Co-evolution of complementary formal and informal requirements

Aneesh Krishna, Aditya K. Ghose
Decision Systems Laboratory
School of IT and Computer Science
University of Wollongong, Australia
ak86, aditya@uow.edu.au

Sergiy A. Vilkomir
Software Quality Research Laboratory
Dept of Computer Sc. & Info. Systems
University of Limerick, Ireland
sergiy.vilkomir@ul.ie

Abstract

Agent-oriented Conceptual Modelling (AoCM, as exemplified by the i* *notation [5], represents an interesting approach to modelling early phase requirements that is particularly effective in capturing organizational contexts, stakeholder intentions and rationale. There are significant benefits in using formal methods for the development of computer systems and improving their quality. We propose a methodology which permits the use of these two otherwise disparate approaches in a complementary and synergistic fashion for requirements engineering.*

1. Introduction

Many modelling techniques tend to address "late-phase" requirements while the vast majority of critical modelling decisions (such as determining the main goals of the system, how the stakeholders depend from each other, and what alternatives exist [5]) are taken in early-phase requirements engineering. The *i** modelling framework [5] is a semi-formal notation built on agent-oriented conceptual modeling that is well-suited for answering these questions. The central concept in *i** is that of the intentional actor or agent. The actor or agent construct is used to identify intentional characteristics represented as dependencies involving goals to be achieved, tasks to be performed, resources to be furnished or softgoals (optimisation objectives or preferences) to be satisfied.

The *i** framework consists of two graphical modelling components: Strategic Dependency (SD) Models and Strategic Rationale (SR) Models. The SD model captures the social context of the system. It consists of a set of nodes and links where each node represents an actor, and each link between the two actors denotes a *dependency*. An actor may depend on another to achieve a*goal*, perform a *task*, provide a *resource* or achieve a *softgoal* (each of these represents a distinct category of dependency). Consider the following example (to be used throughout the rest of the paper) from our earlier case study [7] which concentrates on a key function of the emergency services agency (ESA): managing flood rescue and evacuation operations (this research has been conducted in the context of a larger project to deploy *i** for enterprise modelling in a large ESA). The SD model in Figure 2 (due to limited space, we have also included details of agent internals - these usually appear in separate SR models) involves an *Emergency Coordination Centre Coordinator (ECCC)* agent which depends on *Field Control Centre Coordinator (FCCC)* agents to accomplish its goal to *Rescue People At Risk*. The *ECCC* has a dependency on the *Weather Bureau* to provide *Weather Data*, modelled as a resource dependency (the other dependencies can be explained along similar lines, but are omitted here for brevity).

An SR model (shown together with an SD diagram in Figure 2) provides a more detailed level of modelling by looking "inside" actors to model internal intentional elements such as goals, tasks, resources, and softgoals which appear in an SR model not only as external dependencies, but also as internal elements linked by task-decomposition and means-ends relationships. For example, *Volunteers/Emergency Workers* has an internal task to *Rescue People* which involves the subtasks *Prepare For Rescue*, *Map Reading and Navigation*, *Operate Rescue Boats*, *Communication Equipment Operation*, *Supply Essentials*, *Rescue/Evacuate People at Risk* and the goal *Report Situation* (modelling this as a goal instead of a task suggests that several alternative ways of achieving the goal exist and no commitment has been made to any single one of these). An SR model thus provides a means for modelling stakeholder interests, how they might be met, and the stakeholders' evaluation of various alternatives with respect to their interests.

A number of proposals have been made for combining *i** modelling with late-phase requirements analysis and the downstream stages of the software life-cycle. The TRO-

Proceedings of the 7th International Workshop on Principles of Software Evolution (IWPSE'04)
1550-4077/04 $ 20.00 IEEE

