD model, it is not possible to state that one dependency has been introduced to support another in a stepwise refinement process.

Lack of natural language conventions and standard terminology. Models built by different authors, or even the same, may vary in the way their intentional elements are written, both from the lexical and syntactical points of view [8]. Heterogeneity of styles, syntactical rules, and terminology, impacts not only on the understanding of the models, but also on their size and reuse.

With the purpose of solving these drawbacks, we aim at defining methods to support the construction of SD and SR models. Since both models are quite complex even considered separately, and also very different in nature, we have divided our research into two stages, and the present paper addresses the first one: we propose R*i*SD, a method for building Reduced *i** SD models for modeling organizational goals, and software systems for supporting them, in a guided and prescriptive manner. R*i*SD is defined as a set of activities structured in three phases, the first to analyze the domain of interest, the second for constructing the social system (without software) and the third for constructing the socio-technical system (with software). The second and third phases may involve partial or total construction of the other type of *i** models, namely SR models. R*i*SD includes precise questions and answers as well as patterns and catalogues that guide the development process and provides cut criteria for choosing among different types of intentional elements when diverse options exist. The size of the resulting model is reduced due to the nature of these criteria. R*i*SD also includes some traceability constructs that show the relationships among intentional elements. These intentional elements are written following a few syntactical rules, and some recommendations about vocabulary are given.

2. Related Work

Several authors have stated that the construction of goal- and agent-oriented conceptual models is necessary to build and develop agent-oriented systems, while recognizing that there is a lack of methods to support this activity. Luck *et al.* [12] declare this lack as an open issue on the existing development techniques for the specification of agent-oriented systems. In [13], Mao and Yu state that in order to develop agent-oriented applications successfully, methods should be designed to support system modeling in a robust, reliable and repeatable way. And Alonso *et al.* [14] claim that it is necessary to develop methods suited to the development process of agent-oriented systems.

In fact, there exist some goal-oriented methodologies based on the *i** language, but they do not guide model construction in much detail, merely define some phases or activities to be carried out in this process (see [11] for a comparative analysis). Absence of detailed guidance is one of the reasons behind the lack of repeatability in model construction mentioned in [9]. This is mostly due to the purpose of the model-to-be: usually *i** models are used to discover business processes, to elicit requirements, to explore possible variations in the organization behavior, etc. However, there are situations in which models need to be analyzed in more depth.

For instance, Franch and Maiden use them for assessing and comparing different architectural solutions in the context of Off-The-Shelf component selection [15]. In a similar manner, Kaiya *et al.* [16] and Franch [17] provide metrics for measuring certain properties of the model structure. In these and other cases, it is important to have concrete guidelines for building the model, which allow comparison of different models in a sound way, or providing a clear meaning to those metrics.

A representative example of the existing kinds of methods in the i^* framework is the Tropos method [10, 18], which currently is the most used framework in the context of i^*. Its main purpose is to define a comprehensive i^*-based agent-oriented software development method. Therefore, Tropos supports the whole software development cycle, from early requirements analysis to implementation, proposing an i^* model at each development stage. Furthermore, in [19] some transformations are proposed to refine an early requirements i^* model into an implementation i^* model. However, a look at these transformations reveals that they do not really guide the SD model construction process itself. We may say that Tropos is a coarse-grained i^* modeling method, which may benefit from the existence of a fine-grained one.

A remarkable exception to this state-of-the-art may be found in a particular line of research, namely the generation of UML models from i^* ones. For instance, Estrada *et al.* [20] provide guidelines for business model creation within the i^* framework which take into account the subsequent use case generation.

To sum up, the identified need of prescriptive i^* model construction methods has been the motivation to design our R*i*SD method. We will present the method in the next sections, but first we give a brief introduction on i^*.

3. The *i** Language

As mentioned above, the i^* language proposes the use of two models: a *SD model* that represents the intentional level of a system and *SR models* that represent its rational level (see Fig. 1 for example).

A SD model consists of a set of nodes that represent *actors* and a set of *dependencies* among them. Actors may be specialized through the *is-a* relationship. Dependencies express that an actor (*depender*) depends on some other (*dependee*) in order to obtain some objective (*dependum*). The dependum is an intentional element that can be a *goal*, a *task*, a *softgoal* or a *resource*. The depender depends on the dependee to bring about a certain state in the world in goal dependencies; to carry out an activity in task dependencies; to perform some task that meets a softgoal in softgoal dependencies; and for the availability of an entity in resource dependencies. The type of dependency also characterizes the actor of the dependency that has the responsibility to make decisions: in goal dependencies the dependee is expected to make any decisions required; in task dependencies the depender decides how the task will be performed by the dependee; in softgoal dependencies the depender makes the final decision, but does so with the dependee's know-how; in resource dependencies no decisions are required.

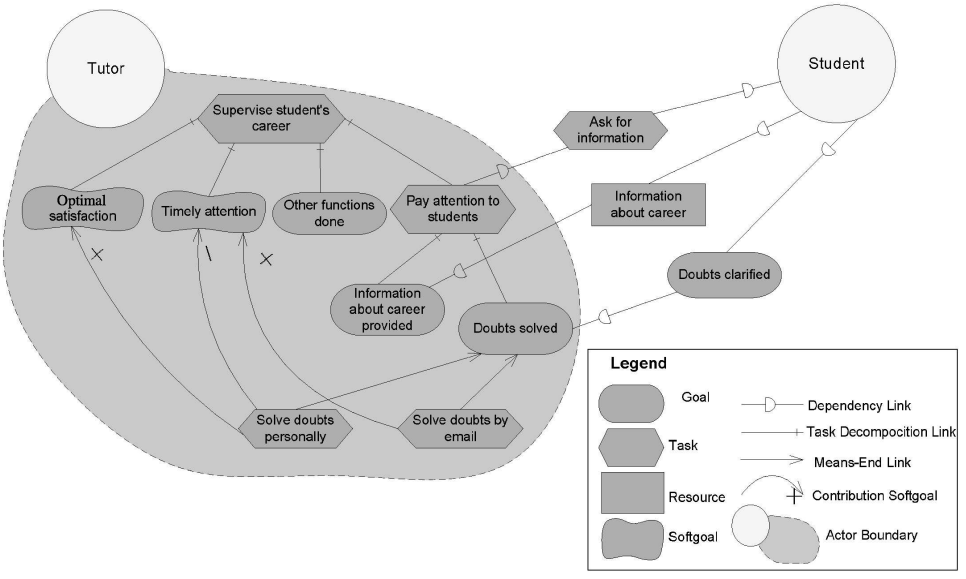

Fig. 1. Excerpt of an i^* model for an academic tutoring system.

A SR model allows visualizing the intentional elements into the *boundary* of an actor in order to refine the SD model to add reasoning ability. The dependencies of the SD model are linked to intentional elements inside the actor boundary. The elements in the SR model are decomposed accordingly to the following links:

— *Means-end* links establish that one or more intentional elements are the means that contribute to the achievement of an end. When there is more than one means, an OR relation is assumed, indicating the different ways to obtain the end.
— *Contributions to softgoals* correspond to means-end links where the end is a softgoal. An attribute stating the type of contribution (+, −) is required.
— *Task-decomposition* links state the decomposition of a task into different intentional elements. There is a relation AND when a task is decomposed.

3.1. *Actor decomposition*

Actors may have subparts that are established by the PART relationship [4, p. 100]. Actors and their parts are independent with respect to their intentionality and thus there can be dependencies among the whole and its parts. In the rest of this paper, we name subactors the actors that are PART of the decomposed actor.

For clarity purposes, in R*i*SD we use a variant in the graphical representation of the PART constructor. Specifically, we allow drawing the subparts of an actor inside its boundary. The equivalence among both graphical representations may be seen in Fig. 2.

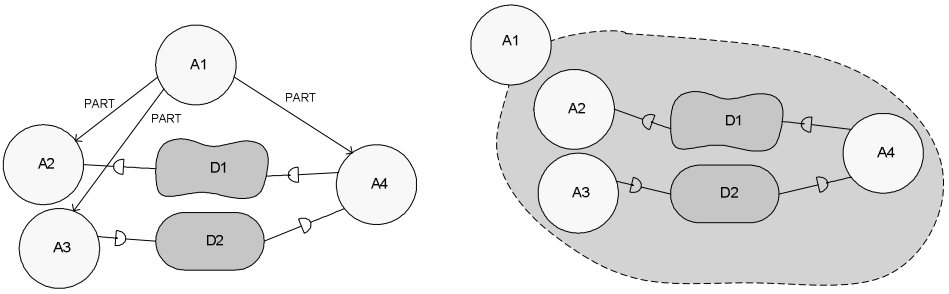

Fig. 2. PART constructor graphical representation: Yu's (left) and R*i*SD's (right).

