Consistency preserving co-evolution of formal specifications and agent-oriented conceptual models

Aneesh Krishna a, Sergiy A. Vilkomir b,*, Aditya K. Ghose a

* Decision Systems Laboratory, School of Information Technology and Computer Science, University of Wollongong, NSW 2522, Australia
b Software Quality Research Laboratory, Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville, TN 37996-3450, USA

A R T I C L E   I N F O

Article history:
Received 24 November 2006
Received in revised form 11 March 2008
Accepted 26 May 2008
Available online 3 June 2008

Keywords:
Requirements Engineering
Z notation
i framework
Formal specifications

A B S T R A C T

Many modelling techniques tend to address “late-phase” requirements while many critical modellings decisions (such as determining the main goals of the system, how the stakeholders depend on each other, and what alternatives exist) are taken during early-phase requirements engineering. The i framework is a semiformal agent-oriented conceptual modelling language that is well-suited for answering these questions. This paper addresses key challenge faced in the practical deployment of agent-oriented conceptual modelling frameworks such as i. Our approach to addressing this problem is based on the observation that the value of conceptual modelling in the i framework lies in its use as a notation complementary to existing requirements modelling and specification languages, i.e., the expressive power of i complements rather than supplants 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. This research examines how this might be done with formal specification notations (specificaly Z).

© 2008 Elsevier B.V. All rights reserved.

1. Introduction

Requirements Engineering is the process which identifies the purpose of a software system and documents it in a form which is responsive to detailed examination, communication followed by implementation [21]. The criticality and centrality of requirements engineering is now widely acknowledged, given that the majority of software errors can be traced back to errors in the requirements engineering phase, and given the difficulty of recovering from these errors in subsequent phases (i.e., design specification, coding, testing, etc.) of the software life cycle [3]. Conceptual modelling plays an important role in requirements engineering, both by offering the means to represent models in a manner that can be understood by a large and diverse set of stakeholders, and also by offering the possibility of performing certain kinds of analysis on these models very early in the software life cycle.

Agent-Oriented Conceptual Modelling (AOCM) is a novel approach to conceptual modelling that has gained considerable credence within the research community over the last decade [13,28]. The key innovation in AOCM is the use of the notion of an agent, together with associated concepts such as goals, plans, commitments etc., as modelling constructs. This has been inspired, in part, by the growing popularity of agent-oriented approaches to the building of intelligent systems, within the artificial intelligence, and related research communities. This paper focuses on a specific AOCM approach – the i framework [28]. AOCM frameworks are of interest for several different reasons. First, they offer modelling constructs at a higher level of abstraction than other existing conceptual modelling notations. In many ways, this parallels the transition from object-oriented programming to agent-oriented programming, with the concomitant benefits that accrue from being able to program at a higher-level of abstraction [31]. Second, they permit greater emphasis to be placed on the early phase of requirements engineering [29]. Much of the existing work in requirements engineering has focused on modelling and specification languages for the late-phase, and assume that an initial statement of the requirements is always available. The late-phase of requirements engineering focuses on the completeness, consistency, and automated verification of requirements. In contrast, the early phase intends to model and analyse stakeholders’ interests and how they might be addressed, or compromised, by means of various system and environment alternatives [30]. AOCM notations offer abstractions that are sufficiently high-level to be particularly suitable to early-phase requirements modelling. Third, they provide an affective means of modelling complex organisational contexts. Fourth, they allow us to build rich models of stakeholder goals and intentions. While goal-oriented requirements engineering has been developed separately, AOCM frameworks such as i permit us to situate goal representations within richer models of organisational context [26]. Models of stakeholder intentions...
permit us to answer “why” questions by pointing to these rich models of stakeholder goals and organisational contexts [27]. While existing research on requirements and design rationale also seek to answer these questions, AOCM notations offer a far richer language for modelling rationale.

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 TROPOS project [7,6] 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 Formal Tropos [12] has been proposed. Proposals to integrate $i^*$ with formal agent programming languages have also been reported in the literature [25]. This paper has similar objectives, but takes a somewhat different approach. We believe that the value of conceptual modelling 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 $i^*$ modelling 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 [19,2]). Our aim, then, is to support the modelling of organisational contexts, intentions and rationale in $i^*$, while traditional specifications of functionality and design proceeds in the formal notation. In this paper, we focus on $Z$ [23] as a prototypical representative of a formal notation, but observe that many of the lessons generalise to other formal methods. More generally, this research suggests how diagrammatic notations for modelling early-phase requirements, organisational contexts and rationale can be used in a complementary manner with more traditional specification notations. Fig. 1 shows the scope of our work with respect to the TROPOS methodology [7].

$Z$ [23] is a formal notation for computer systems and software specification based on set theory and first order predicate logic. This mature formal method is widely used both in theoretical investigations [1] and in practice [5]. The main elements of the $Z$ notation are schemas which are used to specify states and operations for the modelling of systems. While $Z$ can be used for early-phase requirements modelling, the necessary level of formalisation, precision and detail, the lack of a diagrammatic notation to support the visualisation of requirements and the inability to represent the intentional elements all suggest that an alternative notation such as $i^*$ might be better suited for this phase. Our aim, then, is to support the modelling of organisational contexts, intentions and rationale in $i^*$, while traditional specifications of functionality and design proceeds in the $Z$ notation.

Our proposal for a synergistic combination of $i^*$ and $Z$ offers several advantages:

- $i^*$ and $Z$ can be viewed as a pair of complementary representation languages that can be jointly brought to bear on the requirements engineering exercise. The $i^*$ notation permits us to make explicit the intentional aspects of the requirements specification, including an understanding of the organisational context of the proposed system, the alternatives that may be considered in making design decisions as well as the rationale behind these decisions (these latter features support process reengineering). The $Z$ notation permits us to specify late-phase requirements with a degree of precision and formality that $i^*$ does not.
- The $i^*$ notation allows us to represent and reason with softgoals (representations of non-functional requirements or objectives).
- We propose a mapping from $i^*$ models into $Z$ schemas that does not result in any information loss to the original $i^*$ model (this is distinct from proposals such as the one involving mapping $i^*$ models to ConGolog agent programs [25], where aspects of the $i^*$ model are ignored in the translation).
- The mapping of $i^*$ models to $Z$ schemas enables the refinement of these schemas with additional information, such as invariant properties, fulfilment conditions etc. (note that these cannot be represented in the original $i^*$ model).
- Current approaches to the use of formal methods in conjunction with $i^*$ models are unduly complicated. Formal Tropos [12], for instance, is an intermediate language in which $i^*$ models must be defined before an eventual translation into a state machine model on which model checkers can be deployed to verify systems properties (the process also assumes a significant amount of refinement of the original model with additional information). Existing tool support for $Z$, on the other hand, allows analysis of specifications without any of this additional effort.

