Modeling Domain Variability in Requirements Engineering with Contexts

Alexei Lapouchnian and John Mylopoulos

Department of Computer Science, University of Toronto,
Toronto, ON M5S 3G4, Canada
{alexei,jm}@cs.toronto.edu

Abstract. Various characteristics of the problem domain define the context in which the system is to operate and thus impact heavily on its requirements. However, most requirements specifications do not consider contextual properties and few modeling notations explicitly specify how domain variability affects the requirements. In this paper, we propose an approach for using contexts to model domain variability in goal models. We discuss the modeling of contexts, the specification of their effects on system goals, and the analysis of goal models with contextual variability. The approach is illustrated with a case study.

1 Introduction

Domain models constitute an important aspect of requirements engineering (RE) for they constrain the space of possible solutions to a given set of requirements and even impact on the very definition of these requirements. In spite of that, domain models and requirements models have generally been treated in isolation by requirements engineering approaches (e.g., [7]). As software systems are being used in ever more diverse and dynamic environments where they have to routinely and efficiently adapt to changing environmental conditions, their designs must support *high variability* in the behaviours they prescribe.

Not surprisingly, high variability in requirements and design have been recognized as cornerstones in meeting the demands for software systems of the future [14,11,16]. However, the variability of domain models, which captures the changing, dynamic nature of operational environments for software systems, and its impact on software requirements, has not received equal attention in the literature. The problem is that traditional goal models assume that the environment of the system-to-be is mostly uniform and attempt to elicit and refine system goals in a way that would make the goal model adequate for most instances of a problem (e.g., selling goods, scheduling meetings, etc.) in a particular domain. In other words, traditional techniques ignore the impact of domain variability on the requirements to be fulfilled for a system-to-be. Thus, *these approaches are missing an important source of requirements variability*.

A recent proposal [17] did identify the importance of domain variability on requirements. However, it assumes that requirements are given, and concentrates on making sure that they are met in every context. Thus, the approach does not explore the effects of domain variability on intentional variability – the variability in stakeholder goals and their refinements. Also, in pervasive and mobile computing, where

contexts have long been an important research topic, a lot of effort has been directed at modeling various contexts (e.g., [9]), but little research is available on linking those models with software requirements [10].

In a recent paper [14], we concentrate on capturing *intentional* variability in early requirements using goal models. There, the main focus was on identifying all the ways stakeholder goals can be attained. We pointed out that non-intentional variability (that is, time, location, characteristics of stakeholders, entities in the environment, etc.) is an important factor in goal modeling as it constrains intentional variability in a significant way. However, we stopped short of systematically characterizing such domain variability and its effects on requirements. To that end, in this paper, we propose a coherent process for exploring domain/contextual variability and for modeling and analyzing its effects on requirements goal models. We propose a fine-grained model of context that represents the domain state and where each context contains partial requirements model representing the effects of that context on the model. Unlike, e.g., the method of [17], our approach results in *high-variability context-enriched goal models* that capture and refine stakeholder goals in *all* relevant contexts. Moreover, context refinement hierarchies and context inheritance allow incremental definition of the effects of contexts on goal models specified relative to higher-level contexts. These goal models can then be formally analyzed.

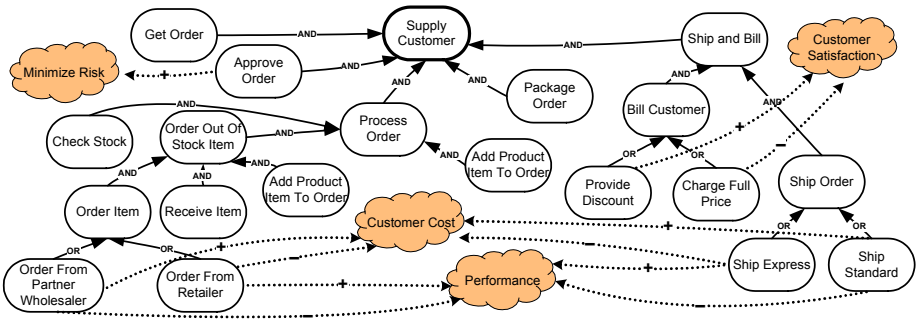

Fig. 1. A high-level goal model for the Distributor.