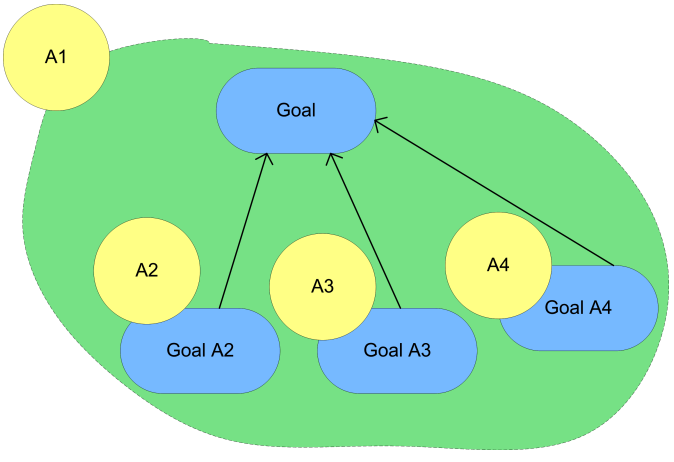

Fig. 3. PART constructor with dependencies among the whole and its parts.

In i^* the main goal of an actor is shown as the root of its SR model. However, we allow making explicit the main goal of actors in SD models, graphically representing it overlapped with the actor. Thus, we may express dependencies among actors and subactors as means-end links (as done in [4, p. 35]) among the main goal of the actor and the main goal of its subactors, as an SR model. In Fig. 3 we declare that to attain a goal of actor A1, it is necessary to attain the goals of its subactors A2, A3, A4.

3.2. *Traceability*

One of the drawbacks that have been mentioned in the introduction is the complexity of the models built in i^*. In particular, SD models for non-trivial systems end up with dozens of intentional elements and i^* does not provide structuring means for arranging them other than naming and drawing conventions. SR models, if delivered with the SD, may help in finding out relationships among SD elements,

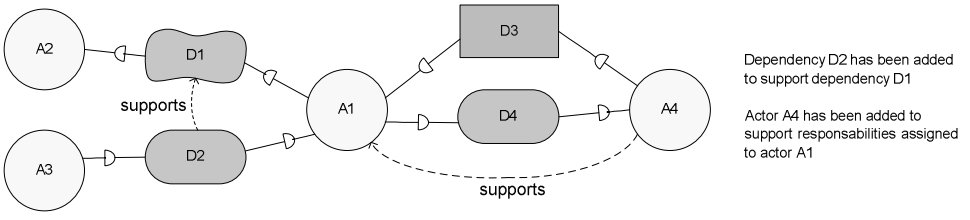

Fig. 4. The *supports* constructor.

but this is time-consuming. Therefore, one could argue that it would be helpful to have constructors for stating these relationships in the SD model itself. We present below two of these constructors that fit well in the stepwise model refinement strategy that characterizes our R*i*SD method: the *supports* and the *refines* constructors.

Stepwise model refinement makes new dependencies appear in order to facilitate the relationships stated by existing dependencies, as well as new actors to come up to attain the objectives of other actors already present in the model. We have defined a *supports* constructor to represent this relationship among dependencies and among actors, which allows keeping track of why new elements have been added to the i^* model. In the case of *supports* among actors with SR decomposition available, the relationship may be established among intentional elements belonging to these SR models. The *supports* relationship is graphically represented by an arrow from the new element to the existent one (see Fig. 4).

Stepwise refinement takes place not only in the context of a single model. Also, we may refine a whole model into another. In this case, most intentional elements of the new model exist because they refine intentional elements of the departing one. We have introduced the *refines* constructor to have traceability among the elements of the two i^* models. Refinement may be many-to-many, that is, one element in the new model may refine more than one element in the existing one, and also one element from the existing model may be refined into more than one element in the new one. Since addition of actors in the new model may change the responsibilities assigned to existing actors, refined dependencies of the new model may not have the same dependers and/or dependees than the ones of the original model. For simplicity of the drawing, refinement relationships are shown through the identifiers enclosed in parenthesis in the target model, which refer to elements of the original one (see Fig. 5).

It should be mentioned that both constructors have a rigorous definition. In [21], we have presented a metamodel for i^* and we have used it for defining several particular constructors proposed by different authors, among which we find our *supports* and *refines*. For more details about the *refines* constructor, [22] uses it intensively to implement the notion of matching among i^* models.

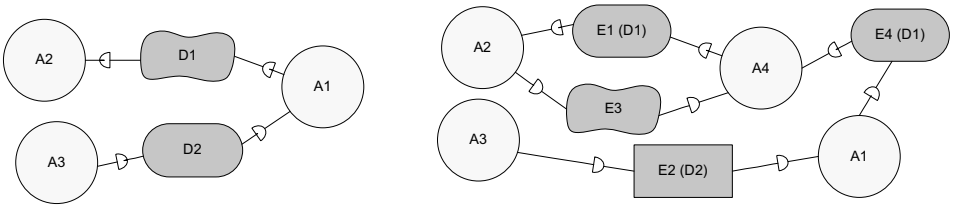

Fig. 5. The *refines* constructor: a departing SD model (left) and another SD model that refines it (right).

4. An Overview of R*i*SD

The R*i*SD method consists of three phases (see Fig. 6). Phase I deals with the analysis of the domain of the social system of interest. Phase II addresses the construction of an SD model that constitutes the social system model. This model is characterized by the fact that it does not include the software system and therefore it focuses on the stakeholder needs. In Phase III, the software system is incorporated to obtain a new SD model reconfigured around this new component. The SD model obtained at the end of this third phase is the socio-technical system model.

The three phases are conceived to be carried out one after the other; however, as it is the usual case in sequential processes, backward transitions are allowed to correct mistakes, add new knowledge, or improve models.

At Phase I the domain of interest is analyzed to obtain the necessary knowledge before starting the model construction activities. Domain analysis techniques may be applied and different artefacts may be built. At the end of the phase, the general objective of the system shall be identified.

The social system model is constructed iteratively. It begins with the identification of an initial set of social actors involved in the problem addressed and their main goals. Then, strategic dependencies among actors are identified and declared by default as goal dependencies. The next activity classifies definitively the added dependencies as goal, task, softgoal or resource. To assist in this classification, R*i*SD provides clear cut criteria by means of short, concise and focused questions. Also the name of the dependum may be slightly modified, in order to make it suitable to the type of dependency and to comply with vocabulary conventions. At this point, a first version of the social system model is obtained. If new actors or new dependencies have to be incorporated in the model, in order to support the existing elements, the process iterates.

The socio-technical system model construction is also iterative. It begins with the inclusion in the social system model of the software system as a new actor (with its main goal). Next, the existing dependencies are analyzed to decide if their depender or dependee must change to be the new software system actor. Afterwards, the system may be decomposed into subsystems (modeled as subactors of the software system actor using the PART constructor) and the existing dependencies