When proposing the co-evolution of two otherwise disparate approaches for requirements engineering, we need to take care the issue of maintaining consistency between the two approaches. The mapping rules can be viewed as providing formal semantics to $i^*$ diagrams by mapping this notation into $Z$ specifications, a language which already has richer semantics. A set of mapping rules is defined to help ensure consistency between the two models.

This paper is an extension and refinement of our earlier results [24,17,18] and is structured as follows. In Section 2, we present formal basis for co-evolution of models. Section 3 describes the Flood Rescue Management case study. This case study will be used to illustrate both the $i^*$ notation and our proposed methodology. Section 4 presents the mapping between $i^*$ models and $Z$ schemas. Section 5 presents the mapping of specific $i^*$ model into $Z$ schemas. Section 6 introduces refinement steps. The examples are specially interesting on account of the pointers we provide on how an initial set of $Z$ schemas obtained from an $i^*$ model might be refined in useful ways. Section 7 introduces a methodology for supporting the co-evolution of $i^*$ models and $Z$ specifications. Section 8 discusses how consistency is preserved during the co-evolution of formal and informal models. Finally, Section 9 presents concluding remarks.

2. A formal basis for co-evolution of models

A key concern of this research is the definition of methodologies that permit models in distinct notations (ideally with complementary representational capabilities) to co-evolve. Informally, models in distinct notations are said to co-evolve if the following are true:

- These models are independently maintained and updated, possibly by distinct sets of stakeholders.
At any given point in time, these models satisfy a set of formal or informal inter-model consistency constraints. In other words, the models in distinct notations must not represent contradictory descriptions of the same reality.

The key to formalising this notion is to understand inter-model consistency constraints. These are, in general, hard to obtain and must be hand-crafted for every pair of notations of interest. A more achievable approach involves, for a pair of models in distinct notations:

- Mapping each model into a model in the other notation.
- Using the consistency rules or semantics intrinsic to each notation to determine if the models are consistent. These consistency rules or semantics are usually easier to come by and even when they are not formally specified, consistency is relatively easy to manually verify.

Formally, this can be achieved by defining a mapping function in the following manner. Let $N_1$ and $N_2$ be two distinct modelling notations. Let $f_{N_1, N_2} : M_{N_1} \rightarrow M_{N_2}$, where $M_{N_1}$ and $M_{N_2}$ are the sets of all possible models expressible in $N_1$ and $N_2$, respectively, be a function that maps a model in $N_1$ to a model in $N_2$. Ideally, such a function must generate an $N_2$ model that expresses as much of the input model (in $N_1$) as can be expressed in $N_2$. $f_{N_1, N_2}$ is similarly defined. Co-evolution of models in $N_1$ and $N_2$ can then be defined as follows:

If $M_1$ and $M_2$ are the current models in $N_1$ and $N_2$, respectively, then it must be the case that $f_{N_1, N_2}(M_1)$ is consistent with $M_2$ and $f_{N_2, N_1}(M_2)$ is consistent with $M_1$.

A key question is to understand the properties of these mapping functions. Intuitively, one would expect these functions to satisfy the properties of soundness and completeness in a specialised sense. Soundness in this context requires that only information included in the input model to the mapping function be represented in the output model. Completeness requires that as much of the information included in the input model as can be represented in the notation of the output model is indeed included in the output model.

These properties are difficult to prove formally for such mapping functions. Realising provably sound and complete mapping functions and formally grounded machinery to support co-evolution represents a major research agenda. We present preliminary progress in that direction in this paper. To be precise (in the context of this paper), the goal of this exercise is not to guarantee consistency, but to detect inconsistencies.

### 3. Case study: flood rescue management

This section presents a case study based on a collaborative project to build a comprehensive enterprise model for an Emergency Services Agency (ESA), Australia [17,18]. The case study concentrates on a key function of the ESA: managing flood rescue and evacuation operations.

The ESA is responsible for managing diverse emergency situations. The case study deals with an event, which can be described in many different ways, but from an ESA perspective the event is definitely a flood response operation. The timing of the emergency response is critical in these scenarios. The ESA is the agency chosen to deal with these kinds of situations since it has the expertise and appropriate resources to deal with the threat.

During this emergency situation, an Emergency Coordination Centre is formed and the Coordinator (ECCC) heads it. The first action taken by the coordinator is to activate the emergency plan partially or fully depending on the situation. The main function of the coordinator is to bring together elements of the organisation together to ensure effective management response and is primarily concerned with the systematic planning and application of resources (manpower and equipment). ECCC is responsible for the acquisition of additional resources requested by different Field Control Centre Coordinators (FCCC). Other responsibilities of ECCC include, to collect and assess field information so that rescue and evacuation operation can be coordinated in an efficient manner. Analyse weather forecast supplied by weather bureau and forward the analysed forecast to the concerned FCCC with necessary comments/observations.

The Field Control Centre is a facility where the FCCC is located, usually near the scene of an emergency, to facilitate control and management of the emergency. The FCCC is primarily responsible for managing the rescue and evacuation operation in the flood affected area. The FCCC is also responsible for publicising evacuation routes for the community, managing volunteers and available resources at his disposal in most optimal way.

The other people involved in the case study are Call Taking Supervisor/System, Volunteers/Emergency Workers, Community and Weather Bureau. Call Taking Supervisor/System is responsible for managing/handling calls from the affected people, classifying/prioritising them and forwarding calls to concerned authorities for further action. Volunteers/Emergency Workers are very important actors in the whole emergency situation. They are trained in all aspects of rescue operation. They are proficient in general rescue, providing first aid, operating communication equipment, map reading and navigation, flood rescue boat operations, giving storm safety advice, provision of essentials to people cut off by flood waters etc. Community actors in our case study are people who are affected by the flood. They are the people who are living in the flood-prone area. They are concerned about many issues and would like to know the answers of following questions:

- How deep could the water get in and around the property?
- Whether I might need to evacuate or will I be cut off by flooding in the area?
- Which are the safest evacuation routes?

The volunteers and the ESA provide the answers to these questions. Weather Bureau is responsible for providing weather forecast data that is crucial for the efficient planning of emergency operation. The other people involved in rescue and evacuation mission are dependent on weather bureau to provide forecast data at regular interval for the duration of emergency.

