Loosely-coupled consistency between agent-oriented conceptual models and Z specifications

Aneesh Krishna
University of Wollongong, aneesh@uow.edu.au

Aditya K. Ghose
University of Wollongong, aditya@uow.edu.au

Sergiy A. Viklomir
University of Limerick

Publication Details
Loosely-coupled consistency between agent-oriented conceptual models and Z specifications

Abstract
Agent-oriented conceptual modelling (AOCM) is a relatively new technique that offers significant benefits in the modelling and development of complex computer systems. It is highly effective in answering questions such that what are the main goals of the system, how key actors depend on each other, and what alternatives exist. A formal method can benefit any stage of the software development lifecycle and improves the quality of the computer systems. The paper defines an approach that allows to complement requirements modelling notations with formal specifications, while preserving the consistency between them.

Disciplines
Physical Sciences and Mathematics

Publication Details

This conference paper is available at Research Online: http://ro.uow.edu.au/infopapers/2824
“Loosely-coupled” Consistency between Agent-Oriented Conceptual Models and Z Specifications

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) is a relatively new technique that offers significant benefits in the modelling and development of complex computer systems. It is highly effective in answering questions such that what are the main goals of the system, how key actors depend on each other, and what alternatives exist. A formal method can benefit any stage of the software development lifecycle and improves the quality of the computer systems. The paper defines an approach that allows to complement requirements modelling notations with formal specifications, while preserving the consistency between them.

1. Introduction
Many existing modelling techniques and frameworks tend to address the “late-phase” of requirements engineering, which focuses on completeness, consistency and automated verification of the requirements [12], while the vast majority of critical modelling decisions (such as determining the main goals of the system, how the stakeholders depend on each other, and what alternatives exist [12]) are taken in the early-phase requirements engineering. Hence, it would be appropriate to present different modelling and reasoning support for the two phases. The i* modelling framework [12] 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) [12] 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 indicates that one actor depends on the other for something in order that the former may attain some goal. An SR model (see Figure 1) provides a more detailed level of modelling by looking “inside” actors to model internal intentional relationships. Intentional elements (goals, tasks, resources, and softgoals) appear in the SR model not only as external dependencies, but also as internal elements linked by task-decomposition and means-ends relationships. Readers are encouraged to refer to [12] for a comprehensive explanation of the i* framework. Consider the following modified example (see Figure 1)(to be used throughout the rest of the paper) from our earlier case study [9] which concentrates on a key function of the emergency services agency (ESA): computer based training system (CBT) for volunteers. This research has been conducted in the context of a larger project to deploy i* for enterprise modelling in a large ESA.

There have been a number of proposals reported in the literature for combining i* modelling with late-phase requirements analysis and the downstream stages of the software life-cycle. One of them combines the i* framework with the formal agent programming language [11]. We have similar objectives with a slightly 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. We believe that, the i* framework when used in conjunction with other modeling/specification in notation X (X could be UML/Z/English) improves the quality of those models/specifications. Our work focuses on the combined use of agent-oriented conceptual modeling and Z notation. The notion of co-evolution is used in a very specific sense to describe a class of methodologies that permits the i* modeling to proceed independently of specification in a distinct notation, while maintaining some modicum of loose coupling via consistency constraints. Our research suggests how diagrammatic notations suitable for model-
ing the requirements; organisational contexts and rationale can be used in a complementary manner with more traditional specification notations (in our case Z, may be UML).

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.

In Section 2, below, we present the mapping methodology between i* models and Z schemas. Section 3 introduces a methodology for supporting the co-evolution of i* models and Z specifications. Section 4 discusses how consistency is preserved during the co-evolution of i* models and Z specifications. Finally, Section 5 presents concluding remarks.

2. i* to Z Transformation

The first step in defining a co-evolution methodology for i* and Z is to define a mapping from i* to Z. We shall be presenting results from our earlier work [10, 7, 6] which has been modified and extended.

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, satisfied, 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 an 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.).

\[
SD = \{\text{SD}_\text{state} : \text{NAME} \rightarrow \text{STATE} \}\]

\[
\text{dom} \text{SD}_\text{state} = \text{all} \text{depend}
\]

\[
\text{Actor}
\]

\[
\begin{align*}
\text{actor} \text{name} : \text{NAME} \\
\text{actor} \text{element} : \mathbb{P}_1 \text{NAME} \\
\text{actor} \text{state} : \text{NAME} \rightarrow \text{STATE}
\end{align*}
\]

\[
\text{dom} \text{actor} \text{state} = \text{all_actors}
\]

As a common pattern for SD dependencies and SR elements, the schema \(\Phi \text{Depend} [10, 7]\) is used (the \(\Phi\) in the schema name is used to flag a partial specification [8]). This schema is an operation schema and changes the state of the SD model \(\Delta \text{SD}\). \(\text{SDependency}\) schema includes the components \(\Phi \text{Depend}\) schema as well as names of actors (dependee and dependee) which are linked by the dependency. This schema also includes the names of the internal elements (dependee internal and dependee internal element) linked to the dependency. The sets actor element dependee and actor element dependee are the names of all the internal elements present in the dependee and dependee respectively. 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.