POS project [1] uses the *i** notation to represent early- and late-phase requirements, architectures and detailed designs. However, the *i** notation in itself is not expressive enough to represent late-phase requirements, architectures and designs. To address this problem, a custom-designed formal languages called FormalTropos [2] has been proposed. Proposals to integrate *i** with formal agent programming languages have also been reported in the literature [4]. This paper has similar objectives, but takes a somewhat different approach. We believe that the value of conceptual modeling in the *i** framework lies in its use as a notation *complementary* to existing specification languages, i.e., the expressive power of *i** complements that of existing notations. The use of *i** in this fashion requires that we define methodologies that support the *co-evolution* of *i** models with more traditional specifications. We use the notion of *co-evolution* in a very specific sense to describe a class of methodologies that permit *i** modeling to proceed independently of specification in a distinct notation, while maintaining some modicum of *loose coupling* via *consistency constraints*. In the current instance, we examine how this might be done with formal specification notations, but such an exercise is of value in the context of a variety of other notations (such as UML). Our aim, then, is to support the modeling of organizational contexts, intentions and rationale in *i**, while traditional specifications of functionality and design proceeds in the formal notation. In this paper, we focus on Z [3] as a prototypical representative of a formal notation, but observe that many of the lessons generalize to other formal methods. More generally, this research suggests how diagrammatic notations for modeling early-phase requirements, organization contexts and rationale can be used in a complementary manner with more traditional specification notations.

2. *i** to Z Transformation

A first step in defining a co-evolution methodology for *i** and Z is to define a mapping from *i** to Z. We summarize below some results from our earlier work [6] where this mapping was initially defined. Some of our examples will be drawn from a detailed case study of the application of this mapping [7]. Considerable detail has been omitted in this section due to space limitations, but examples and full versions of the schemas described below can be found at www.uow.edu.au/ aditya/research/iz.html

The sets of all actor names, all_actors, and dependency names, all_depend, are defined as power sets of the set $NAME$. Free types $STATE$ (which can be any one of *inapplicable, unresolved, fulfilled, violated, satisficed, denied* or *undetermined*), $TYPE$ (either *goal, softgoal, task, resource* or *ISA*), $DEGREE$ (either *open, committed* or *critical*) and $LINK\_TYPE$ (any one of *task-decomp, means-ends, contrib* or *not applicable*) describe the possible states, types and degrees of dependencies and the types of links between the internal intentional elements respectively. The notion of $STATE$ is implicit in *i**, but requires explication in Z specifications. The state of a SD model is the set of states of all its dependencies. The state of an actor is given by the set of states of all its internal (SR) elements (i.e., goals, tasks etc.).

SD

$$SD\_state : NAME \pfun STATE$$

$$\mathrm{dom}\ SD\_state = all\_depend$$

Actor

$$actor\_name : NAME$$
$$actor\_element : \mathbb{P}_1\ NAME$$
$$actor\_state : NAME \pfun STATE$$

$$actor\_name \in all\_actors$$
$$\mathrm{dom}\ actor\_state = actor\_element$$

As a common pattern for SD dependencies and SR elements, the schema $\Phi Depend$ is used (the Φ in the schema name is used to flag a partial specification [3]). We use these to define an $SDependency$ as an operational schema (which defines how a dependency might change state, e.g. from *unresolved* to *fulfilled*) and which also describes the structure of the dependency (the two actors involved, the object of the dependency etc.). Details are omitted for brevity but can be found in the full version of this paper.

Links between internal actor elements as described in an SR model (task decomposition, means-ends, softgoal contribution) are represented using the first of the following two schemas. The second schema describes the structure of actor internal elements such as tasks, goals, softgoals etc.

Link

$$\Phi Depend$$
$$int\_components, ext\_components : \mathbb{P}\ NAME$$
$$contrib\_p, contrib\_m : \mathbb{P}\ NAME$$
$$link : LINK\_TYPE$$

$$link = task\_decomp \Rightarrow type = task$$
$$link = contrib \Rightarrow type = softgoal$$
$$contrib\_p \cup contrib\_m \neq \varnothing \Rightarrow link = contrib \land \langle contrib\_p, contrib\_m \rangle \text{ partitions } int\_components$$
$$ext\_components \neq \varnothing \Rightarrow link = task\_decomp$$
$$link = NA \Leftrightarrow int\_components \cup ext\_components = \varnothing$$

AElement

$$\Delta Actor$$
$$Link$$

$$dependum \in actor\_element$$
$$int\_components \subset actor\_element$$
$$ext\_components \subseteq all\_depend$$
$$actor\_name' = actor\_name$$
$$actor\_element' = actor\_element$$
$$actor\_state' = actor\_state \oplus \{dependum \mapsto result!\}$$