As a motivation for this research, let us look at system for supplying customers with goods. At a first glance, it seems that gathering requirements for such a system is rather straightforward: we have the domain consisting of the distributor, the customers, the goods, the orders, the shipping companies, etc. Following a goal-oriented RE approach, we can identify the functional goals that the system needs to achieve (e.g., Supply Customer, see Fig. 1) and the relevant softgoals/quality constraints (the cloudy shapes) like Minimize Risk and then refine them into subgoals until they are simple enough to be achieved by software components and/or humans. The produced requirements specification assumes that the domain is uniform – i.e., the specification and thus the system will work for *all* customers, *all* orders, etc. However, it is easy to see that this view is overly simplistic as it ignores the variations in the domain that have important effects on system requirements. E.g., international orders need to have customs paperwork filled out, while domestic orders do not. Large orders are good for business, so they may be encouraged by discounts or free shipping. And the list goes

on. So, our aim in this paper is to introduce an approach that allows us to model these and other effects of domain non-uniformity and variability on software requirements.

The rest of the paper is structured as follows. Section 2 is the research baseline for this work, covering context, goal modeling and related work. Section 3 presents our formal framework. Section 4 talks about context-dependent goal models. Discussion and future work are presented in Section 5, while Section 6 concludes the paper.

2 Background and Related Work

There exist a lot of definitions of context in Computer Science. E.g., [4] defines context as "any information that can be used to characterize persons, places or objects that are considered relevant to the interactions between a user and an application, including users and applications themselves". Brezillon [2], says that "context is what constrains problem solving without intervening in it explicitly". McCarthy states that "context is a generalization of a collection of assumptions" [15]. This definition fits well with our treatment of context as properties of entities in the environment and of the environment itself that influence stakeholder goals and means of achieving them. Thus, we define a context as an abstraction over a set of environment assumptions.

In various areas of computing, the notion of context has long been recognized as important. For example, in the case of context-aware computing the problem is to adapt to changing computational capabilities as well as to user behaviour and preferences [4]. In pervasive computing, context is used to model environment and user characteristics as well as to proactively infer new knowledge about users.

There have been quite a few recent efforts directed at context modeling. While some approaches adapt existing modeling techniques, others propose new or significantly modified notations. Henricksen and Indulska present their Context Modeling Language (CML) notation [9]. Their graphical modeling notation allows for capturing of fact types (e.g., *Located At*, *Engaged In*) that relate object types (e.g., location, person, and device). The model can distinguish between static and dynamic facts. Moreover, it classifies dynamic facts into profiled facts (supplied by users), sensed facts (provided by sensors), and derived facts (derived from other facts through logical formulas). Dependencies among facts can also be specified. A special temporal fact type can be used to capture time-varying facts. Additional features of the approach include, for example, support for ambiguous context as well as for context quality (e.g., certainty).

Standard modeling approaches like UML and ER have been used for context modeling. However, they are not well suited for capturing certain special characteristics of contextual information [9]. For example, in [6], UML class diagrams are used to model user, personalization, and context metadata subschemas together in one model. Ontologies have are also used for context modeling. They provide extensibility, flexibility and composability for contexts. In [4], a generic top ontology, which can be augmented with domain-dependent ones, is proposed. These approaches do not focus on the use of context in applications.

Much research has also been dedicated to the formal handling of contexts in the area of Artificial Intelligence and Knowledge Representation and Reasoning [1].

