Capturing Contextual Variability in *i** Models

Alexei Lapouchnian[1] and John Mylopoulos[2]

[1] Department of Computer Science, University of Toronto, Canada
alexei@cs.toronto.edu
[2] Department of Information Engineering and Computer Science, University of Trento, Italy
jm@disi.unitn.it

Abstract. Exploration and analysis of alternatives is one of the main activities in requirements engineering, both in early and in late requirements phases. While *i** and *i**-derived modeling notations provide facilities for capturing certain types of variability, domain properties (and other external influences) and their effects on *i** models cannot be easily modeled. In this paper, we propose to explore how our previous work on context-dependent goal models can be extended to support *i**. Here, we examine how *i** modeling can benefit from both monitorable (i.e., defined through real-world phenomena) and non-monitorable (e.g., viewpoints, versions, etc.) contexts defined using our context framework.

1 Introduction

*i** is an agent-oriented modeling framework that centers on the notions of intelligent actor and intentional dependency. The Strategic Dependency (SD) model of *i** focuses on representing the relevant actors in the organization together with their intentional dependencies, while the Strategic Rationale (SR) model captures the rationale behind the processes in organizations from the point of view of participating actors.

Variability in requirements and in design has been identified as crucial for developing future software systems [4,5]. Moreover, flexible, robust, adaptive, mobile and pervasive applications are expected to account for the properties of (as well as to adapt to changes in) their environments. Thus, modeling variability in the system environment and its effects on requirements and on other types of models is a highly desirable feature of a modeling framework. However, *i** does not support capturing of how domain variations affect its diagrams. This leads to two situations. First, an oversimplification of the diagrams through the assumption of domain uniformity with the hope of producing an *i** model that is adequate for the most instances of a problem. Second, the production of multiple *i** models to accommodate all domain variations. The former case leads to models that fail to account for the richness of domain variations, while the latter introduces serious model management problems due to the need to oversee large numbers of models as well as to capture their relationships. In this paper, we adapt the ideas from [3] to the *i** modeling framework and propose an approach that uses *contexts* to structure domain variability and to concisely represent and analyze the variations in *i** models resulting from this domain variability as well as from other external factors such as viewpoints, etc. Using the proposed approach,

we are able to capture in a single context-parameterized model how varying domain characteristics affect stakeholders, their goals, and their intentional dependencies.

2 Research Objectives

While *i** has capabilities to represent certain types of variations in it diagrams, they are not adequate to capture the effects of domain variability on the models. In SD models, one cannot state that the actors and dependencies appear in the model only in certain circumstances. E.g., if we look at a system where a Distributor accepts orders from Customers, fulfills them through Suppliers, and then ships those orders through Shipping companies, it is not possible to capture in a single SD diagram the fact that in the case of an international order, another actor comes in, the Customs Broker, through which the Distributor clears the order before shipping it.

In SR models, OR decompositions (or means-ends links) are the tools to represent variation points. Still, they are not enough to capture all the possible effects that domain variations can have on SR models, such as varying sets of top-level actor goals, different goal refinements, and changing evaluations of alternatives w.r.t. softgoals.

We propose to adapt the ideas of [3] to *i** and use contexts as a way to parameterize *i** models in order identify changes due to such external factors. A *context* is an abstraction over relevant domain properties. *internationalOrder*, *largeOrder*, *importantCustomer* are examples of contexts that influence the *i** diagrams modeling the Distributor system. Additionally, we show how related contexts can be organized into inheritance hierarchies and how this simplifies the modeling process as well as discuss the notion of visibility of model elements as a way to combine model variants.