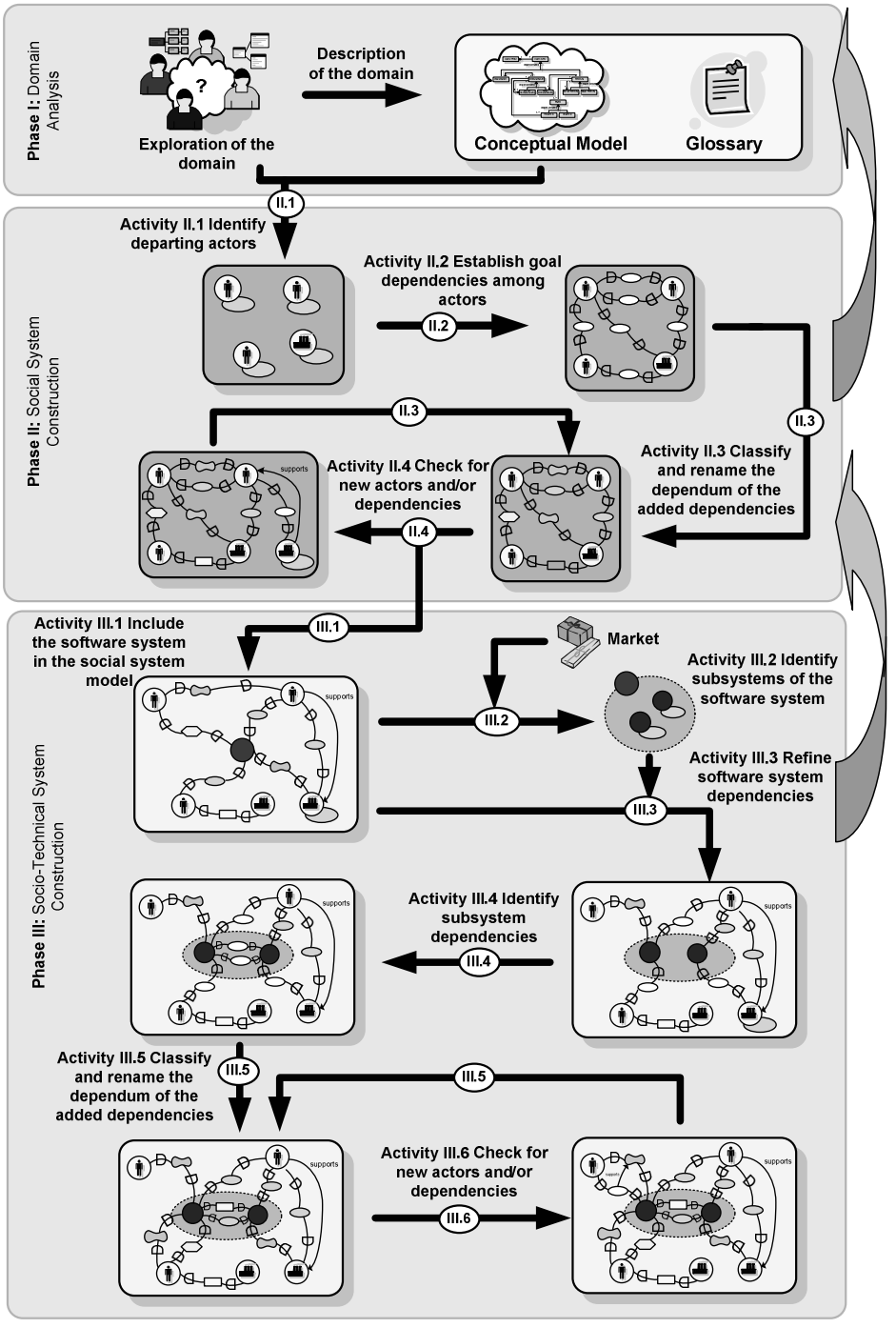

Fig. 6. The phases and activities of the R*i*SD method.

are refined into new ones that involve the subsystems. Probably other new dependencies will be necessary to state how subsystems depend on each other. Like in Phase II, first of all the added dependencies are goal dependencies and then they may be classified and renamed taking into account the suitable type of dependum. Finally, in the same way than in the social model, the process iterates in case that new actors or new dependencies have to be incorporated to the model to support the existing elements.

The paper includes an annex with a summary of Phases II and III.

Throughout the paper we use an example for describing the application of the different phases of the method, namely the specification of a software system for supporting information reliability in an organization.

5. Phase I: Domain Analysis

The first phase is carried out using domain analysis techniques [23]. There are many proposals available (e.g. [24–26]), each of them having its own characteristics and applicability conditions. In R*i*SD we have preferred not to commit to any of them and leave this point open.

Regardless of the technique, some knowledge may be expected to be acquired. Data conceptual models such as UML class diagrams are one of the most valuable types of artifacts for representing and understanding the different concepts involved in the domain. Also, since one of the most endangering points when examining a domain is the lack of a standard terminology, the construction of a glossary of terms may help to avoid semantic problems when constructing the model. Both artifacts may in fact be considered to be part of an ontology for the domain [27]. Building a comprehensive ontology, or just spare artifacts, or even only a glossary, will mainly depend on both the available knowledge of the domain and the effort scheduled to be invested. We do not include in this section any of these artifacts for space reasons.

Whilst the domain of interest is explored, the objective of the system becomes clearer. In our information reliability example introduced above, the objectives are: 1) to guarantee to the people who interact with the organization the reliability of the information that they submit and to preserve it for them; 2) to make the organization confident about the digital information it receives and stores.

6. Phase II: Social System Construction

Phase II is composed of 4 activities that are presented below.

6.1. *Activity II.1. Identify departing actors*

The goal of this activity is to discover the main actors of the social system and their goals. The actors are required to have a clear strategic value for the modeled system.

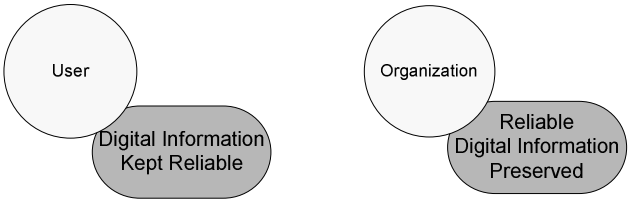

Fig. 7. Departing actors for the information reliability example.

There are two possible ways for doing this activity, the first one is just to deduce which are the actors and their goals from the result of the domain analysis and the second one is to recognize the problem as an instance of a known pattern. Two kinds of patterns are especially useful: *organizational patterns* [28] and the concept of *metaphor* that appears in several areas of software engineering (e.g., in agile development [29]).

In our case, we use a client-server metaphor: a user (the client) provides and consumes a resource (the information) that is under the control of an organization (the server). Thus, the departing actors and their goals are: the *User* with the goal *Digital Information Kept Reliable*, and the *Organization*, with the goal *Reliable Digital Information Preserved* as we illustrate in Fig. 7.

6.2. *Activity II.2. Establish goal dependencies among actors*

This activity aims at identifying the main dependencies among actors. By default, we classify them as goal dependencies, which are the most common type of dependencies found in social system models due to their strategic value. In activity II.3 the type of dependency will be confirmed or changed.

The crucial point of this activity is to identify just those dependencies that are really needed. In the case that no metaphor or pattern has been used, two questions may be asked to discover the dependencies (see the Annex) that model which kind of help one actor have from other actors to achieve its goal.

When using metaphors or patterns, the dependencies that are part of them can be adapted to the social system we are modeling, forming then the first set of dependencies of the model. This is easier if the metaphor or pattern is also expressed in i^*, as done in [28]. In our example, assuming that the metaphor is not modeled in i^*, we ask questions relatives to the client-server metaphor.

First, we shall respond to the question: *Which services does the user require from the organization for the social system providing some added value?* For each service, we include a dependency from the user to the organization. Then, we shall respond to another question: *Which behavior does the organization require the user for supporting* (*at least partially*) *the requested services?* If answers to these questions are not clear, we may build a first level of SR decomposition of some actor, which will show more explicitly which means can be undertaken by the actor itself and which others need the support of some other actor and therefore a dependency.

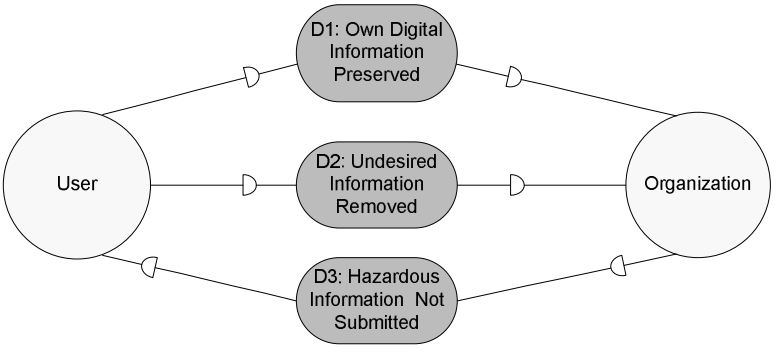

Fig. 8. Main dependencies in the information reliability case.

In our example, we identify two main services requested by the user, and one behavior that partially supports them (see Fig. 8). At this point, dependencies shall be generic enough in order not to exclude important aspects. For instance, if we were directly talking about viruses, other aspects such as filtering of spam could be incorrectly left out of the system.

6.3. *Activity II.3. Classify and rename the dependum of the added dependencies*

The goal of this activity is to classify the type of dependum (task, resource, softgoal or goal) of each added dependency and change the syntax of dependums in dependencies taking into account the type of dependum and the ontological information collected in Phase I.

We propose two complementary approaches to identify which is the most adequate type of dependum. Both approaches are based on answering a series of questions. In the first one, these questions rely on the own nature of the dependum. In the second approach, we put the emphasis on which are the responsibilities of the depender and the dependee with respect to the achievement of the dependum.

Applying one of them should be enough to classify a dependency. However, by applying both approaches, the classification process is made more accurate, since more intentional aspects of an i^* dependency models are considered. There is no specific order or condition to apply these approaches; for instance, the first approach may be applied to determine the dependum type, and this type may be corroborated by applying the second approach.