\[
\begin{align*}
\Phi \text{Depend} \\
\text{dependee, dependee internal element} : \text{NAME} \\
\text{actor element dependee} : \mathbb{P}_1 \text{NAME} \\
\text{dependee internal element} \in \text{actor element dependee} \\
\text{dependee internal element} \in \text{actor element dependee}
\end{align*}
\]

\[
\text{SD}_\text{state}' = \text{SD}_\text{state} \oplus \{\text{dependum} \mapsto \text{result!}\}
\]

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.

\[
\text{Link}
\]

\[
\begin{align*}
\Phi \text{Depend} \\
\text{int components, ext components} : \mathbb{P} \text{NAME} \\
\text{contrib}_p, \text{contrib}_n : \mathbb{P} \text{NAME} \\
\text{link} : \text{LINK_TYPE}
\end{align*}
\]

\[
\begin{align*}
\text{link} = \text{task decomp} & \Rightarrow \text{type} = \text{task} \\
\text{link} = \text{contrib} & \Rightarrow \text{type} = \text{softgoal} \\
\text{link} \land \text{contrib}_p \cup \text{contrib}_n \neq \emptyset & \Rightarrow \text{link} = \text{contrib} \land \{\text{contrib}_p, \text{contrib}_n\} \text{ partitions int components} \\
\text{ext components} \neq \emptyset & \Rightarrow \text{type} = \text{task} \\
\text{link} = \text{NA} & \Rightarrow \text{int components} = \emptyset
\end{align*}
\]
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 the Z specification. We shall refer to these basic schemas as model schemas.

The next step in our methodology is the mapping of specific i* model into Z schemas. Following steps are carried out to realise this goal: i) Names of all the actors and external dependencies are specified. This is the first step in mapping the SD model of the CBT system. ii) The second step in the mapping is based on the creation of Z schema for every dependency using SDependency schema as a basis. iii) The first step in mapping the SR model is to specify the names of all the internal intentional elements of the selected actor. iv) The second step is the creation of a Z schema for every internal intentional element using AElement schema as a basis. Schemas for actors, dependencies, actor internal intentional elements and the links between them in a specific i* model are defined using these model schemas - we shall call these as element schemas. Considerable detail has been omitted in this section due to space limitations, but examples and full versions of the schemas described can be found in [10, 7].

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 - e.g., temporal sequencing of dependencies, fulfillment 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 proposed methodology permits the maintenance of loose coupling between an i* model and Z specification (refer to Figure 2). The strategy we have adopted is to localize the impact of changes. The idea is to look at two specific points:

- explain 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).
- explain 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).

It is worth mentioning here that changes in the i* model only affect the element schemas, but not the model schemas. The solution to the first of the identified questions (i.e. obtaining an unrefined Z specification from the modified i* model) is obtained by defining techniques that require reference to the prior i* model and the corresponding prior unrefined Z specification. 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 discuss 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 soft-goal contribution link to an existing soft-goal. 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 for 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 words, 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 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 the 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.

The solution to the second of the identified question (i.e. 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) is obtained by identifying 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. Our aim is to reflect the refinements in the 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.

We shall now present an illustration to explain the methodology supporting the co-evolution of i* and Z. This example is based on the CBT system case study. The following modifications/additions were performed on the initial i* diagram: Introducing a task Let Training Co-ordinator Schedule Training into the SR model of the actor Training Co-ordinator will lead to the modification of the original i* diagram (consider that initially this task does not exist in the model) and creation of an ad-
4. 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 one. We believe that these semantics are largely consistent with the somewhat implicit semantics for i* developed in [12]. 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 modeler 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 already has richer semantics. We believe that these semantics are largely consistent with the somewhat implicit semantics for i* developed in [12]. Semantic inconsistencies 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.

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 formalize the i* diagrams, we are using the customary facilities available for Z like: i) type checking the components ii) proving properties in relation to the components and iii) 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. First of them is CADiZ [5], which is a UNIX based tool for checking and typesetting Z specifications. Zola the WYSIWYG editor is another interesting tool, which supports the production and typesetting of Z specifications. Also included are a type-checker and a Tactical Proof System (available from http://www.ist.co.uk/PRODUCTS/zola.html). The integration of i* diagrams and Z allows one to use Z type checkers like ZTC [3] and Z animation tools like ZANS [4] to analyse the models. It is projected to be compliant with the second edition of Spivey’s Z reference manual. Formaliser [1] is a syntax-directed Z editor as well as an interactive type-checker, running under Microsoft Windows obtainable from Logica.

5. Conclusion

We presented a methodology to support the complementary use of an early-phase requirements modeling notation such as i* with formal specifications, in this instance Z. The issue of preserving consistency in the co-evolution of formal and informal models was discussed in this work. We have not investigated the possibility of articulating semantic consistency constraints between i* models (possibly augmented with FormalTropos annotations)[2] and formal specifications. This is the focus of our future work.

References