3 The Context Framework for *i**

3.1 Visibility of Model Elements, Contexts, and Context Inheritance

The main idea of our context framework [3] is that models (e.g., ER and *i** diagrams, or knowledge bases) are viewed as collections of elements (i.e ., nodes, edged, facts), some associated with conditions that describe when the elements are *visible* – i.e., present in the model. These conditions are captured through (possibly many) sets of *contextual tags* assigned to model elements. The tags model the (many) *contexts* in which the elements are *valid*. E.g., the tag assignment *{{largeOrder}, {mediumOrder, importantCustomer}}* indicates that some model element is visible either when the order is large or with a medium-sized order from an important customer. The absence of any condition indicates that a model element is valid in all contexts (visible in all model variants). Each contextual tag has a definition describing when it is *active*. Thus, a set of tags can be viewed as a propositional DNF formula. Through their definitions, contexts can be monitored in the environment of the system to determine when they are active. Therefore, a context-parameterized model will be changing as the environment conditions change. In [3], which presents the details of this formal visibility framework, we applied the framework to goal models, while here we do so

for *i**. We are interested in capturing the effects of two types of external factors on *i** models: monitorable contexts and changes due to viewpoints, model versions, etc.

For added flexibility, the formal context framework supports non-monotonic inheritance of contextual tags. This way, the modeler can declare that a new contextual tag (e.g., *mediumOrder*) inherits from an existing one (e.g., *substantialOrder)*. Thus, model elements tagged with *mediumOrder* (i.e., valid/visible in that context) are automatically tagged with *substantialOrder*. Additional model elements can be explicitly assigned the derived tag, while others, to which the parent tag had been previously applied, can be excluded from the derived tag (hence the non-monotonicity of the inheritance). This mechanism is a means for structuring the domain and supports incremental development of context-dependent models through tag reuse.

The framework states that an element is visible in the model if the DNF formula derived by substituting contextual tags with their definitions (and also taking into account contextual tag inheritance) holds. So, given a domain in some state, we evaluate the tag definitions to determine which ones are active and then conclude which model elements are visible *in the current domain state*. Since this framework is model-agnostic and does not take into consideration the syntax/semantics of the *i** modeling framework, we need to create a method to process context-parameterized *i** models and produce, for each model element, the expression that determines its visibility.

3.2 Applying the Framework to *i** Models

To apply the above-described contextual framework to *i**, we need to associate contextual constraints to *i** model elements (actors, goals, dependencies, and so on). We use *contextual annotations* to specify that certain *i** model elements are only visible in particular contexts, thus taking domain variability into account. Once these annotations are applied to SD/SR diagrams, an algorithm similar to the one presented in [3] for goal models will process these diagrams and the context hierarchies that accompany them, propagate appropriate contextual tags (see below) and generate for each model element a contextual tag expression defining when they are visible. Due to space constraints, we do not present the algorithm in this paper.

In SD diagrams, actor nodes and dependencies can both be parameterized with contextual annotations. For instance**,** Fig. 1A shows that the Customs Broker agent and its incoming dependency from Distributor are only visible in the context of international orders (we are using a simplified form of contextual annotations compared to [3]).

To avoid dangling dependencies, if there is a dependency Dep from actor A1 to actor A2, and these are parameterized with context annotations C_D, C_{A1}, and C_{A2} respectively (Fig. 1B), then the dependency visibility is defined by the conjunction of these annotations since for a dependency to appear in the model, its own context and the contexts for its source and destination actors have to be active (they have to be in the model). This illustrates that one of the annotations in Fig. 1A is, in fact, redundant. Overall, we are utilizing the hierarchical nature of *i** (i.e., the decompositions) as well as its navigational rules (through dependencies) to minimize the number and size of contextual annotations that are needed. So, for the dependency in Fig. 1B, the complete visibility constraint can be automatically generated from up to three annotations.