6.3.1. *Dependency classification under the dependum's nature point of view*

In this approach, to classify a dependency, we focus on the nature of its dependum. We propose a set of questions that shall be answered following a predefined ordering

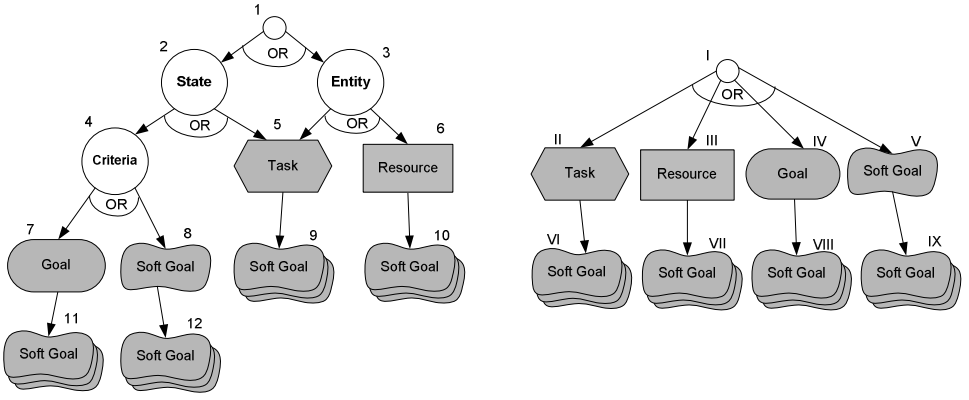

Fig. 9. Graphs to classify i^* dependencies: under dependum's nature (left), under responsibilities (right).

as shown in the graph of Fig. 9 (left). In nodes 1 to 4 a question must be answered; in nodes 5 to 8 a specific type of dependency has been identified; in nodes 9 to 12 some additional softgoal dependencies may be added to the model.

Starting at node 1, questions to answer at each node to classify the dependency D, from a depender A to a dependee B, are applied as stated below:

1. *Does the depender depend on the dependee to achieve an entity, or to attain a certain state?* If entity, go to 3; else, go to 2.
2. *Is the depender interested in attaining the state following a particular process?* If so, classify D as task dependency and go to 5; else, go to 4.
3. *Is the depender interested in obtaining the entity following a particular process?* If so, classify D as task dependency and go to 5; else, classify D as resource dependency and go to 6.
4. *Is there a clear cut criterion to determine the achievement of the state?* If so, confirm the dependency D as goal dependency and go to 7; else, classify D as softgoal dependency and go to 8.
5. *Are there some additional restrictions on how to execute the task?* If so, for each restriction, establish a new softgoal dependency from A to B.
6. *Are there some additional properties that the resource must met to be acceptable?* If so, for each property, establish a new softgoal dependency from A to B.
7. *Are there some additional conditions that the achievement of the goal must satisfy?* If so, for each condition, establish a new softgoal dependency from A to B.
8. *Are there some additional conditions necessary to consider the softgoal achieved?* If so, for each condition, establish a new softgoal dependency from A to B.

6.3.2. *Dependency classification under associated responsibilities point of view*

In this approach, to classify a dependency, we focus on the responsibilities of the depender and dependee in the achievement of the dependum. In this case (see graph of Fig. 9, right), given a dependency D, from a depender A to a dependee B, the questions to answer in each node are:

I. For this node one of the following applies:
   - *Is the depender the one that determines how the dependee must achieve the dependum and the dependee must achieve it in this way?* If so, classify D as a task dependency and go to II.
   - *Is the dependee only responsible for making the dependum available to the depender?* If so, classify D as a resource dependency and go to III. Notice that, in this case, there is no decision on how to make available the dependum.
   - *Is the dependee completely free to decide how to achieve the dependum without the participation of the depender?* If so, confirm D as a goal dependency and go to IV.
   - *Do the dependee and the depender decide together between different ways on how to achieve the dependum?* If so, classify D as a softgoal dependency and go to V.

II. *Is it necessary that the dependee and the depender define additional restrictions on how to execute the task?* If so, for each restriction, establish a new softgoal dependency from A to B.

III. *Is it necessary that the dependee and the depender define additional properties for the resource to be obtained?* If so, for each property, establish a new softgoal dependency from A to B.

IV. *Is it necessary that the dependee and the depender define additional conditions for the goal to be attained?* If so, for each condition, establish a new softgoal dependency from A to B.

V. *Is it necessary that the dependee and the depender define additional conditions to agree on the achievement of the softgoal?* If so, for each condition, establish a new softgoal dependency from A to B.

By applying this second approach, we should obtain the same result as applying the first one, or at least a very similar one. Notice that nodes from 5 to 12 of Fig. 9, left, agree with nodes from II to IX of Fig. 9, right.

In our example, regardless of the approach followed to classify the three departing dependencies (Fig. 8), we leave them as goal dependencies, since all of them correspond to states and their achievement. Moreover, the dependee is completely free to decide how to achieve them and there are no additional conditions for the achievement of each goal. Thus, Fig. 8 becomes also the result of activity II.3.

Table 1. Syntactic conventions for i^* dependums.

Dependum	Syntax	Example
Task	Verb + (Object) + (Complement)	Answer doubts by e-mail
Resource	(Adjective) + Object + (Qualifier/Modifier)	Virus List
Goal	Object + Passive Verb + (non-manner Complement), possibly negated	Information kept preserved
Softgoal	Goal syntax + Complement of manner (Object) + Complement of manner ([Dependum])	Information checked transparently Timely [Virus List]

6.3.3. *Dependum renaming*

Once dependencies are classified, it may be necessary to rename them if the current writing does not comply with the conventions agreed. Names assigned to dependums shall be kept short and precise (longer descriptions can be added to the documentation, especially if using tool support such as OME [30] or REDEPEND [31]) and be consistent throughout the model. Our naming conventions cover two dimensions:

- Syntactic rules. Each type of dependum shall be compliant with the syntactic rules that are considered to highlight their nature. There are some conventions issued by different authors, and we adhere to that of Yu [4], summarized in Table 1 (parentheses stand for optionality). We note the case of softgoal dependencies, in which we distinguish among dependencies that stand alone (node 8 in the graph of Fig. 9, left, or node V in the graph of Fig. 9, right), whose pattern is *Goal Syntax + Complement of manner*; and dependencies that qualify another dependum of the model (nodes 9 to 12 of Fig. 9, left, or nodes V to IX in the graph of Fig. 9, right), in which the qualifier is a *Complement* and optionally the dependum between brackets (as done in [4]). Note that using these syntactical patterns we will use short names that are specific to the semantics of the dependum, increasing in this way the understandability of the model.
- Vocabulary. For determining the lexicon to be used, we use mainly two types of information: cross-domain standards and documents, and the artifacts obtained during Phase I. For instance, if available, we may use data conceptual models for giving name to resources dependums, or activity names from activity diagrams to give name to task dependums. A remarkable source of cross-domain document is the ISO/IEC 9126-1 standard which defines 27 quality criteria such as time efficiency, security or integrity; we have extended this quality model [32] and the extended catalogue is our base for giving names to softgoal dependums (e.g., timely, transparent, etc.).

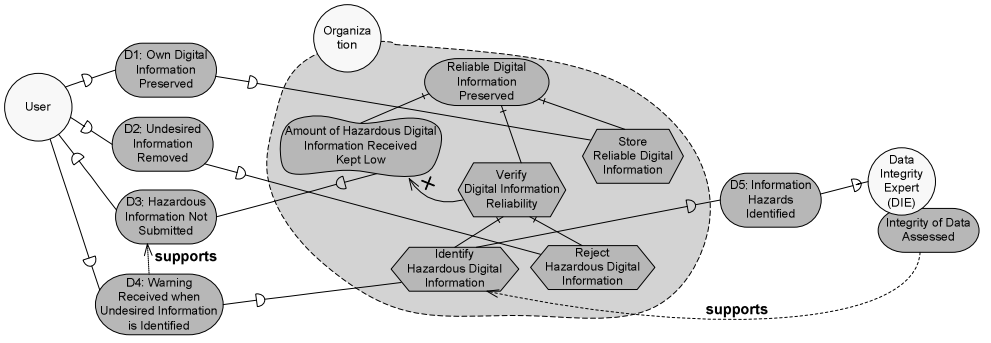

Fig. 10. First refinement in the information security case.

6.4. *Activity II.4. Check for new actors and/or dependencies*

This activity consists of checking the need for introducing new actors or dependencies to support the actors that participate as dependees in some dependency.

The identification is done by checking, for every dependency added in the preceding iteration (either in activity II.2 or activity II.4), if either the dependee is able to satisfy by itself the required dependum or if it needs some help from other actor that may already exist, or not yet (in the last case, its goal must be declared first, as done in activity II.1). For deciding this, a question is raised whose concrete form depends on the type of dependum: *Does the depender need some support to attain the goal, or produce the resource, or execute the task, or accomplish the behavior?* If the answer to this question is not obvious, it may be necessary to develop one or two levels of decomposition of the SR diagram of the dependee actor taking as root an intentional element equivalent to the involved dependum. When new elements are included to the model, a "supports" relationship (see Sec. 2.2) is defined among the new element and the one that is the reason for its inclusion.