Consider now the ESA model using $i$. The $i$ notation consists of two main modelling components: the Strategic Dependency Model (SD) and the Strategic Rationale Model (SR) [29]. The SD model provides an external characterisation of an actor/agent in terms of two sets of dependencies: incoming dependencies (the agent acting as dependee) and outgoing dependencies (the agent acting as depender) [31]. In SD model this is represented by showing the left half-arrow (pointing from right to left) to denote incoming dependency and the right half-arrow to denote outgoing dependency. The case study shown in Fig. 2 is used to illustrate the SD model [28] of managing flood rescue and evacuation operations by ESA. The modelling process begins with the identification of actors/agents involved in the flood rescue and evacuation operation and identifying mutual relationships between them. The EmergencyCoordinationCentreCoordinator (ECCC) agent depends on the FieldControlCentreCoordinator (FCCC) agents to accomplish its goal RescuePeopleAtRisk. Similarly the FCCC agent depends on the ECCC agent to achieve CoordinationSupport goal. The ECCC depends on the CallTakingSupervisor/System to provide InformationAboutPeopleAtRisk, modelled as a goal dependency. Similarly, ECCC agent depends on WeatherBureau and Volunteers/EmergencyWorkers agents...
to achieve the goals WeatherForecast/Warnings and Evacuation and RescueMission, respectively. The ECCC has a dependency on the WeatherBureau to provide WeatherData, modelled as a resource dependency. Similarly, ECCC has a dependency on the Volunteers/EmergencyWorkers and FCCC agents to provide FieldInformation and AcknowledgmentOfEmergencyNotification, LocalInformationUpdate, respectively, modelled as resource dependencies. The remaining dependencies may well be explained on the similar lines.

An SR model 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 intentional
elements linked by task-decomposition and means-ends relationships. The case study shown in Fig. 3 is used to illustrate the SR model [31] of managing flood rescue and evacuation operations by ESA. Also shown are SR models to show the internal intentional elements of the actors/agents namely, CallTakingSupervisor/System (Fig. 4), Volunteers/EmergencyWorkers (Fig. 5), EmergencyCoordinationCentreCoordinator (Fig. 6), FieldControlCentreCoordinator (Fig. 7) and Community (Fig. 8), respectively.

For example, Volunteers/EmergencyWorkers has an internal task to RescuePeople. This task can be performed by subtasks PrepareForRescue, MapReading&Navigation, OperateRescueBoats, CommunicationEquipmentOperation, SupplyEssentials.
Rescue/EvacuatePeopleatRisk and the goal ReportSituation (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). Each of these tasks and goals are related to the parent task via task decomposition links. The Rescue/EvacuatePeopleatRisk task is further decomposed into subtasks ProvideFirstAid and ConductEmergencyDrills. The softgoal Fast&Efficiently is also related to the RescuePeople task via a task decomposition link. When a softgoal is a component in task decomposition, it serves as a quality goal for that task. Assessment of the local flood situation by Volunteers/EmergencyWorkers in the flood-affected region is crucial for the ECC and FCCC agents to further plan their resources and strategies in an optimal way. This is represented as subgoal ReportSituation in the SR model of Volunteers/EmergencyWorkers. This subgoal can be achieved by using any one of the shown subtasks, Radio/MobilePhone or SatellitePhone (this is represented by a means-ends link connecting the sub-goal to the two alternative means). Observe that both task-decomposition and means-ends links answer the “how” question relating to tasks and goals, respectively. 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.

4. Mapping $i'$ into Z

4.1. Mapping a general SD model into Z

The notion of states of dependency is implicit in $i'$, but requires explication in Z specifications. We are presenting results from
our earlier work [24] which has been thoroughly refined and extended to incorporate co evolution methodology and discussion on the detection of inconsistencies.

All elements (actors and dependencies) of a SD model differ in names. For describing their names in the Z notation we introduce the basic type (given set) NAME. Given an SD model, one can refer to distinct subsets of NAME. The subset all_actors contains the names of all actors while the subset all_depend contains the names of all dependencies in the SD model.

\[
\begin{align*}
\text{NAME} \\
\text{all_actors, all_depend : P \_ NAME}
\end{align*}
\]

It is necessary to mention that names of internal intentional elements of a SR model are also members of the given set NAME but do not belong to subset all_depend. Formalisation of these internal intentional elements is considered later in the paper.

Both SD and SR models provide a description of the intentional relationships among actors of a process and do not directly address the dynamics of this process. But exactly the dynamics are the most important for process or system specification. To reflect it, we use the fact that all dependencies in SD and internal intentional elements in SR are realised dynamically: a goal is achieved, task is performed or resource becomes available. We consider different states of the dependencies (elements) before and after realisation using the following free type definition:

\[
\begin{align*}
\text{STATE} & \equiv \text{inapplicable} \mid \text{unresolved} \mid \text{fulfilled} \\
& \mid \text{violated} \mid \text{satisfied} \mid \text{denied} \\
& \mid \text{undetermined}
\end{align*}
\]

State inapplicable is held before the creation of a new instance of a dependency (element). State unresolved conforms to a dependency (element) after the creation but before realisation and all other states are conforming to a dependency (element) after realisation. The dependency (element) is in state fulfilled if realisation is successful and in state violated if realisation is unsuccessful. With the idea of keeping uniform terminology with other researchers (e.g., [8]) in the area, for softgoals we use two states satisficed and denied. The last state undetermined can also be used only for a softgoal. Softgoals are often identified with quality criteria and sometimes it is impossible to conclude immediately after realisation whether a quality criterion is satisfied. It means that it may not be clear whether the realisation had been successful or not. In this case we consider the softgoal is in the undetermined state.

Note on Softgoals: Softgoals are viewed as optimisation goals where there is no way of actually specifying whether the softgoal was achieved completely. Where a task contributes positively to the achievement of a softgoal for a system, it is desirable that the system design implements that task. In a case where there are multiple tasks contributing positively to the achievement of a softgoal we are faced with a design decision. Since either task will contribute positively to the goal there is a need to evaluate which of the tasks should be implemented.

A task, which contributes negatively to a softgoal, should be avoided in the system implementation and further design and elaboration work should ensure that the task is not implemented. There may however be scenarios where a task needs to be implemented, as a result of another dependency which will contribute negatively to a softgoal. At that stage domain experts and stakeholders will need to outweigh the advantages against the disadvantages of not achieving the specific softgoal.

This work does not propose a new way of dealing with the softgoals during requirements engineering and this remains focus of the future work. The ideas in this procedure are complementary to other approaches which specifically focus on softgoals during requirements engineering. One such approach is the NFR framework [8,9].

The state of a whole SD model is the set of states of all its dependencies for this SD model that is reflected in SD schema:

\[
\text{SD} \\
\text{SD\_state : NAME -> STATE} \\
\text{dom.SD\_state = all\_depend}
\]

Thus, the realisation of a dependency changes its state and at the same time changes a state of the whole SD model.

Each SD dependency or SR element has its own specific features and differs first in types and degrees.

\[
\begin{align*}
\text{TYPE} & \equiv \text{goal} \mid \text{softgoal} \mid \text{task} \mid \text{resource} \mid \text{ISA} \\
\text{DEGREE} & \equiv \text{open} \mid \text{committed} \mid \text{critical}
\end{align*}
\]

In contrast to other values, the ISA type does not represent a dependency. It means that one actor can be considered as a special instance of other actor. Since, ISA is a relationship between two actors it is convenient for us to consider them together as a different values of TYPE. All other values of free type definitions TYPE and DEGREE are standard for the i framework.