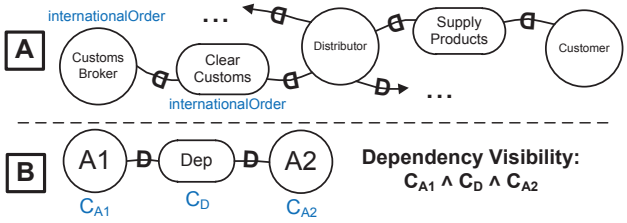

Fig. 1. Applying contexts to SD models

In [3], we described how contextual annotations could be applied to goal models, which are, in essence, actor-less, SR models. The main idea was that a contextual annotation applied to a (soft)goal node is automatically propagated to the subtree rooted at that node (i.e., its refinement). This greatly reduces the number of annotations one needs to apply. Moreover, multiple annotations within the same subtree are combined in a way shown in Fig. 2A: when the annotation C_2 is applied to a node G1 within the subtree already adorned by C_1, both contexts must be active for G1 and its descendants to be visible in the model. For context-parameterized SR models, resource and task nodes are handled similarly. Annotations can also be applied to softgoal contribution links to capture the fact that the evaluation of alternatives can be different in different contexts. E.g., the automatic approval of orders may have a negative contribution to the softgoal Minimize Risk (Fig. 2B) for low-risk customers, but in the case of a high-risk one, the contribution is changed to *break*.

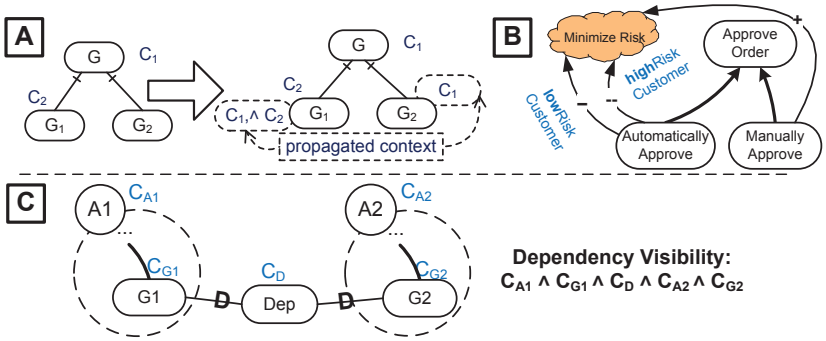

Fig. 2. Context propagation (A); contextual annotations applied to contribution links (B); (C) Context-parameterized dependencies in SR diagrams

The key additions to SR diagrams compared to goal models of [3] are actor bubbles and dependencies. Since some actors may be present only in certain contexts (e.g., the Customs Broker in the context of an international order), in SR models they too can be annotated with contexts. Such annotations are implicitly applied to all the model elements within that actor's bubble. The actor's context will be combined with other context annotations within the bubble as shown in Fig. 2A. This way, whenever the context associated with the actor node is not active, the whole bubble disappears.

When looking at a context-parameterized intentional dependency at the SR level, we treat its edges together with the dependum as a single model element parameter-

ized with a context. The contexts for its source node, its target node, and the dependency itself must be active for the dependency to be visible in the model (see Fig. 2C).

3.3 Using Context-Parameterized *i** models

With the above approach, we produce a context-parameterized *i** model, in which each model element is associated with a visibility condition (this, in fact, combines into a single model many different model variations). Evaluating these conditions in some domain state produces a model variant with only a subset of the elements. Thus, in effect, contexts act as filters on *i** models by removing irrelevant model fragments.

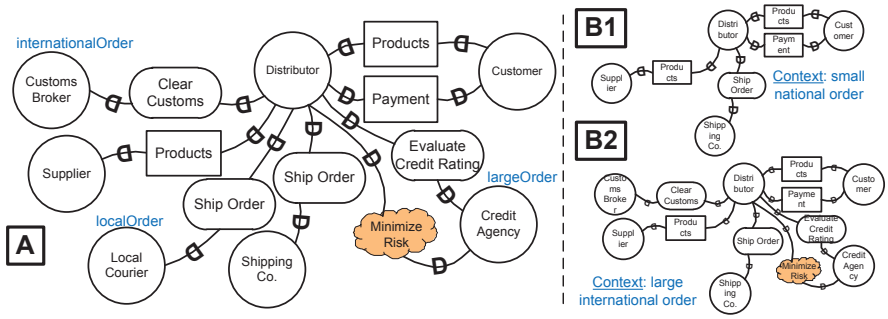

Fig. 3. Using contexts as filters on SD models

Given the context definitions that are rooted in real-world phenomena (i.e., *monitorable contexts*), *i** models can be dynamically adapted to changing domain conditions. E.g., the context-parameterized SD model for the distributor scenario in Fig. 3A is seen differently in the context of a small national order (Fig. 3B1) compared to the context of a large international order (Fig. 3B2). Note that these two models are no longer parameterized with contexts – their contextual variability has been *bound*. Thus, they can be analyzed using conventional goal model or *i** analysis techniques.