In Fig. 10 we may see the identification of two new elements. On the one hand, the dependency *D4*: *Warning Received when Undesired Information Identified* among the user and the organization appears when we ask the question: *Does the user need some support to attain the goal implied by dependency D3?*. On the other hand, the actor *Data Integrity Expert* (DIE) is determined in order to give support to the task *Identify Hazardous Digital Information* that has to be done by the organization to accomplish its objective. This task has been identified when building the SR model of the *Organization* actor. We have reassigned all the depender and dependees that involved the *Organization* to the intentional elements that appear in its SR model.

In the case that no new elements (actors or dependencies) are identified, or the only elements identified in the last iteration are resource and task dependencies, the phase is finished and the resulting model is the social system model. Refinement of task and resource dependencies is usually too prescriptive at the social system

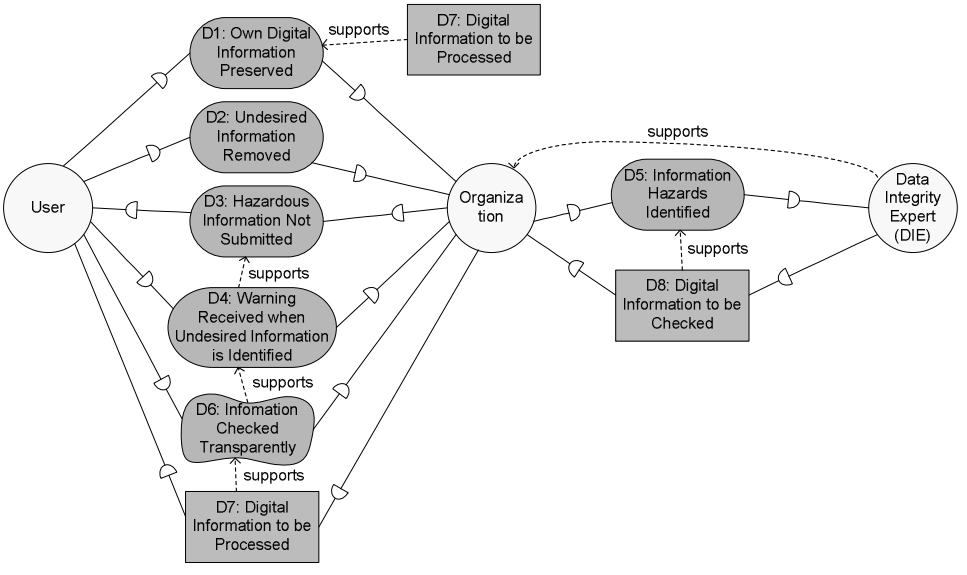

Fig. 11. Final social system in the information security case (D7 has been duplicated for clarity of the diagram).

level, since it just would identify steps of the tasks, or components of the resource. If new elements have been introduced, it is necessary to iterate on activities II.3 and II.4.

The social system model in the example Fig. 11 shows the resulting social system model of the example. We needed just 2 more iterations of activities II.3 and II.4. The final model consists of 3 actors (User, Organization and DIE) and 8 dependencies (D1 to D8), with 4 supporting relationships among them. The last dependencies added were resources. The added softgoal dependency incorporates a fundamental behavior that shall be observed in the system. This model provides a highly strategic view of the social system, ready to be reconsidered once the software system is incorporated.

7. Phase III: Socio-Technical System Construction

In this section we describe the activities to construct the socio-technical system model, starting from the social system model. The annex includes the guides and questions to be used in each activity of this phase.

7.1. *Activity III.1. Include the software system in the social system model*

The first activity defines a new actor for the software system, states its high-level goal and reassigns the existing dependencies as needed. To define the goal of the

new actor we answer the following question: *Which is the purpose of the software system?* In some cases the answer can be stated simply by thinking in the general pursued objective. In our example, since this objective is to ensure information reliability in an organization, the goal is *Reliable Digital Information Managed*.

Dependency reassignment can be decided by answering the question: *May the software system provide any assistance on attaining the goal/producing the resource/executing the task/achieving the property of the dependency?* It is very likely that more than one answer is possible, meaning that there is more than one way to assign responsibilities, in which case we have different alternatives to be analyzed. In [33] we have carried out an analysis of different situations on how to reassign the existing dependencies of a business i^* model when a new actor is introduced to the model. Basically, the patterns proposed in [33] offer two possibilities: to maintain the same depender and dependee of a dependency, if the depender and dependee responsibilities remain in the same social actors; or to delegate the depender/dependee responsibility (completely or in part) to the new actor, which means automating the responsibility modeled by the dependency. In the second case, a first level of construction of the SR models of the software system and the social actor is recommended, to determine how responsibilities are reassigned, and thus, if this reassignment requires new dependencies.

In Fig. 12 we show the assignment of responsibilities to the software system in our example. In this case, we take the profit of the SR diagram that we constructed for the *Organization* actor in previous activities (see Fig. 10) and we reallocate most of the existing *Organization* intentional elements to the new software actor. Also, a new dependency is added in order to state how the organization depends on the software actor.

7.2. *Activity III.2. Identify subsystems of the software system*

Usually software systems are too large or complex to be modeled by just one monolithic actor. In these cases, the software system may be split into several subsystems, each of them with a well defined goal. The combination of all the subsystem goals must correspond to the goal of the whole system. We can use i^* actor decomposition facilities to reflect the new subsystems in the model. Two different strategies may be used:

- *Market-driven*: Add new subsystems taking into account the knowledge about the software packages marketplace. The question in this case could be: *Which types of software packages apply to the problem on hand?*
- *Dependency-driven*: Looking at each dependency on the software system actor, decompose the main goal of the software system into subgoals and assign them to new subsystem actors. For each dependency, we must answer the following question: "does the dependency give rise to more specific goals for the software system?"

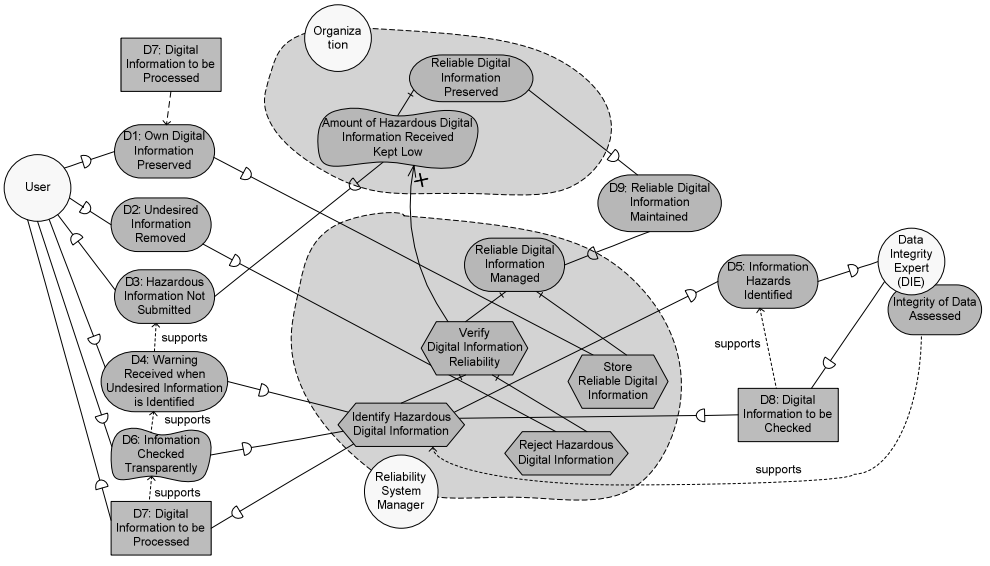

Fig. 12. Putting the software system into the social system.

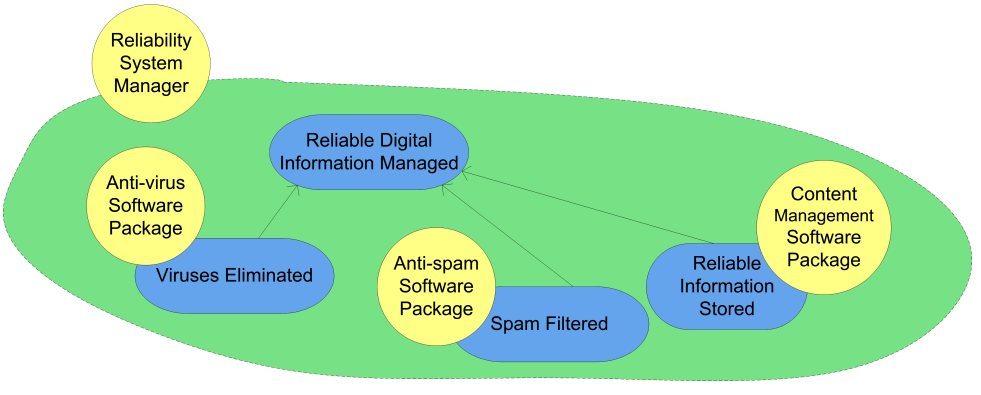

Fig. 13. Splitting the software system into subsystems (excerpt).