All the dependencies in SD (as well as every element in SR model) are described by its own schema. A general structure of SD dependencies (external between actors) varies from a general structure of SR elements (inside actors) but at the same time they have some common patterns. That is why we use the following steps of formalisation, creating successively:

- Φdepend schema which describes a common pattern of SD dependencies and SR elements; (the Φdepend in the schema name is used to flag a partial specification [23]).
- SDependency schema which describes a general structure of all the SD dependencies and includes Φdepend schema as one of the component part.
- A detailed schema for every SD dependency using SDependency

\footnote{All schemas in this paper are checked using the ZTC type-checker package [14].}
schema as a basis.

Common patterns for SD dependencies and SR elements are represented in \(\Phi\text{Depend}\) schema. Here, \(\Phi\) is a part of the schema name, not an operator. It is just a naming convention used to indicate a partial (incomplete) specification [4].

\[
\Phi\text{Depend}
\]
\[
\begin{align*}
\text{dependum} & : \text{NAME} \\
\text{type} & : \text{TYPE} \\
\text{degree} & : \text{DEGREE} \\
\text{result} & : \text{STATE}
\end{align*}
\]

result\(\neq\) unresolved
result\(=\)satisfied \lor result\(=\) denied \lor
result\(=\)undetermined \(\Rightarrow\) type = softgoal

Except for the above-mentioned type and degree, specific features of every dependency are its name \(\text{dependum}\) and resulting state, which is represented by the output variable \(\text{result}\):

- The first line of the predicate part of \(\Phi\text{Depend}\) describes the fact that the resulting state cannot be unresolved.
- The second line of the predicate part of \(\Phi\text{Depend}\) reflects that the resulting state can take the satisfied, denied or undetermined value only for softgoals.

The following \(SD\text{ependency}\) schema is a result of one-to-one mapping of the general structure of a SD dependency into the Z notation. This schema is an operation schema and changes the state of the SD model \(\Delta SD\). \(SD\text{ependency}\) schema includes the components \(\Phi\text{Depend}\) schema as well as names of actors \(\text{dependee} / \text{depender}\) which are linked by the dependency. While this schema represents a general structure, its name, type, degree and names of actors are not specified. It could be done later on during the consideration of an \(i^*\) model for a specific example.

\[
SD\text{ependency}
\]
\[
\Delta SD
\]
\[
\Phi\text{Depend}
\]
\[
\begin{align*}
\text{dependee}, \text{depender} & : \text{NAME} \\
\text{dependee} \in \text{all\_depend} \\
\text{depender} \in \text{all\_depend} \\
\text{dependee} \in \text{all\_actors} \\
\text{depender} \in \text{all\_actors} \\
SD\text{_state} = SD\text{_state} \uplus \{ \text{dependum} \mapsto \text{result} \}
\end{align*}
\]

The most significant information is contained in the last line of the predicate part of this schema, which describes how the realisation of the dependency changes the state of the SD model. Using the \text{override} operator \(\Rightarrow\) shows that the value of the SD model’s state function \(SD\text{_state}\) after the dependency realisation differs from its value \(SD\text{_state}\) before the realisation only in the part of the considered dependency and coincides for all other dependencies.

The formal correspondence between schemas and dependencies is established by using variable \text{dependum} inside the schemas without explicitly using the names of schemas. The formal rule of correspondence is described below:

\[
\begin{align*}
\text{correspond} & : \text{NAME} \mapsto \Phi\text{Depend} \\
\text{dom correspond} & = \text{all\_depend} \\
\forall x: \text{NAME} \mid x \in \text{all\_depend} \quad (\text{correspond}(x)) \text{ dependum} = x
\end{align*}
\]

The extension of the \(SD\text{ependency}\) schema is considered in the next section using information about the internal structure of actors.

4.2. Mapping a general SR model into Z

Our approach of mapping a SR model into the Z notation is similar to the approach for SD diagrams which were considered in the previous subsection. The mapping consists in consecutively creating:

- \(\text{Actor}\) schema which describes a general structure of all the actors in SR diagrams;
- \(A\text{Element}\) schema which describes a general structure of all the SR internal intentional elements and includes \(\Phi\text{Depend}\) schema as one of the component part;
- A detailed schema for every actor in the specific SR model using \(\text{Actor}\) schema as a basis;
- A detailed schema for every internal intentional element of every actor using \(A\text{Element}\) schema as a basis.

The following schema describes a general structure of all the actors. The state of an actor is given by the set of states of all its internal (SR) intentional elements (i.e., goals, tasks etc). An actor is characterised by its name \(\text{name}\), set \(\text{element}\), names of all the internal intentional elements, and state function \(\text{state}\).

\[
\text{Actor}
\]
\[
\begin{align*}
\text{actor\_name} & : \text{NAME} \\
\text{actor\_element} & : \text{p}, \text{NAME} \\
\text{actor\_state} & : \text{NAME} \mapsto \text{STATE}
\end{align*}
\]
\[
\text{dom actor\_state} = \text{actor\_element}
\]

The \(\text{actor\_state}\) function is similar to the SD model’s state function \(SD\text{_state}\) and represents a collection of states of all internal intentional elements of the actor.

For formalizing a general structure of all SR elements, we need to introduce a new free type, which describes possible types of links between the elements.

\[
\text{LINK\_TYPE} ::= \text{NA} \mid \text{task\_decomp} \mid \text{means\_ends}
\]
\[
\mid \text{contrib}
\]

Type \(\text{NA}\) (Non-Applicable) is used for elements which have no means for attaining them and have no components. Type \(\text{task\_decomp}\) represents task decomposition links. Types \(\text{means\_ends}\) and \(\text{contrib}\) describe means-ends links. Type \(\text{means\_ends}\) is used for Goal-Task Link, Resource-Task Link, Softgoal-Task Link and Softgoal-Softgoal Link, respectively. Type \(\text{contrib}\) represents special kinds of means-ends links for softgoal (Softgoal-Task and Softgoal-Softgoal links).

For convenience, we allocate all conditions connected with links into a separate schema \(\text{Link}\). This schema includes:

- names of internal (inside the actor) elements \(\text{int\_components}\) which are linked with the considered element;
- names of external (from SD model) dependencies \(\text{ext\_components}\) which are linked with the considered element;
- type of the link;
- names of elements which give positive \((\text{contrib\_p})\) and negative \((\text{contrib\_m})\) contribution to the softgoals.
The predicate part describes the following constraints between types of links and types of elements:

- Task decomposition links are used only for tasks.
- Positive or negative contribution is possible only for softgoals.
- Tasks are used in connection with external components.