Goal models. Goal models [5,7] are a way to capture and refine stakeholder intentions to generate functional and non-functional requirements. The main concept there

is the *goal*, such as Supply Customer for a distributor company (Fig. 1). Goals may be AND/OR decomposed. For example, Supply Customer is AND-decomposed into subgoals for getting customer orders, then processing and shipping them. All of these subgoals need to be achieved for the parent goal to be achieved. On the other hand, one or more subgoals in an OR decomposition need to be achieved for the parent goal to be attained (e.g., achieving either Ship Standard or Ship Express will satisfy Ship Order). OR decompositions thus introduce variability into the model. *Softgoals* are qualitative goals (e.g., [Maximize] Customer Satisfaction). Softgoals do not have a clear-cut criterion for their fulfillment, and may be *satisficed* – met to an acceptable degree. In addition, goals/softgoals can be related to softgoals through help (+), hurt (–), make (++), or break (--) relationships (represented with the dotted line arrows in Fig. 1). These *contribution links* allow us to *qualitatively* specify that there is evidence that certain goals/softgoals contribute positively or negatively to the satisficing of softgoals. Then, a softgoal is satisficed if there is sufficient positive and little negative evidence for this claim. This simple language is sufficient for modeling and analyzing goals during early requirements, covering both functional and quality requirements, which in this framework are treated as first-class citizens.

High-variability goal models attempt to capture many different ways goals can be met in order to facilitate in designing flexible, adaptive, or customizable software [11,12,14]. In [14], an approach for systematic development of high-variability goal models is presented. The approach, however, does not cover domain variability and its affect on requirements.

Related Work. Our view of contexts is somewhat similar to the CYC common sense knowledge base [13]. CYC has 12 context dimensions along which contexts vary. Each region in this 12-dimensional space implicitly defines an overall context for CYC assertions. We, however, propose more fine-grained handling of context with possibly many more domain-specific dimensions.

Brezillon et al. [3] propose an approach for modeling the effects of context on decision making using contextual graphs. Contextual graphs are based on decision trees with event nodes becoming context ones. They are used to capture various context-dependent ways of executing procedures. Whenever some context plays a role in the procedure, it splits it up into branches according to the value of the context. Branches may recombine later. This work is close to our approach in the sense that it attempts to capture all the effects of contextual variability in one model. However, we are doing this at the intentional level, while contextual graphs is a process-level notation. Moreover, quality attributes are not considered there. We have looked at generating process-level specifications from goal models [12] and we believe that contextual graphs can be generated from the context-enriched goal models as well.

Salifu et al. [17] suggest a Problem Frames-based approach for the modeling of domain variability and for the specification of monitoring and switching requirements. They identify domain variability (modeled by a set of domain variables) using variant problem frames and try to assess its impact on requirements. For each context that causes the requirements to fail, a variant frame is created and analyzed in order to ensure the satisfaction of requirements. This approach differs from ours in that it assumes that the requirements specification is given, while we are concentrating on activities that precede its formulation. Another substantial difference is that we pro-

pose the use of a single high-variability requirements goal model for capturing all of the domain's variability.

3 The Formal Framework

In this section, we present a formal framework for managing models through the use of contexts. While we are mainly interested in the graphical models such as requirements goal models, our approach equally applies to any type of model, e.g., formal theories. We view instances of models (e.g., the Supply Customer goal model) as collections of model element instances (e.g., Ship Order). There may be other important structural properties of models that need capturing, but we are chiefly concerned with the *ability to model under which circumstances certain model elements are present (i.e., visible) in the model* and with the ability to display a version of the model for the particular set of circumstances. Thus, we are concerned with capturing *model variability* due to a wide variety of *external factors*. These factors can include viewpoints, model versions, domain assumptions, etc. This formal framework can be instantiated for any model to help with managing this kind of variability. In section 4.3, we present an algorithm that generates this formal framework given an instance of a requirements goal model.