In our case we take a market-driven approach. The questions may be based on the objective and the tasks that exist in the SR diagram of the software system (see Fig. 12): *Which types of software packages apply to verify digital information reliability?* and *Which types of software packages apply to store reliable digital information?* An excerpt of the result of this activity is shown in Fig. 13.

7.3. *Activity III.3. Refine software system dependencies*

In this activity it is necessary to refine the dependencies that involve the software system by substituting them by new dependencies that take into account the

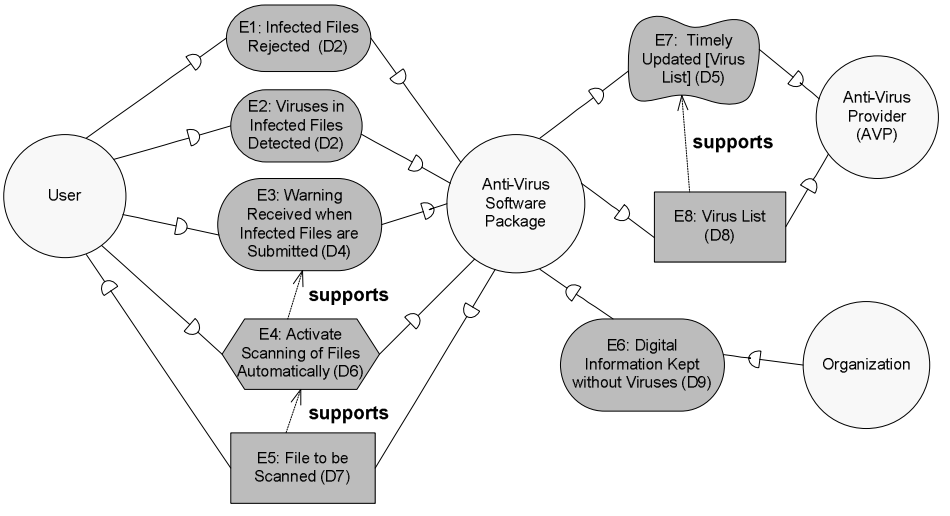

Fig. 14. From software system to subsystem dependencies.

decomposition into subsystems. For each dependency and subsystem, we first ask if "Does the dependency involve the subsystem?" The answer is straightforward if activity III.2 has followed the dependency-driven strategy. If the answer is yes, then a second question applies: "How does the subsystem interpret the concepts involved by the dependency?"

Let us consider the anti-virus subsystem. In this context, the correspondence of key concepts is: the hazard is the virus, the information that flows among actors is a file, and reliable interchange means detecting (and removing whenever possible) viruses from the file. The resulting model is shown in Fig. 14. Note that dependencies D1 and D3 are not refined since they do not have anything to do with the Anti-Virus Software Package.

7.4. *Activity III.4. Identify subsystems dependencies*

If the software system has been decomposed into several subsystems, it is very likely that they depend on each other to achieve their goal. In particular, we will usually find that one subsystem may provide services needed by others. In this activity the aim is to identify these dependencies. Therefore for each subsystem the following question is raised: *Which services does the subsystem require of each other subsystem in order to attain its goal and in order to provide the services required from its external actors?*

In our example, one of the types of spam are messages containing viruses, thus we may establish a goal dependency among the anti-virus and the anti-spam subsystems (see Fig. 15).

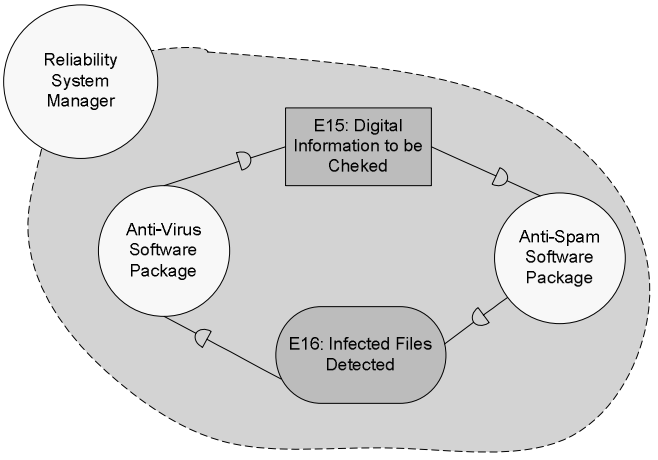

Fig. 15. Establishing subsystem dependencies.

7.5. *Activities III.5 and III.6. Classify and rename the dependum of the added dependencies and check for new actors and/or new dependencies*

These activities are analogous to activities II.3 and II.4 (therefore they have the same name). Thus, their objective is also to deal with the dependencies added in the previous activities and to add new dependencies and/or actors if it is necessary. The guides and questions that have been enumerated in activities II.3 and II.4 apply here in the same way, and also both activities are iterated until no new relevant dependencies are found. When this occurs the final socio-technical system model has been built.

7.6. *The socio-technical system model in the example*

In Fig. 16 we show an excerpt of the socio-technical system model, just considering the anti-virus part. Two new actors have been identified in activity III.6: the *Anti-Virus Administrator* and the general concept of *Other Software Package* that may also act as anti-virus user, and therefore it has been defined as a specialization of the user. This part of the system is composed by 6 actors and 14 dependencies, which is a reasonable size for a concrete facet of information management as reliability is.

8. Conclusions

Building i^* models in the context of large-scale software systems development is not an easy task. To overcome the inherent complexity of this activity, we need methodological support guiding the incremental construction of those models by stepwise refinement. This has been the goal of our R*i*SD method for building i^* Strategic

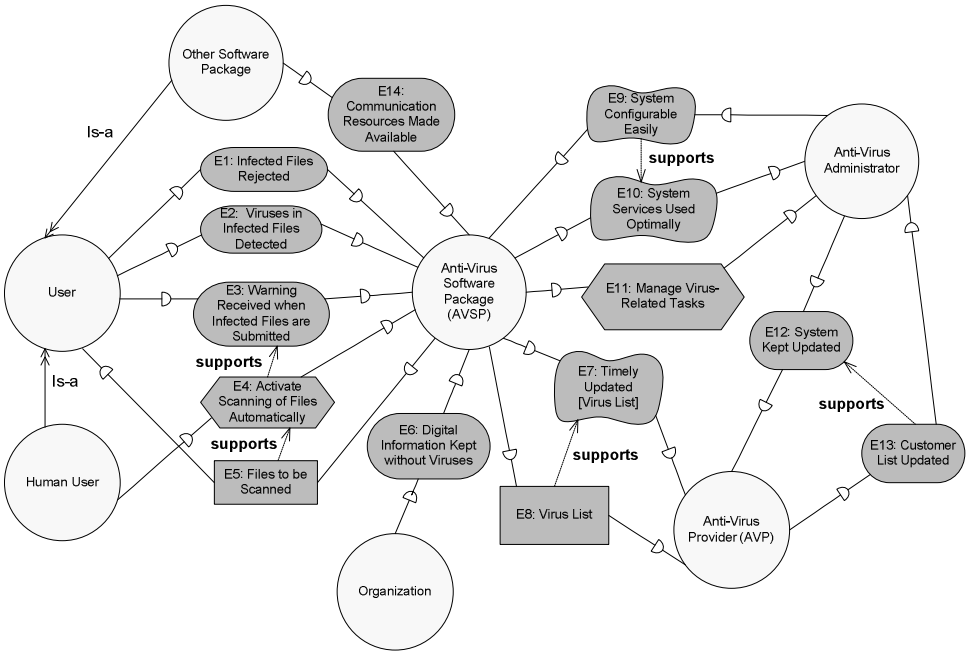

Fig. 16. Excerpt of the resulting socio-technical system model for the information security case.

Dependency (SD) models. The formulation of a similar method for Strategic Rationale (SR) models is part of our future work.

The most relevant contributions of R*i*SD are in relation to the three drawbacks that we identified in Sec. 1:

Absence of prescriptive construction methods. R*i*SD provides prescriptive guidance reducing therefore the subjectivity that is inherent in goal-oriented modeling. The activities that are defined in the two model construction phases are supported by rules, criteria, questions and patterns. Remarkably, we have given accurate hints for identifying and classifying dependencies. On the other hand, iteration and intertwining are recognized in the method making it flexible.