The following schema describes a general structure of all the SR internal elements. This operational schema changes the state of the general model of an actor (\(D\)Actor). Similarly SDependency schema, AElement one includes as components \(\Phi\)Depend schema. Inclusion of Link schema brings all the information concerning links between the elements.

### Link

\[
\text{Link} \\
\Phi\text{Depend} \\
\text{int}_{\text{components}}, \text{ext}_{\text{components}} : \text{P NAME} \\
\text{contrib}_p, \text{contrib}_n : \text{P NAME} \\
\text{link} : \text{LINK\_TYPE}
\]

\[
\text{link} = \text{task}\_\text{decomp} \Rightarrow \text{type} = \text{task} \\
\text{link} = \text{contrib} \Rightarrow \text{type} = \text{softgoal} \\
\text{contrib}_p \cup \text{contrib}_n \neq \emptyset \Rightarrow \text{link} = \text{contrib} \wedge \\
\langle \text{contrib}_p, \text{contrib}_n \rangle \text{partitions} \\
\text{int}_{\text{components}} \\
\text{ext}_{\text{components}} \neq \emptyset \Rightarrow \text{type} = \text{task} \\
\text{link} = \text{NA} \Rightarrow \text{int}_{\text{components}} = \emptyset
\]

The predicate part of AElement schema formalises the changes of Actor schema under the realisation of the internal intentional element. Only one component of Actor schema namely the actor's state function \(\text{actor\_state}\) is changed. Similar to the SD model's state function \(\text{SD\_state}\), the difference between the values of \(\text{actor\_state}\) before and after the element realisation exists only in the state of the considered element.

Using information about internal elements of actors, we can now extend the SDependency schema and add the names of the internal intentional elements (\(\text{depend}\_\text{internal\_element}\) and \(\text{dependee\_internal\_element}\)) linked to the dependency:

### AElement

\[
\Delta\text{Actor} \\
\text{Link}
\]

\[
\text{dependem} \in \text{actor\_element} \\
\text{int}_{\text{components}} \subset \text{actor\_element} \\
\text{ext}_{\text{components}} \subset \text{all} \text{depend} \\
\text{actor\_name'} = \text{actor\_name} \\
\text{actor\_state'} = \text{actor\_state} \circ \\
\{\text{dependem} \mapsto \text{result} \}
\]

The next step in our methodology is the mapping of specific SD model into Z schemas. Following steps are carried out to realise this goal:

1. Specifications of the names of all the actors and external dependencies.
2. The creation of Z schema for every dependency using SDependency schema as a basis.
3. Specifications of the names of all the internal intentional elements of the selected actor.
4. The creation of a Z schema for every internal intentional element using AElement schema as a basis.

#### 5.1. Mapping the SD model

Names of all the actors and external dependencies are specified. This is the first step of mapping SD model of managing flood rescue and evacuation operations case study. First of all it is necessary to describe their names in Z using the following axiomatic definition:

Line \(\text{depend} = \text{evacuation}\) shows the name of the dependency. It is a goal dependency so \(\text{type} = \text{goal}\). The community actor depends on the supervisor so \(\text{depend} = \text{community}\) and \(\text{dependee} = \text{supervisor}\). The importance of the dependency is not marked in the SD diagram hence we consider \(\text{degree} = \text{committed}\).

Thus, \(\text{Evacuation}\) schema corresponds to the evacuation dependency. It is intuitively obvious because of the similarity of names (we use this similarity only for clarity purpose). The value of variable \(\text{depend}\) determines the formal correspondence between the schema and the dependency.

More detailed examples of schemas for another types of dependencies are considered in the next section using SDependencySR schema as a basis.

#### 5.2. Mapping the SR model

The first step of mapping (formalisation) SR model of managing flood rescue and evacuation operations case study is the creation
of a Z schema for every actor using Actor schema as a starting point. In these schemas we need to specify names of all the internal intentional elements of the selected actor. The schemas for supervisor and field_coordinator actors are provided as ready reference below:

```plaintext
_Supervisor_

Actor
classifying_calls, fast_response, mobile, email,
forward_calls, manage_emergency_calls,
answer_calls : NAME

actor_name = supervisor
actor_element = (classifying_calls,
fast_response, mobile, email, forward_calls,
manage_emergency_calls, answer_calls)

_FieldCoordinator_

Actor
manage_rescue_operation,
report_local_situation, manage_resources,
activate_local_evacuation_plan,
publicise_evacuation_routes,
manage_volunteers, use_loudspeakers,
radio_transmission, assess_weather_situation,
quickly, efficiently, plan_rescue : NAME

actor_name = field_coordinator
actor_element = (manage_rescue_operation,
report_local_situation, manage_resources,
activate_local_evacuation_plan,
publicise_evacuation_routes,
manage_volunteers, use_loudspeakers,
radio_transmission, assess_weather_situation,
quickly, efficiently, plan_rescue)
```

The second step is the creation of a Z schema for every internal intentional element using AElement schema as a basis. As an example, we will need to create 7 schemas for the internal intentional elements of supervisor actor. It is necessary to specify the name of the dependee, the type and the degree of the element (similar to external dependencies) but also the kind of the link and names of external and internal components of the considered element. To demonstrate this approach, we are showing five schemas as ready reference:

```plaintext
_AElement

Dependence

Supervisor

classifying_calls

type = task

degree = committed

int_components = {}

ext_components = {}

link = NA

Map

AElement

Volunteer

classifying_calls

type = task

degree = committed

int_components = {}

ext_components = {}

link = NA

PubliciseEvacRoutes

AElement

Field_coordinator

classifying_calls

type = task

degree = committed

int_components = {}

ext_components = {}

link = NA

Using information from schemas of specific actors and SDependencySR schema as a basis, we can provide more detailed description of dependencies. We consider the following three schemas as examples:

WeatherData

SDependencySR

classifying_calls

type = task

degree = committed

int_components = {}

ext_components = {}

link = NA

RescuePlan

SDependencySR

classifying_calls

type = task

degree = committed

int_components = {}

ext_components = {}

link = NA

RespondQuickly

SDependencySR

classifying_calls

type = task

degree = committed

int_components = {}

ext_components = {}

link = NA
```
The schemas for all the other dependencies are similar and can be realised in the same manner.

We have considered Z schemas represented above as part of one to one mapping of \( i \) models into the Z notation. Using this approach, all the information from the \( i \) models is reflected in Z. 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).

6. Refinement

A key subsequent step is the refinement of these essentially structural schemas (as discussed in the previous section) with additional information (i.e., information not included in an \( i \) model, but obtained via further Analysis – e.g., temporal sequencing of dependencies, fulfilment conditions for dependencies etc). The finer points of this refinement are provided in this section. We shall refer to the Z specification obtained after these refinements as the Extended Z Specification.