We shall refer to these basic schemas as *model schemas*. Schemas for actors, dependencies, actor internal elements and the links between them in a specific *i** model are defined using these model schemas - we shall call these *element schemas*. The mapping process that we have described so far leads to a Z specification that captures the structure represented in an *i** model (and in the instance of states, obliges the analyst to represent some additional information as well). A key subsequent step is the *refinement* of these essentially structural schemas with additional information (i.e. information not included in an *i** model, but obtained via further analysis). In [7] and [6], we have provided detailed examples of such refinement (e.g., temporal sequencing of dependencies, fulfilment conditions for dependencies etc.). We shall refer to the Z specification obtained after these refinements as the *Extended Z Specification*.

3. Methodology supporting the co-evolution of *i** and Z

The focus of our work in this paper is on defining a methodology that permits the maintenance of the *loose coupling* between an *i** model and a Z specification (refer Figure 1). Our strategy is to *localize* the impact of changes. We do this at two specific points. First, we define techniques for reflecting changes in an *i** model in the corresponding (unrefined) Z specification (i.e., the Z model obtained by directly applying the mapping techniques discussed in the previous section to the prior *i** model). Second, we define techniques for reflecting the refinements contained in the prior extended Z specification to obtain a new extended Z specification (i.e., one which contains all of the prior refinements, while reflecting the changes in the corresponding *i** model). We note that changes in the *i** model only affect the element schemas, but not the model schemas.

Let us consider the first of these two questions: obtaining an unrefined Z specification from the modified *i** model. We define techniques for achieving this that require reference to the prior *i** model and the corresponding prior unrefined Z specification. We note that sixteen categories of possible changes may occur to an *i** model. These are the *addition* and *deletion*, respectively, of the following eight elements: *Dependencies, Tasks, Goals, Resources, Softgoals, Means-end links, Task-decomposition links and Actors*. We shall consider each of these cases in turn.

Addition/deletion of a dependency to an existing SD model: i) Addition leads to the creation of an additional *element schema* for the new dependency (deletion leads to the removal of this schema). *ii*) The internal intentional elements as represented in the SR models for the pair of actors involved in the dependency may need to be modified, since all the external dependencies are connected to some internal element of an actor. This change is localized to the following simple step: we add (or delete) the dependency name from the $ext\_components$ set in the corresponding element schema for the relevant internal element.

Addition/deletion of a task to an existing SR model: i) Addition will result in the creation of a new element schema for the task (deletion leads to its removal). A newly added task is typically related via a means-ends link to a goal, or via a task decomposition link to another task. Potentially, it may also be related via a softgoal contribution link to an existing softgoal. Schemas for these links must then also be added along the lines described below. *ii*) The element schemas for the goals, tasks and softgoals that this new task might be linked to (as discussed above) need to be modified by adding (resp. deleting) the name of the task to the $int\_components$ set of the corresponding schema(s). *iii*) The name of the task must be added (resp. deleted) to the $actor\_element$ set in the element schema for the corresponding actor. *iv*) The name of the task must be added (resp. deleted) as the value of the $depender\_internal\_element$ variable in the schema for any dependency related to the task (should such a relationship be established after the task is added) in which the corresponding actor (into whose SR model the task has been added) is the depender. In a similar fashion, the name of the task is added as the value of the $dependee\_internal\_element$ variable in the schema of any dependency related to the task in which the corresponding actor is the dependee. *v*) A downstream effect of the addition of a task in an SR model followed by the creation of a new dependency connecting this task to an internal element in another actor is that the steps outlined for the addition (resp. deletion) of a dependency (outlined above) have to be followed.

Addition/deletion of a goal/resource/softgoal to an existing SR model: We follow steps similar to those described above for the addition/deletion of tasks.

Addition/deletion of a means-ends link to an existing SR model: Means-ends links (as with task decomposition links) are not represented via separate schemas, but via the schemas of the internal (SR) elements that they relate. A means-ends link offers alternative means for achieving a given goal (we shall refer to this as the *end*). In other word, it is effectively the analogue of an OR node in an AND-OR goal graph. The addition of a means-ends link results in the value of the $link$ variable in the element schema for the end being assigned the value *means-ends* and the $int\_components$ set in the same schema being defined as the collection of the internal SR elements (which could be tasks, goals or resources) related to the end via the means-ends link. Deletion results in these values being removed.