Complexity of the models. As a result, we have obtained models that are more easily analyzed since there is a well-defined and consistent rationale behind. We have also incorporated traceability with two new constructs, "supports" and "refines". In addition, we have recognized the need of having clear syntactic conventions to be used consistently. Due to the nature of R*i*SD, the resulting models are kept as small as possible, trying to cope with one of the most important problems in the use of i^*, namely scalability of the models.

Lack of natural language conventions and standard terminology. We have adopted one of the various proposed syntactic conventions for the writing of i^* elements and clarified some points. For the particular case of softgoals, we have proposed a widespread quality catalogue to provide a common vocabulary. For

the case of tasks and resources, we have proposed to build some artifacts during the domain analysis preliminary which provide a system-dependant vocabulary of terms. Altogether we are providing some guidelines that foster the homogeneity of SD models.

This work has been done in parallel with the construction of a metamodel for i^* which defines in a rigorous way each single intentional element that conforms to the language [21]. The combination of the metamodel and the method has proven its value in several cases we have developed, especially if considered in relation to some of the evaluation criteria mentioned in [9]:

- *Refinement*: The iterative nature of our method supports refinement because more and more refined models are obtained progressively.
- *Repeatability*: We think that this is one of our main contributions, since we are providing a highly prescriptive methodological support to the model construction process. Remarkably, we are providing rationale for identifying actors and dependencies, and for classifying the dependencies into the four i^* types.
- *Complexity management*: Very related to refinement, we are providing not only iterative processes, but distinguishing the social and the socio-technical system explicitly.
- *Traceability*: We have introduced two constructs, "supports" and dependency refinement, which keep track of the relationships among intentional elements as the model becomes more detailed.
- *Reusability*: We have explicitly considered the existence of organizational patterns and metaphors, although we have not exploited this issue in this paper.

Concerning validation, we have moved along two directions. First, we have assessed the method by applying it to some reported exemplars (meeting scheduler [4, 34], e-media shop [18], and others). Second, we have built several models for new cases that we have needed in other experiences, mainly related to the analysis of domains in the context of software quality analysis. In both cases, we have observed several expected benefits, some mentioned above but also others, e.g. novices on i^* acquire modeling skills faster and inter-group communication has become more agile and unambiguous. We are currently preparing a more exhaustive long-term validation plan following the guidelines provided by Wohlin *et al.* [35]. We have determined two different populations to work with in different experiments. On the one hand, undergraduate students without any experience on i^* and, on the other hand, researchers with interest or even experience on i^*; for this second population, the recently created collaborative wiki on i^* (http://istar.rwth-aachen.de) will be crucial.

As a final remark, we think that our method may contribute also to other agent-oriented development methods like Tropos [10], Gaia [36], Prometheus [37], MESSAGE [38], and others, more precisely on their model construction step. Specifically, in the case of Tropos, we think that the prescriptivism of R*i*SD (questions and guides) could help in the progress among diagrams corresponding

to the different phases covered by the method. Further investigation on how R*i*SD may fit in these methods and how it must be adapted to their modeling languages is part of our future work.

Acknowledgments

This work has been done in the framework of the research project UPIC, Ref. TIN2004-07461-C02-01, supported by the Spanish Ministerio de Ciencia y Tecnología. Some authors have grants that partially support their work: C. Ayala, by the Mexican Council of Science and Technology (CONACyT) and the Catalan government Generalitat de Catalunya; C. Cares, by the MECE-SUP FRO0105 Project of the Chilean government; G. Grau, by a UPC research scholarship; F. Navarrete, by an IGSOC scholarship.

References

1. A. van Lamsweerde, Goal-oriented requirements engineering: A guided tour, in *Proc. Int. Symp. on Requirements Engineering* (*RE'01*), Toronto, Canada, 2001, pp. 249–263.
2. P. Loucopoulos and V. Karakostas, *Systems Requirements Engineering* (McGraw-Hill, 1995).
3. E. Yu and J. Mylopoulos, Understanding *why* in software process modeling, analysis, and design, in *Proc. Int. Conf. on Software Engineering* (*ICSE'94*), Italy, 1994, pp. 159–168.
4. E. Yu, *Modeling Strategic Relationships for Process Reengineering*, PhD. thesis, University of Toronto, 1995.
5. E. Yu, Towards modeling and reasoning support for early-phase requirements engineering, in *Proc. Int. Symp. on Requirements Engineering* (*RE'97*), Washington, USA, 1997, pp. 226–235.
6. P. Giorgini, N. Maiden, J. Mylopoulos, and E. Yu (eds.), *Social Modeling for Requirements Engineering* (MIT Press, 2007), in press.
7. N. Maiden, S. Jones, C. Ncube and J. Lockerbie, *Using i^* in Requirements Projects: Some Experiences and Lessons*, book chapter in [6], 2007.
8. C. Ayala, C. Cares, J. P. Carvallo, G. Grau, M. Haya, G. Salazar, X. Franch, E. Mayol, and C. Quer, A comparative study of i^*-based goal-oriented modeling languages, in *Proc. 17th Int. Conf. on Software Engineering and Knowledge Engineering* (*SEKE'05*), Taipei, Taiwan, 2005, pp. 43–50.
9. H. Estrada, A. Martínez, O. Pastor, and J. Mylopoulos, An experimental evaluation of the i^* framework in a model-based software generation environment, in *Proc. Int. Conf. on Advanced Information Systems Engineering* (*CAiSE'06*), Luxembourg, 2006, pp. 513–527.
10. A. Fuxman, L. Liu, J. Mylopoulos, M. Pistore, M. Roveri, and P. Traverso, Specifying and analyzing early requirements in Tropos, *Requirements Engineering Journal* **9**(2) (2004) 132–150.
11. G. Grau, C. Cares, X. Franch and F. J. Navarrete, A comparative analysis of i^* agent-oriented modeling techniques, in *Proc. 18th Int. Conf. on Software Engineering and Knowledge Engineering* (*SEKE'06*), San Francisco Bay, USA, 2006, pp. 657–663.

12. M. Luck, P. McBurney and C. Preist, A manifesto for agent technology: Towards next generation computing, *Autonomous Agents and Multiagent Systems* **9**(3) (2004) 203–252.
13. X. J. Mao and E. Yu, Organizational and social concepts in agent oriented software engineering, in *Proc. Agent-Oriented Software Engineering* (*AOSE'05*), Utrecht, Netherlands, 2005, pp. 1–15.
14. F. Alonso, S. Frutos, L. Martínez and C. Montes, Towards a natural agent paradigm development methodology, in *Proc. Multi-Agent System Technologies* (*MATES'04*), Erfurt, Germany, 2004, pp. 155–168.
15. X. Franch and N. Maiden, Modeling component dependencies to inform their selection, in *Proc. Second Int. Conf. on COTS-Based Software Systems* (*ICCBSS'03*), Ottawa, Canada, 2003, pp. 81–91.
16. H. Kaiya, H. Horai and M. Saeki, AGORA: Attributed goal-oriented requirements analysis method, in *Proc. Int. Conf. on Requirements Engineering* (*RE'02*), Essen, Germany, 2002, pp. 13–22.
17. X. Franch, On the quantitative analysis of agent-oriented models, in *Proc. Int. Conf. on Advanced Information Systems Engineering* (*CAiSE'06*), Luxembourg, 2006, pp. 495–509.
18. J. Castro, M. Kolp and J. Mylopoulos, A requirements-driven development methodology, in *Proc. Int. Conf. on Advanced Information Systems Engineering* (*CAiSE'01*), Interlaken, Switzerland, 2001, pp. 108–117.
19. P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia and J. Mylopoulos, Modeling early requirements in Tropos: A transformation based approach, in *Proc. Agent-Oriented Software Engineering* (*AOSE'01*), Montreal, Canada, 2001, pp. 151–168.
20. H. Estrada, A. Martínez and O. Pastor, Goal-based business modeling oriented towards late requirements generation, in *Proc. Int. Conf. on Conceptual Modeling* (*ER'03*), Chicago, USA, 2003, pp. 277–290.
21. C. Cares, X. Franch, E. Mayol and C. Quer, *A Reference Model for i**, book chapter in [6], 2007.
22. X. Franch, On the lightweight use of goal-oriented models for software package selection, in *Proc. Int. Conf. on Advanced Information Systems Engineering* (*CAiSE'05*), Porto, Portugal, 2005, pp. 551–566.
23. G. Arango, Domain analysis: From art form to engineering discipline, in *Proc. Int. Workshop on Software Specification and Design* (*IWSSD'89*), Pennsylvania, USA, 1989, pp. 152–159.
24. K. Kang, S. Cohen, J. Hess, W. Novak and A. Peterson, *Feature-Oriented Domain Analysis* (*FODA*) *Feasibility Study*, CMU/SEI-90-TR-021, 1990.
25. I. Reihartz and A. Sturm, Behavior domain analysis — The application-based domain modeling approach, in *Proc. <<UML>> Conference* (*UML'04*), Lisbon, Portugal, 2004, pp. 410–424.
26. C. Ayala and X. Franch, Domain analysis for supporting commercial off-the-shelf components selection, in *Proc. Int. Conf. on Conceptual Modeling* (*ER2006*), Tucson, AZ, 2006, pp. 354–370.
27. M. Uschold and M. Gruninger, Ontologies: Principles, methods and applications, *Knowledge Engineering Review* **11**(2) (1996) 93–155.
28. M. Kolp, P. Giorgini and J. Mylopoulos, Organizational patterns for early requirements analysis, in *Proc. Int. Conf. on Advanced Information Systems Engineering* (*CAiSE'03*), Klagenfurt/Velden, Austria, 2003, pp. 617–632.
29. K. Beck, *Extreme Programming Explained* (Addison-Wesley, 1999).
30. OME3 page, http://www.cs.toronto.edu/km/ome, last accessed January 2007.