We are providing the essence of some possible approaches with examples.

6.1. Order or Sequence on the dependencies

When we inspect SD and SR diagrams, we are not in a position to ascertain the order or sequence in which various dependencies are realised. This missing information can be easily represented in the resulting Z schemas without making any changes to the SD or SR diagrams.

EXAMPLE I: Refer to the SR diagram which describes the intentional relationships that are “internal” to the Call Taking Supervisor/System actor. It is necessary to answer and classify a call before forwarding it to the appropriate authority so dependencies answer_calls and classifying_calls should be realised before dependency forward_calls. This fact is not reflected in the \( i \) diagrams but can be easily incorporated into Z schemas. For this, it is necessary to include into the predicate part of ForwardCalls schema the following implication:

\[
(SDstate(forward_calls) = \text{fulfilled}) \Rightarrow \\
(SDstate(answer_calls) = \text{fulfilled}) \land \\
(SDstate(classifying_calls) = \text{fulfilled}).
\]

EXAMPLE II: For realizing resource dependency Prioritised List of Jobs, Volunteers/Emergency Workers agent depends on the Field Control Centre Coordinator agent. Prior to the success of this dependency, following dependencies must be realised: Task dependency contact_emergency_services between Community and the Call Taking Supervisor/System agents and Resource dependency list_location_people_to_be_rescued between Call Taking Supervisor/System and Field Control Centre Coordinator agents. For this, it is necessary to include into the predicate part of PrioritisedList schema the following implication:

\[
(SDstate(prioritised_list_jobs) = \text{fulfilled}) \Rightarrow \\
[(SDstate(contact_emergency_services) = \text{fulfilled}) \land \\
(SDstate(list_location_people_to_be_rescued) = \text{fulfilled})].
\]

6.2. Resource dependency structure

After inspecting the \( i \) diagrams, we cannot predict about the organization, construction and contents of the given resource dependency. We shall now present an example to explain different categories of resources possible in \( i \) framework with their contents and mode of communication.

Free types CONTENT_TYPE, FORM_TYPE describe the contents and form (or mode of communication) of the resources, respectively.

\[
\begin{align*}
\text{CONTENT_TYPE} & := \text{yes_no} \mid \text{running_text} \\
\text{FORM_TYPE} & := \text{e-mail} \mid \text{fax} \mid \text{booklets} \\
& \mid \text{telephone} \mid \text{mobile_phone} \\
& \mid \text{pager} \mid \text{miscellaneous}
\end{align*}
\]

EXAMPLE I: The Field Control Centre Coordinator actor/agent depends on the Emergency Coordination Centre Coordinator actor for providing Analysed Weather Data. This is modelled as a resource dependency in the SD diagram. The contents of this resource are information about the flood warnings, rainfall (rainfall location and intensity – every 10 min) and river information (if any in the area), forecast and observation index etc. This resource is in the form of technical data/report and is communicated via e-mail attachment or faxed.

\[
\begin{align*}
\text{AnalysedData} & \quad \text{SDependencySR} \\
& \quad \text{content} : \text{CONTENT_TYPE} \\
& \quad \text{form} : \text{FORM_TYPE}
\end{align*}
\]

\[
\begin{align*}
\text{dependum} & = \text{analysed_weather_data} \\
\text{dependder} & = \text{field_coordinator} \\
\text{dependec} & = \text{coordinator} \\
\text{dependder\_internal\_element} & = \text{FieldCoordinator.analysed_weather_forecast} \\
\text{dependec\_internal\_element} & = \text{Coordinator.analysed_weather_forecast} \\
\text{type} & = \text{resource} \\
\text{degree} & = \text{committed} \\
\text{content} & = \text{technical\_data} \\
\text{form} & = \text{e-mail} \lor \text{form} = \text{fax}
\end{align*}
\]
EXAMPLE II: The Field Control Centre Coordinator actor/agent depends on the Call Taking Supervisor/System actor for forwarding the List and Location of People to be rescued. This is modelled as a resource dependency in the SD diagram. It consists of the list of people to be rescued along with their address, location and telephone numbers. There is priority on the list depending upon the flood/storm situation. This resource is in the form of the list and is communicated via e-mail, fax or mobile/radio communication.

6.3. Association or Connection between different resources

The information about the association between the resources is missing in the i* framework. Whether two or more resource dependencies are connected with each other in a certain way is missing in the SD and SR diagrams. Some resources may be subset of the preceding resource dependency; some resource dependencies might contain the same information (but they might exist between different actors). We shall now present an example to explain this refinement case.

EXAMPLE: The resource dependency Prioritised List of Jobs (between Volunteers/Emergency Workers agents/actors and Field Control Centre Coordinator) contains all the information from the resource dependency List and Location of People to be Rescued (between Call Taking Supervisor/System agents/actors and Field Control Centre Coordinator). We can say that first resource dependency is the subset or part of the second resource dependency. The other observation is that the second resource precedes the first resource dependency. We are going to use the basic type RESCUE_INFORMATION to describe the information about the people to be rescued.

[RESCUE_INFORMATION]

Now both ListTobeRescued and PrioritiseList schemas can be extended. A new variable list_people should be added in ListTobeRescued. An extended version of PrioritiseList schema is the following:

6.4. Temporal features and operators

Temporal Logic allows us to specify assertions about program behavior as time progresses. In other words we can describe sequences of state changes and properties of behaviors. We observe that it is worthwhile to introduce temporal logic features and operators in the mapping from i* model to Z schemas.

Fig. 9. Co-evolution of i* models and Z specifications.
The state function $SD_{state}$ represents the snapshot state of the system. To describe the behaviour of the system in time, consider all the possible sequences of system states

\[
\begin{align*}
SD_{decenarios} & \cdot \mathcal{P}(\text{seq } SD) \\
SD_{future}, SD_{past} & : SD \rightarrow \mathcal{P}(\text{seq } SD)
\end{align*}
\]

For each state function $s$ consider all the behaviours $SD_{future}$ which are started in $s$ (the future of the system) and behaviours $SD_{past}$ that are finished in $s$ (the past of the system). Now we can formalise all the main temporal operators such as sometimes in the past, always in the past, always in the future, etc., which are used in different techniques of requirement engineering, for example, KAOS [10], Formal Tropos [12]. Thus, the operator $\square \phi$ always in the future [12] for state $s$ can be modelled as:

\[
\forall c : \text{seq } SD | c \in SD_{future}(s) \wedge \{c \in SD_{past}(s) \} \wedge S D_{past}(s) = \{p : \text{seq } SD | \text{last } p = s\}
\]

Correspondingly, the operator $\circ \phi$ nextstate for state $s$ can be modelled as

\[
\forall c : \text{seq } SD | c \in SD_{future}(s) \wedge c = c(2) \cdot \phi
\]

and the operator $\diamond \phi$ eventuallyinthefuture for state $s$ can be modelled as

\[
\forall c : \text{seq } SD | c \in SD_{future}(s) \wedge \exists c : \text{seq } SD | c \in SD_{past}(s) \cdot \phi
\]

For dealing with systems which stipulate special timing requirements (like concurrent real-time reactive systems), it is possible to use special extensions of Z like Timed Communicating Object Z (TCOZ) [20] which are designed for modelling real-time applications.