We assume that there are different types of elements in a modeling notation. For example, in graphical models, we have various types of nodes and links among them. Let **M** be the set of model element instances in a model. Let **T** be the set of various model element types available in a modeling notation (e.g., goals, softgoals, etc.). The function *L* maps each element of **M** into an element of **T**, thus associating a type with every model element instance. Only certain types of elements in a modeling notation may be affected by contexts and thus belong to a variable part of a model. We define $\mathbf{T}^{\mathbf{C}}$ as the subset of **T** containing such *context-dependent* model element types. If a model element type is not in $\mathbf{T}^{\mathbf{C}}$, it is excluded from our formalization. The contents of the $\mathbf{T}^{\mathbf{C}}$ set are notation- and model-dependent. Let $\boldsymbol{M^C} \stackrel{\text{def}}{=} \{n |\, n \in \boldsymbol{M} \wedge L(n) \in \boldsymbol{T^C}\}$ be the set of modeling elements of the types that can be affected by contexts.

We next define the set **C** of *contextual tags*. These are labels that are assigned to model elements to capture the conditions that those elements require to be visible in the model. To properly define what contextual tags model, we assign each tag a Boolean expression that specifies when the tag is *active*. Since the tags represent domain properties, assumptions, etc., the associated expressions precisely define when the contextual tags affect the model and when they are not (we define **P** to be the set of Boolean expressions): $active: \boldsymbol{C} \rightarrow \boldsymbol{P}$**.** For example, the tag *largeOrder* describes a real world entity and may be defined as an order with the sum being over $10K. So, when some order *is* over $10K, the tag becomes active and thus can affect the model. The approach can also be used to capture viewpoints, model versions, etc. In those cases, the definition of tags can be simple: they can be turned on and off depending on what the modeler is interested in (e.g., *versionOne* = *true*). We also allow negated tags to be used in the approach: $(\forall t \in \boldsymbol{C})\ active(\neg \mathrm{t}) \stackrel{\text{def}}{=} \neg active(t)$.

We define a special *default* tag that is always active and if assigned to an element of a model signifies that the element does not require any assumptions to hold to be present in the model. To associate tags with model elements we create a special unit called *taggedElement* ($\wp$ is a powerset): $taggedElement \subseteq \boldsymbol{M^c} \times \wp(\wp(\boldsymbol{C}))$.

To each element of $\mathbf{M}^{\mathbf{C}}$ we assign possibly many tag combinations (sets of tags). E.g., the set *{{a,b},{c,d}}* assigned to an element *n* specifies that *n* appears in the model in two cases: when both *a* and *b* are active or when both *c* and *d* are active. The outer set is the set of alternative tag assignments, either of which is enough for the element to be visible. In fact, the above set can be interpreted as $(a \wedge b) \vee (c \wedge d)$, so our set of set of tags can be viewed as a propositional DNF formula.

The function *newTaggedElement* creates a new tagged element entity given a model element and a set of tags. It can be called from within algorithms that process input models for which we want to use the formal framework. Given a model element, the function *tags* returns the set of contextual tags of a *taggedElement*. In order to eliminate possible inconsistent sets of tags (i.e., having both a tag and its negation) from the set returned by *tags(n)*, we define the following set for each model ment: $context(n) = \{K | K \in tags(n) \wedge \neg inconsistent(K)\}$.

Inheritance of contextual tags. Contextual tags can *inherit* from other tags (no circular inheritance is allowed). This is to make specifying the effects of external factors on models easier. E.g., we have a tag *substantialOrder* applied to certain elements of a business process (BP) model. Now, we define a tag *largeOrder* inheriting from *substantialOrder*. Then, since *largeOrder is-a substantialOrder*, the derived tag can be substituted everywhere for the parent tag. Thus, the elements that are tagged with *substantialOrder* are effectively tagged with *largeOrder* as well. Of course, the converse is not true. Apart from being automatically applied to all the elements already tagged by *substantialOrder*, we can explicitly apply *largeOrder* to new nodes to specify, for example, that the goal Apply Discount requires large orders. The benefits of contextual tag inheritance include the ability to reuse already defined and applied tags and thus to develop context-dependent models incrementally. We state that one tag inherits from another by using the predicate *parent(parentTag,childTag)*. *Multiple inheritance* is allowed, so a tag can inherit from more than one parent tags. In this case, the derived tag can be used in place of all of its parent tags, thus inheriting the elements tagged by them. *parent* is extensionally defined based on the contextual tag inheritance hierarchy associated with the source model. *ancestor(anc,dec)* is defined trough *parent* to indicate that the tag *anc* is an ancestor of *dec*.