The same framework can be used to integrate a number of variations of the same model, each capturing a particular model version, a viewpoint, etc. In this case, viewpoints, versions, etc. are represented by contexts that are not monitored, but can be manually turned on or off (e.g., *version1 = true*). These contexts can also be organized into inheritance hierarchies to better facilitate incremental model development and even to mimic approaches like [2]. The idea of context-based visibility can likewise be employed to label alternative sets of leaf-level nodes in SR models (i.e., *strategies* for achieving high-level goals) with contexts, and then turning these contexts "on" or "off", and running analysis algorithms on the resulting model variations.

4 Ongoing and Future Work

We are working on the flexible implementation of the context framework to support various flavours of *i**-based and goal modeling notations. This will also help with the validation of the approach. In addition, we are exploring the links between

contexts and business rules. We also plan to identify synergies with the approach of [1], which while having a lot of similarities with our proposal, has a different focus.

We currently view contexts as global and thus shared among actors. However, in some applications, it may be beneficial model contexts on per-actor basis, which implies that in that case, the context-parameterized models would capture the actors' possibly incompatible viewpoints on the system. Also, there may be flavours of *i** modeling, which do not comply with the handling of dependencies in SR models illustrated in Fig. 2C. We are looking into a number of context propagation customizations to accommodate these modeling techniques.

Contexts can be thought of as specifying dynamically loadable model fragments. When the formula defining the context holds, the context becomes active and the model elements that are "in" context become visible in (or loaded into) the model. We plan to explore the idea of context encapsulation as a way to improve scalability in *i**.

5 Conclusions

In this paper, we applied to *i** a context mechanism based on the flexible idea of model elements' visibility defined by external factors such as domain characteristics, viewpoints, etc. With the proposed framework, we are able to capture the effects of domain variability on a system using a single context-parameterized *i** model and to automatically produce variations of that model based on the currently active contexts. In essence, contexts here act as filters on *i** models by removing model elements not applicable in the current state of the domain. Contextual variability is thus bound, so conventional goal analysis techniques can be utilized with the generated models. The framework can be used for both monitorable and non-monitorable contexts. Context inheritance is another feature of the approach. It allows for adding structure to domain models, for context reuse, and for incremental *i** model development.

The context framework's aim is not to address the scalability/complexity issues in *i** – it is to help integrate multiple model variations into a single diagram. Here, one has to balance the need for and the benefits of adjusting the model to different domain characteristics and thus the need to have a large number of system variations against the complexity of eliciting and maintaining this large number of system variants.

References

1. R. Ali, F. Dalpiaz, P. Giorgini. A Goal-based Framework for Contextual Requirements Modeling and Analysis. REJ, 15(4):439-459, 2010.
2. S. M. Easterbrook. Domain Modelling with Hierarchies of Alternative Viewpoints. In Proc. *RE'93*, San Diego, January 1993.
3. A. Lapouchnian and J. Mylopoulos. Modeling Domain Variability in Requirements Engineering with Contexts. In Proc. *ER 2009*, Gramado, Brazil, Nov 9-12, 2009.
4. A. Lapouchnian, Y. Yu, S. Liaskos, J. Mylopoulos. Requirements-Driven Design of Autonomic Application Software. In Proc. *CASCON 2006*, Toronto, Canada, Oct 16–19, 2006.
5. S. Liaskos, A. Lapouchnian, Y. Yu, E. Yu, J. Mylopoulos. On Goal-based Variability Acquisition and Analysis. In Proc. *RE'06*, Minneapolis, USA, Sep 11-15, 2006.