7. 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 loose coupling between an $i^*$ model and a $Z$ specification. Our strategy is to localise the impact of changes. We do this at two specific points (refer to Fig. 9):

1. 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).

2. 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 (key steps in the proposed methodology are shown in Table 1 as ready reference):

- Dependencies
- Tasks
- Goals
- Resources
- Softgoals
- Means-end links
- Task-decomposition links
- 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,

<table>
<thead>
<tr>
<th>Table 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key steps in the proposed methodology of co-evolution of $i^*$ and $Z$</td>
</tr>
<tr>
<td><strong>Key steps in the proposed methodology</strong></td>
</tr>
<tr>
<td>1. Define techniques for reflecting changes in an $i^*$ model in the corresponding (unrefined) $Z$ specification</td>
</tr>
<tr>
<td>2. Define techniques for reflecting the refinements contained in the prior extended $Z$ specification to obtain a new extended $Z$ specification</td>
</tr>
<tr>
<td>3. Reverse mapping from a collection of $Z$ schemas to an $i^*$ model</td>
</tr>
<tr>
<td>(ii). 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)</td>
</tr>
<tr>
<td>(ii). Reconstruct the corresponding $i^<em>$ model without loss of information (any refinements made will, of course, not be reflected in the $i^</em>$ model)</td>
</tr>
</tbody>
</table>
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 dependee_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 dependee. 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 to an existing SR model:

(i) Addition will result in the creation of a new element schema for the goal (deletion leads to its removal). A newly added goal is typically related via a means-ends link to a task, or via a task decomposition link to another task. Schemas for these links must then also be added along the lines described below.

(ii) The element schemas for the tasks that this new goal might be linked to (as discussed above) need to be modified by adding (resp. deleting) the name of the goal to the int_components set of the corresponding schema(s).

(iii) The name of the goal must be added (resp. deleted) to the actor_element set in the element schema for the corresponding actor.

Addition/deletion of a resource to an existing SR model:

(i) Addition will result in the creation of a new element schema for the resource (deletion leads to its removal). A newly added resource is typically related via a means-ends link to a task, or via a task decomposition link to another task. Schemas for these links must then also be added along the lines described below.

(ii) The element schemas for the tasks that this new resource might be linked to (as discussed above) need to be modified by adding (resp. deleting) the name of the resource to the int_components set of the corresponding schema(s).

(iii) The name of the resource must be added (resp. deleted) to the actor_element set in the element schema for the corresponding actor.

Addition/deletion of a softgoal to an existing SR model:

(i) Addition will result in the creation of a new element schema for the softgoal (deletion leads to its removal). A newly added softgoal is typically related via a task decomposition link to another task. Potentially, it may also be related via a softgoal contribution link to an existing softgoal or a task. Schemas for these links must then also be added along the lines described below.

(ii) The element schemas for the tasks and softgoals that this new softgoal might be linked to (as discussed above) need to be modified by adding (resp. deleting) the name of the softgoal to the int_components set of the corresponding schema(s).

(iii) The name of the softgoal must be added (resp. deleted) to the actor_element set in the element schema for the corresponding actor.
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. The modified parts of SD and SR models of the managing flood rescue and evacuation operations by ESA are shown in Figs. 10 and 11, respectively.

\[
\begin{align*}
\text{SimplifiedWeatherData} \\
\text{SDependencySR} \\
\text{dependum} = \text{simplified_weather_data} \\
\text{depend} = \text{volunteer} \\
\text{dependee} = \text{field_coordinator} \\
\text{dependee_internal_element} = \text{Volunteer rescue people} \\
\text{field_coordinator_asses_weather_situation} \\
\text{type} = \text{resource} \\
\text{degree} = \text{committed}
\end{align*}
\]

The newly added resource dependency Simplified Weather Data 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 realised before dependency simplified_weather_data. For this, it is necessary to include into the predicate part of SimplifiedData schema the following implication:

\[
\begin{align*}
\text{SD state}(\text{simplified_weather_data}) &= \text{fulfilled} \\
\text{SD state}(\text{analysed_weather_forecast}) &= \text{fulfilled}.
\end{align*}
\]

Based on the second guideline 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. Minor modification is performed on the pred-

Fig. 10. Modified part of strategic dependency model of the flood rescue management case study.

Fig. 11. Modified part of strategic rationale model of the flood rescue management case study.
icate part of the concerned internal intentional element Z schema(s). We only add the dependency name Simplified Weather Data under the ext_components. 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 set in respective Z schemas. The revised Z schemas of internal intentional elements are provided as ready reference:

RescuePeople

AElement
Volunteer

dependency = rescue_people
type = task
degree = committed
int_components = {report_situation, prepare_for_rescue, evacuate_people_risk, operate_helicopter, supply_essentials, map_reading, communication_equipment_operation, fast_efficiently}

ext_components = {analysed_weather_forecast, simplified_weather_data}

link = task_decomp

We note that a reverse mapping from a collection of Z schemas to an \( \ast \) model is possible provided the following assumptions hold.

- The Z schemas were obtained from an initial \( \ast \) model via mapping and refinement along the lines described above.
- The prior \( \ast \) model is available for reference.
- 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.

The rest of the mapped Z schemas remain unchanged for the modified \( \ast \) model.

Now, lets delete internal softgoal dependency Fast and Efficiently with the actor Volunteers/Emergency Workers. This will lead to the modification of the original \( \ast \) diagram and deletion of the element Z schema (internal dependency). The element schemas for the tasks and softgoals that this softgoal might be linked to (as discussed before) need to be modified by deleting the name of the softgoal in the int_components set of the corresponding schema(s). In this case this softgoal is connected to the Rescue People task. Along with this, the name of the softgoal must be deleted from the actor_element set in the element schema for the corresponding actor (which is Volunteers/Emergency Workers in this case).

The element schema of the task Rescue People after the deletion operation will be:

The element schema of the task Rescue People before the deletion is shown as below:
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).