We also support a simple version of *non-monotonic inheritance* where certain elements tagged by an ancestor tag may not be inherited by the derived tag. Suppose the goal Apply Shipping Discount is tagged with *substantialOrder*, i.e., applies to substantial (large and medium) orders only. However, we might not want this goal to apply to large orders (as it would with regular inheritance) since we want them to ship for free. So, we declare this model element *abnormal* w.r.t. the inheritance of *largeOrder* from *substantialOrder* and that particular activity, which means that the *largeOrder* tag will not apply to it. We can do this by using the following: *ab(dec,anc,n)*, where *dec*, *anc* $\in \mathbf{C}$ and $n \in \mathbf{M}^{\mathbf{C}}$. This states that for the element *n* the descendent contextual tag (*dec*) cannot be substituted for the ancestor tag (*anc*). In fact, given tag combinations applied to *n*, we can determine if it is abnormal w.r.t. some inheritance hierarchy if there is a tag combination with an ancestor tag and a negation of a descendent tag:

$$ab(dec, anc, n) \stackrel{\text{def}}{\equiv} \left(\exists K \in context(n)\right) ancestor(anc, dec) \wedge anc \in K \wedge \neg dec \in K$$

Once a context *dec* is found to be abnormal w.r.t. one of its ancestors *anc* and a node *n*, all of *dec*'s descendents are automatically declared abnormal as well:

$$(\forall c, dec, anc \in \boldsymbol{C})(\forall n \in \boldsymbol{M^C}) ab(dec, anc, n) \wedge ancestor(dec, c) \rightarrow ab(c, anc, n)$$

Visibility of modeling elements. Given the sets of contextual tags applied to context-dependent model elements and the formulas defining when those tags are active, we can determine for each such element whether it is visible in the model. We define the following function:

$$visible: \boldsymbol{M^C} \rightarrow \{true, false\}$$
$$visible(n) \stackrel{\text{def}}{=} \big(\exists K \in context(n)\big)(\forall e_i \in K)$$
$$\bigwedge\nolimits_{1 \le i \le |K|} active(e_i) \vee (ancestor(e_i, d) \wedge \neg ab(d, e_i, n) \wedge active(d))$$

Thus, we define a context-dependent model element to be visible in a model if there exists a contextual tag assignment *K* for that element where each tag is either active itself or there exists its active non-abnormal descendent tag. Now we can produce the definition of the subset of visible context-dependent elements of a model: $\boldsymbol{V} \stackrel{\text{def}}{=} \{n | n \in \boldsymbol{M^C} \wedge visible(n)\}$. Note that for most modeling notations we also need other (e.g., structural) information in addition to the set **V** to produce a valid submodel corresponding to the current context. Since that information is notation-dependent, it is not part of our generic framework. Also note that since the definitions of contextual tags likely refer to real world phenomena, if the approach is used at runtime, the visibility of model elements can dynamically change from situation to situation.

4 Contextual Variability in Goal Modeling

In this section, we introduce our approach for modeling and analysing the effects of context on requirements goal models. We use the Distributor case study (see Fig. 1), which is a variation of the one presented in [12]. Due to space limitations, we are unable to present the complete goal model for the case study, although, we will be illustrating the approach with portions of it. The complete case study featured over 60 goals and six context refinement hierarchies.