Addition/deletion of a task decomposition link to an existing SR model: A task decomposition link functions as the

analogue of an AND node in an AND-OR goal graph and provides a singly, unique means of decomposing a task (we shall refer to this as the *parent task*) into a collection of subtasks, subgoals, resources etc. The addition of a task decomposition link results in the following changes to the element schema for the parent task: the $link$ variable is assigned the value *task-decomposition* while the $int\_components$ set is defined as the collection of subtasks, subgoals etc. related to the parent task by this link. Deletion results in these values being removed.

Addition of an actor to an existing i diagram will lead to following four steps:* A new element schema for the actor is created. In the instance of each internal (SR) element for the actor, the steps outlined above are followed. The same applies for any dependencies that this actor might participate in.

We shall now discuss the second area where we are able to localize the impact of changes: the generation of a new extended Z specification given the new set of Z schemas (corresponding to the modified *i** model) and the prior extended (refined) Z specification. Our aim is to reflect the refinements in prior set of Z schemas (that led to the prior extended Z specification) in the new collection of Z schemas, without having to re-do the refinements. This is a relatively simple affair. We identify the set of Z schemas in the prior collection of (unrefined) Z schemas (obtained from the prior *i** model) that were refined in some fashion. We identify schemas with the same names (if they exist, since some might have been deleted) in the current collection of (unrefined) Z schemas (obtained from the revised *i** model), and apply the same refinements to these. This gives us the new extended Z specification.

We shall now present few illustrations to explain the methodology supporting the co-evolution of *i** and Z. These examples are based on the managing flood rescue and evacuation case study. Following modifications/additions were performed on the initial *i** diagrams:

Introducing a resource dependency *Simplified Weather Data* between *Volunteers/ Emergency Workers* and *FCCC* will lead to the modification of the original *i** diagram and creation of an additional element Z schema (external dependency). The concerned actors/agents *Volunteers/Emergency Workers and FCCC* internal intentional elements Z schema(s) (which is affected) are going to be modified because of this action (since all the external dependencies are connected to some internal element of an actor somewhere in the SR diagram). In this case the internal intentional elements in *Volunteers/Emergency Workers* and *FCCC* are *Rescue People* and *Asses Weather Situation* respectively.

$SimplifiedData$

$SDependency$
$dependum = simplified\_weather\_data$
$depender = worker$
$dependee = field\_coordinator$
$type = resource$
$degree = committed$

The newly added resource dependency *Simplified Weather Data*'s, Z schema is further refined with additional information derived from the *i** models (refinement)-this is known as Extended Z model. Our observation is that this extended Z schema is not going to affect any other previous extended Z schema. We can directly perform some minor modifications in the predicate part of the newly created Z schema of the resource dependency (as basis) to arrive at this extended Z schema. For example, dependency $analysed\_weather\_forecast$ should be realized before dependency $simplified\_weather\_data$. For this, it is necessary to include into the predicate part of $SimplifiedData$ schema the following precondition: $SD\_state(simplified\_weather\_data) = fulfilled \Rightarrow SD\_state(analysed\_weather\_forecast) = fulfilled$.

$SimplifiedData1$

$SDependency$
$dependum = simplified\_weather\_data$
$depender = worker$
$dependee = field\_coordinator$
$type = resource$
$degree = committed$
$simplified\_weather\_data \subseteq analysed\_forecast$
$SD\_state(simplified\_weather\_data) = fulfilled$ $\Rightarrow SD\_state(analysed\_weather\_forecast)$ $= fulfilled$

Based on the second guideline provided under addition of dependency (of our Co-evolution methodology), the affected actors/agents internal intentional elements Z schema (which is directly connected to the dependency in question - in this case *Rescue People* and *Asses Weather Situation* respectively) is also going to be modified (since all the external dependencies are connected to some internal element of an actor somewhere in the SR diagram). Minor modification is performed on the predicate part of the concerned internal intentional element Z schema(s). The revised Z schemas of internal intentional elements *Rescue People* and *Asses Weather Situation* are going to have *Simplified Weather Data* as additional entry under the ext_components listing in respective Z schemas. The revised Z schemas of internal intentional elements are provided as ready reference:

AssesWeatherSituation

AElement
field_coordinator

dependum = *asses_weather_situation*
type = *task*
degree = *committed*
int_components = ∅
ext_components = {*analysed_weather_forecast*
simplified_weather_data}
link = *NA*

RescuePeople

AElement
Worker

dependum = *rescue_people*
type = *task*
degree = *committed*
operate_rescue_boat, *commn_equip_operation*
map_reading, *prepare_rescue*, *rescue_people*
fast_efficient}
ext_components = {*evacuation_mission*
quick_response, *simplified_weather_data*}
link = *task_decomp*

The rest of the mapped Z schemas remain unchanged for the modified *i** model.

We note that a reverse mapping from a collection of Z schemas to an *i** model is possible provided the following assumptions hold. First, the Z schemas were obtained from an initial *i** model via mapping and refinement along the lines described above. Second, the prior *i** model is available for reference. Finally, the integrity of the element schemas must be maintained throughout the refinement process, i.e., refinement steps may add to but not modify existing element schemas. Given these assumptions it is relatively simple to identify the named element schemas in a Z specification and thus reconstruct the corresponding *i** model without loss of information (any refinements made will, of course, not be reflected in the *i** model).

4. Conclusion

We present a relatively simple methodology to support the complementary use of an early-phase requirements modeling notation such as *i** with formal specifications, in this instance Z. We have not investigated the possibility of articulating semantic consistency constraints between *i** models (possibly augmented with FormalTropos annotations) and formal specifications. This is the focus of our future work. We have already shown in our earlier work how two otherwise disparate approaches (Agent-oriented Conceptual Modelling and Formal methods) might be used in a complementary and synergistic fashion.

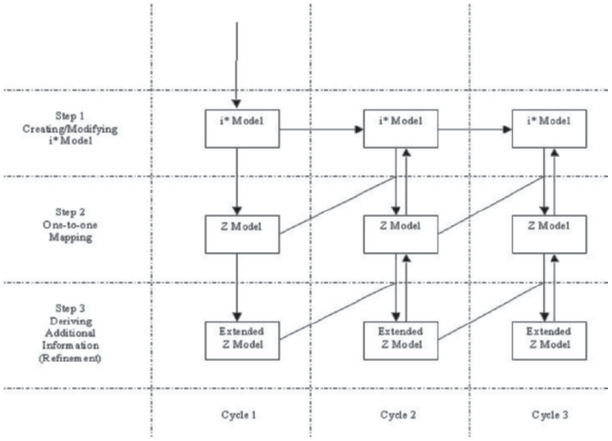

Figure 1. Co-evolution of i* models and Z specifications

References

[1] J. Castro, M. Kolp and J. Mylopoulos. Towards Requirements-Driven Information Systems Engineering: The Tropos Project. To appear in *Information Systems*, Elsevier, Amsterdam, The Netherlands, 2002.

[2] Fuxman, A., Pistore, M., Mylopoulos, J., Traverso, P. Model checking early requirements specifications in Tropos. *Proceedings of Fifth IEEE International Symposium on Requirements Engineering*, Toronto, Canada, August 27-31, 2001, pp. 174 -181.

[3] Spivey, J. M. *The Z Notation: A Reference Manual.* Prentice Hall International Series in Computer Science, 2nd edition, 1992.

[4] Wang, X., Lesprance, Y. Agent-Oriented Requirements Engineering Using ConGolog and *i**. *Proceedings of 3rd International Bi-Conference Workshop Agent-Oriented Information Systems (AOIS-2001)*, Berlin, Germany, 2001, pp. 59-78.

[5] Yu, E. Modelling Strategic Relationships for Process Reengineering. *PhD Thesis, Graduate Department of Computer Science, University of Toronto*, Toronto, Canada, 1995, pp. 124.

[6] Vilkomir, S., Ghose, A.K., Krishna, A. Combining agent-oriented conceptual modeling with formal methods. *Proceedings of ASWEC-2004: The 2004 Australian Software Engineering Conference*, Melbourne, Australia, April, 2004.

[7] Krishna, A., Vilkomir, S., Ghose, A.K. A case study of combining *i** framework and the Z notation. *Proceedings of ICEIS-2004: The 6th International Conference on Enterprise Information Systems*, Porto - Portugal, April, 2004.