8. Preserving consistency in the co-evolution of formal and informal models

When proposing the co-evolution of two otherwise disparate approaches for requirements engineering, we need to maintain consistency between the two approaches. The mapping rules can be viewed as providing formal semantics to the \( i \) diagrams by mapping this notation into Z specifications, a language which already has rich formal semantics. We believe that these semantics are largely consistent with the somewhat implicit semantics for \( i \) developed in [29]. A set of mapping rules is defined to help ensure consistency between the two models. We have proposed a set of mapping rules that constrains the analyst to map the elements of the \( i \) model to appropriate Z schemas and ensures that the two models are consistent. This allows us to trace corresponding elements in the two models when changes are made. We are interested in providing a taxonomy of inconsistencies that may occur from translating \( i \) models into Z specifications (and their co-evolution). The main types of inconsistencies that may occur when performing the co-evolution of formal and informal models are listed below. The discussion on how our methodology provides support to overcome these issues is presented.

Structural inconsistency: According to our methodology, it is necessary to introduce Z schemas corresponding to the elements in the \( i \) model. If the Z specification lacks a schema for a certain \( i \) element, the combined model is inconsistent with respect to this regime. In our co-evolution methodology we are keeping the structural inconsistency issue under control by strictly adhering to the mapping rules to accommodate any changes. This allows us to keep track of corresponding elements in the two models when changes are made. 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). Hence, parsing of Z specifications will lead to one \( i \) model. Likewise, from the given \( i \) model we are in a position to arrive at Z specifications which capture and represent all the structural information contained in the given \( i \) model. Hence, with the help of clear mapping rules and a supporting methodology we are in a position to avoid structural inconsistencies.

Semantic inconsistency: As we have explained earlier, the mapping rules can be viewed as giving a formal semantics to \( i \) diagrams by mapping this notation into Z specifications, a language which has well-defined formal semantics. Semantic inconsistences may arise if the creation conditions are contradictory; invariants are not maintained. Inconsistencies may arise if the default creation condition of a subgoal of a task decomposition link or a means-ends link is that the parent goal exists, but has not been fulfilled. The fulfillment condition of the parent goal depends on the fulfillment of the subgoals. If the subgoals are connected to the parent goal with means-ends links, then fulfillment of at least one of the subgoals is necessary for the fulfillment of the parent goal. If they are connected with task-decomposition links then the fulfillment of all the subgoals is necessary. We have proposed a set of translation rules and guidelines that permit us to systematically derive these constraints. These rules capture the intuitive semantics that we use when designing an \( i \) model. For instance, a temporal ordering or sequencing refinement technique is applied in the Z schema of the parent task in the task decomposition links to include the pre-condition that all of the subgoals or subtasks are fulfilled prior to the fulfillment of the parent task. This helps us in taking care of semantic inconsistencies which may arise in the mapping of \( i \) diagrams into Z specifications.

Semantic inconsistencies can be avoided in the case of task-decomposition links in \( i \) model if all the subgoals, subtasks or softgoals connected to the parent task are realised (fulfilled or satisfied) before the realisation of the parent task. We shall provide an example to explain this. We observe that task dependency manage calls (for Supervisor agent) is decomposed into subtasks classifying calls and answer calls, subgoal forward calls and softgoal fast response. To avoid semantic inconsistencies, we should include in the predicate part of the Z schema of the task dependency manage calls, following implication:

\[
\text{(SDstate(manage calls) = fulfilled)} \Rightarrow \\
((\text{SDstate(answer calls) = fulfilled}) \land \text{(SDstate(classifying calls) = fulfilled)} \land \text{(SDstate(fast response) = satisfied)} \land \text{(SDstate(forward calls) = fulfilled)})
\]

The same kind of reasoning can be performed for the means-ends link in \( i \) model using Z. Let's consider the case of a goal dependency named forward calls (for Supervisor agent). Here, the subtasks mobile, email are connected to the parent goal with the means-ends links. Fulfillment of at least one of the subtasks is necessary for the fulfillment of the parent goal forward calls. To avoid semantic inconsistency, we should include in the predicate part of the Z schema of the goal dependency forward calls, following implication:

\[
\text{(SDstate(forward calls) = fulfilled)} \Rightarrow \\
((\text{SDstate(mobile) = fulfilled}) \lor \text{(SDstate(email) = fulfilled)})
\]
State invariants are associated with the state variables representing “Healthiness conditions” which must be always satisfied. Inconsistencies may arise in the formal specifications due to a contradiction in the state invariant. Other possible source of inconsistencies are the violation of the state invariants. These different kinds of inconsistencies are easy to check and correct using formal notations like Z. For example, if an additional invariant is accidentally introduced that states that “Positive or negative contributions are possible only for goals”, then due to this invariant, there will be contradiction in the invariants of the Link Schema (Section 4), which will lead to inconsistencies in the specifications. These cases can be easily detected using formal methods. The other positive aspect is that when we use state invariants in the Z schema, we are making specifications precise and formal, which are not amenable to inconsistencies and its easy to detect any inconsistencies in the specifications.

Existing tool support for Z, on the other hand, allows analysis of specifications without any additional effort. By making use of formal notation like Z to formalise the \( i \) diagrams, we are using the customary facilities available for Z like:

- type checking the components
- proving properties in relation to the components and
- providing precise rules for manipulating the components

For realising above-mentioned objectives, various tools for formatting, type-checking and aiding proofs in Z are available. We are listing some of them that might be used:

- CADiZ [16], which is a UNIX based tool for checking and typesetting Z specifications.
- The WYSIWYG editor Zola, which supports the production and typesetting of Z specifications.
- Z type checkers like ZTC [14].
- Z animation tools like ZANS [15].
- Formaliser [11], a syntax-directed Z editor as well as an interactive type-checker, running under Microsoft Windows obtainable from Logica.
- Z/EVES [22], which supports the analysis of Z specifications in several ways: for syntax and type checking, schema expansion, and precondition calculation.

Reasoning about the internal consistency of a formal specification written in Z is conducted primarily from the software developers point of view. We are aware that inconsistencies in the specifications may arise from an ill conceived problem specification resulting from the lack of understanding of the object of specification and incorrect formalisation.

9. Conclusion

We have shown in this work how two otherwise disparate approaches (agent-oriented conceptual modelling and formal methods) might be used in a complementary and synergistic fashion. This approach makes use of the advantages of \( i \) for the early-phase of requirements engineering (visualisation of requirements, possibility of easy modifications, etc.) and then continues with the specification of requirements in Z. The Z notation permits us to specify requirements with a degree of precision and formality that \( i \) does not. 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 shown 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). We also propose a hybrid modelling, or co-evolution, approach which permits \( i \) modelling to proceed independently of specification in a distinct notation (Z here), while maintaining some modicum of loose coupling via consistency constraints. More generally, this research suggests how diagrammatic notations for modelling early-phase requirements, organisational contexts and rationale can be used in a complementary manner with more traditional specification notations. 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. At the same time the proposed methodology needs to be validated with respect to other approaches, and tool support should be provided. This also remains focus of our future work.

References