31. N. Maiden, P. Pavan, A. Gizikis, O. Clause, H. Kim and X. Zhu, Integrating decision-making techniques into requirements engineering, in *Proc. Int. Workshop on Requirements Engineering: Foundation for Software Quality* (*REFSQ'02*), Essen, Germany, 2002.
32. J. P. Carvallo, X. Franch, G. Grau and C. Quer, Reaching an agreement on COTS quality through the use of quality models, in *Proc. Second Workshop on Software Quality* (*WoSQ'04*), Edinburgh, 2004, pp. 18–23.
33. G. Grau, X. Franch and N. Maiden, A goal-based round-trip method for system development, in *Proc. Int. Workshop on Requirements Engineering: Foundations for Software Quality* (*REFSQ'05*), Porto, Portugal, 2005, pp. 71–86.
34. E. Yu and J. Mylopoulos, An actor dependency model of organizational work: With application to business process reengineering, in *Proc. Conf. on Organizational Computing Systems* (*COOCS'93*), Milpitas, CA, 1993, pp. 258–268.
35. C. Wohlin, P. Runeson, M. Host, M. C. Ohlsson, B. Regnell and A. Wesslen, *Experimentation in Software Engineering: An Introduction* (Springer, 1999).
36. M. Wooldridge, N. R. Jennings and D. Kinny, The Gaia methodology for agent-oriented analysis and design, in *Proc. Autonomous Agents and Multi-Agent Systems* **3**(3) (2000) 285–312.
37. L. Padgham and M. Winikoff, Prometheus: A pragmatic methodology for engineering intelligent agents, in *Proc. Workshop on Agent-Oriented Methodologies* (*OOPSLA'02*), Seattle, WA, 2002, pp. 97–108.
38. G. Caire, W. Coulier, F. Garijo, J. Gomez, J. Pavón, F. Leal, P. Chainho, P. E. Kearney, J. Stark, R. Evans and P. Massonet, Agent oriented analysis using message/UML, in *Proc. Agent Oriented Software Engineering* (*AOSE'01*), Montreal, Canada, 2001, pp. 119–135.

Annex

Table A.1. Social system construction.

Phase II Social System Construction			
Activity	**Description**	**Questions and Guides**	
II.1 Identify departing actors	Discover the main actors and their goals	Option 1. An appropriate organizational pattern or metaphor is found – (Pattern) Give problem-related names to the pattern actors and their main goals – (Metaphor) Ask questions related to the selected metaphor.	Option 2. Neither organizational patterns nor metaphors are found. – Which is the main goal of each actor? – Which are the actors of the social system?
II.2 Establish goal dependencies among actors	Identify the main dependencies among actors in the social system	– (Pattern) Give problem-related names to the pattern dependencies – (Dependencies) Ask questions relative to the selected metaphor	– Which services does every actor require from each other in order to provide some added value to the social system? – Which behavior do the actors that provide some service require from other actors for supporting (at least partially) the requested service?
II.3 Classify and rename the dependum of the added dependencies	Classify the type of dependum of each added dependency *D* (with a depender *A* and a dependee *B*)	Option 1. Classify under dependum's nature point of view 1. Does the depender depend on the dependee to achieve an entity or to attain a certain state? If entity, go to 3; else, go to 2. 2. Is the depender interested in attaining the state following a particular process? If so, classify *D* as task dependency and go to 5; else, go to 4. 3. Is the depender interested in obtaining the entity following a particular process? If so, classify *D* as task dependency and go to 5; else, classify *D* as resource dependency and go to 6. 4. Is there a clear cut criterion to determine the achievement of the state? If so, confirm the dependency *D* as goal dependency and go to 7; else, classify *D* as softgoal dependency. Are there some additional: 5. restrictions on how to execute the task? 6. properties that the resource must met to be acceptable? 7. conditions that the achievement of the goal must satisfy? 8. conditions necessary to consider the softgoal achieved ? For each restriction, property, condition: add a softgoal dependency from *A* to *B*	Option 2. Classify under associated responsibilities point of view I. Is the depender the one that determine how the dependee must achieve the dependum and the dependee must achieve it in this way? If so, classify *D* as a task dependency and go to II. I. Is the dependee completely free to decide how to achieve the dependum without the participation of the depender? If so, confirm *D* as a goal dependency and go to III. I. Is the dependee responsible only to make the dependum available to the depender? If so, classify *D* as resource dependency and go to IV. I. Do the dependee and the depender decide together between different ways on how to achieve the dependum? If so, classify *D* as a softgoal dependency. Is it necessary that the depender and the dependee define additional: II. restrictions on how to execute the task? III. properties for the resource to be obtained? IV. conditions for the goal to be attained? V. conditions to agree on the achievement of the softgoal? For each restriction, condition, property: add a softgoal dependency from *A* to *B*
	Rename dependums taking into account their type	– In task dependencies use Verb + (Object) + (Complement) if activity diagrams available, use activity names – In resource dependencies use (Adjective) + Object + (Qualifier / Modifier) if data conceptual model available, use class and attribute names – In goal dependencies use Object + Passive Verb + (Complement) complement is not of manner; may be negated – In softgoal dependencies use Goal syntax + Complement or (Object) + Complement([dependum]) complement is of manner; use our extended ISO catalogue	
II.4 Check for new actors and/or dependencies	Identify new actors and/or dependencies to support dependees in the added dependencies	– For task dependencies: does the dependee need some support (actor and/or dependency) to execute the task? – For resource dependencies: does the dependee need some support (actor and/or dependency) to produce the resource? – For goal dependencies: does the dependee need some (actor and/or dependency) support to attain the goal? – For softgoal dependencies: does the dependee need some support (actor and/or dependency) to accomplish the behavior?	

Table A.2. Socio-technical system construction.

Phase III Socio-Technical System Construction			
Activity	**Description**	**Questions and Guides**	
III.1 Include the software system in the social system model	Define a new actor corresponding to the software system, state its high level goal	– Which is the purpose of the software system?	
	Reassign the existent dependencies as needed	For each dependency in the social system model: – May the software system provides any assistance on: · attaining the goal of the dependency? · producing the resource of the dependency? · executing the task of the dependency? · achieving the property of the dependency?	
III.2 Identify subsystems of the software system	Split the software system into several subsystems, each of them with a well defined goal, that decompose the main goal of the system	Option 1. Dependency-driven strategy. Looking at each dependency on the software system actor, decompose the main goal of the software system into subgoals and assign them to new subsystem actors. – Does the dependency give rise to more specific goals for the software system?	Option 2. Market-driven strategy. Add new subsystem taking into account the knowledge of the market of software packages. – Which types of software packages apply to the problem at hand?
III.3 Refine software system dependencies	Refine the dependencies that involve the software system by substituting them by new dependencies that take into account the division in subsystems.	For each dependency and subsystem: – Does the dependency involve the subsystem? – For dependencies that involve a subsystem: How does the subsystem interpret the concepts involved in the dependency?	
III.4 Identify subsystem dependencies	Identify the dependencies that are necessary among the subsystems	For each subsystem: – Which services does a subsystem require of each other subsystem in order to attain its goals and in order to provide the services required from external actors?	
III.5 Classify and rename the dependum of the added dependencies	Classify the type of dependum of each added dependency *D* (with a depender *A* and a dependee *B*)	See questions and guides of Activity II.3	
	Rename dependums taking into account their type		
III.6 Check for new actors and/or dependencies	Check the need of new actors or dependencies to support actors that participate as dependees in each added dependency	See questions and guides of Activity II.4	