[8] Unni, A., Krishna, A., Ghose, A. K., Hyland, P. Practical early phase requirements engineering via agent-oriented conceptual modelling. *Proceedings of ACIS-2003: The 2003 Australasian Conference on Information Systems*, Perth, Australia, November 26-28, 2003.

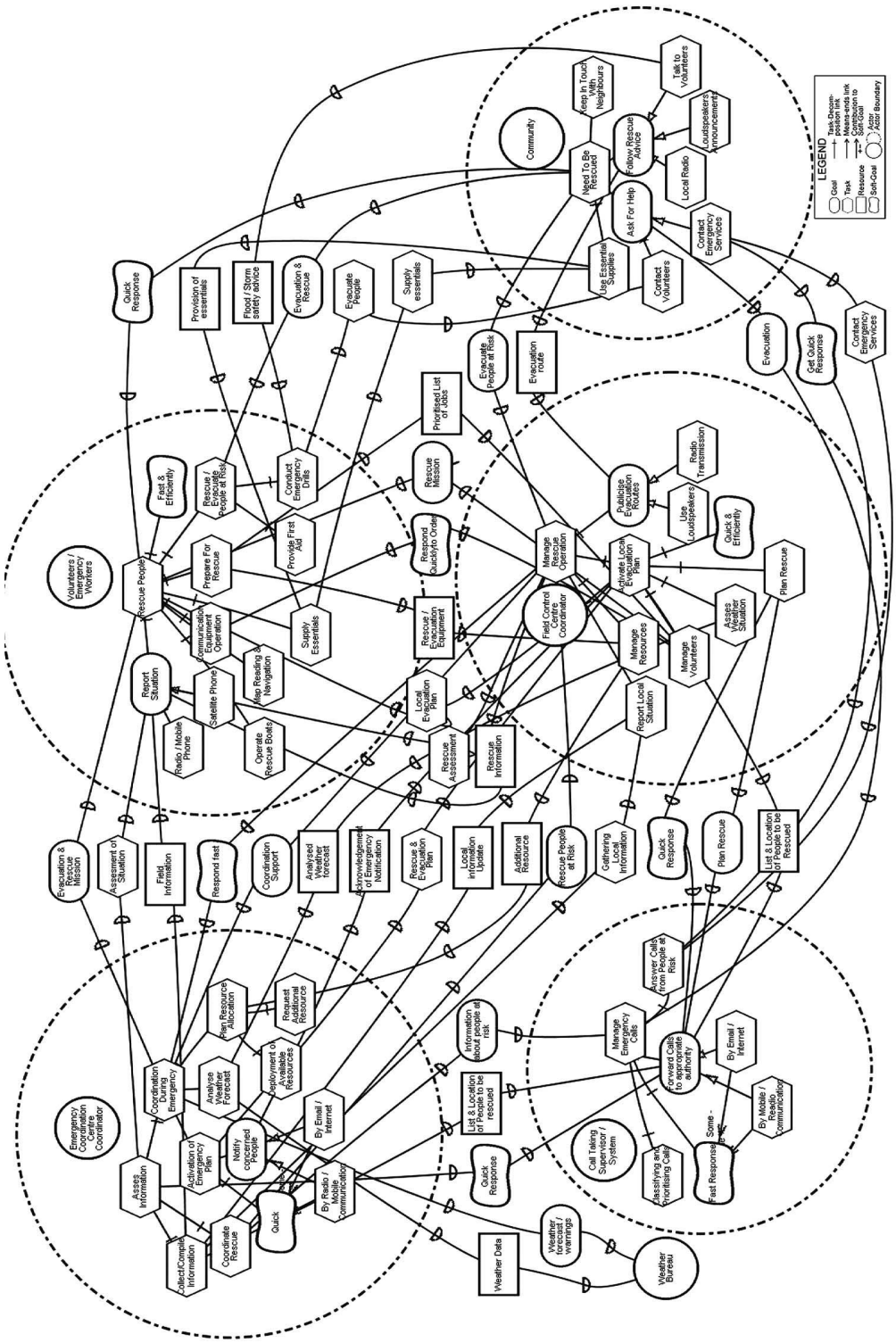

Figure 2. The *i Model of the Flood Rescue Management case study**