Our method involves a number of activities. Some of these activities are discussed in the subsequent sections, while here we outline the approach:

1. Identify the main purpose of the system (its high-level goals) and the domain where the system is to operate.
2. *Iterative step*. Refine the goals into lower-level subgoals.
3. *Iterative step*. Identify the entities in the domain and their characteristics that can affect the newly identified goals. Capture those effects using *contextual tags*. Update the context model.
4. Generate the formal model for managing context-dependent variability.
5. Analyze context-enriched goal models
    a. Given currently active context(s), produce the corresponding goal model.
    b. Analyze whether top-level system goals can be attained given currently active context(s). The standard goal reasoning techniques can be applied since *the contextual variability has been removed*.

4.1 Context Identification and Modeling

Our goal in this approach is to systematically identify domain variability and its effect on stakeholder goals and goal refinements. Unlike intentional variability discussed in [14], domain variability is external to the requirements model, but influences intentional variability and thus requirements. We represent domain models in terms of *contexts* – properties or characteristics of the domain that have effect on requirements – and thus variability in the domain is reflected in the contextual variability.

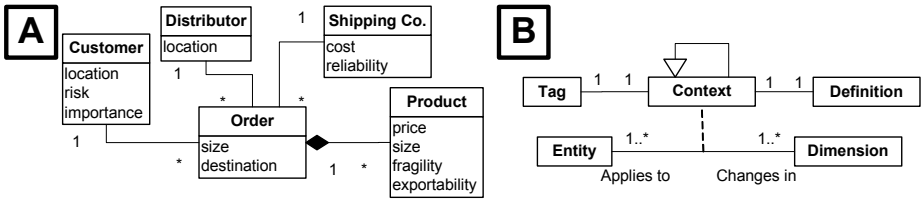

Fig. 2. UML context model for the case study (A) and our context metamodel (B)

Note that there may be certain aspects of the domain that do not affect requirements and these are not important to us. *Context entities*, such as actors, devices, resources, data items, etc., are things in the domain that influence the requirements (e.g., an *Order* is a context entity). They are the *sources of domain variability*. We define a context entity called *env* for specifying ambient properties of the environment. A *context variability dimension* is an aspect of a domain along which that domain changes. It may be related to one or more context entities (e.g., *size(Order)* and *relativeLocation(Warehouse,Customer)*). A dimension can be thought of as defining a range or a set of values. A *context* is a particular value for a dimension (e.g., *size(Order,$5000)*, *relativeLocation(Warehouse,Customer,local)*).

Fig. 2B shows the metamodel that we use for capturing the basic properties of domain variability (such as context entities and variability dimensions) in our approach. Additional models can also be useful. As mentioned in Section 2, there are a number of notations that can be employed for context modeling. Fig. 2A presents a UML class diagram variation showing the context entities in our case study (their corresponding context dimensions are modeled as attributes). In addition to UML or ER diagrams for context modeling, specialized notations like the CML are able to specify advanced properties of contexts (e.g., derived contexts).

Unlike the simpler notion of context in CML and in some other approaches, we are proposing the use of *context refinement hierarchies* for the appropriate context dimensions. Their purpose is twofold: first, they can be used to map the too-low-level contexts into higher-level ones that are more appropriate for some particular application (e.g., GPS coordinates can be mapped into cities and towns). This is commonly done in existing context-aware applications in the fields such as mobile and pervasive computing. Second, abstract context hierarchies may be useful in terms of splitting contexts into meaningful, appropriately named high-level ranges. For example, an order size (in terms of the dollar amount) is a number. So, one can specify the effects of orders of over $5,000 on the achievement of the subgoal Approve Order, then orders over $10,000, etc. However, very frequently, and *especially during requirements*

elicitation and analysis, it is more convenient to specify what effect certain *ranges* of context have on goal models. For example, instead of thinking in terms of the dollar amounts in the example above, it might be more convenient to reason in qualitative terms like *Large Order* or *Medium Order* (see Fig. 3A, where *Size* is the context dimension of the *Order* context entity, while the arrows represent IS-A relationships among contexts and the boxes capture the possible contexts in the hierarchy). The high-level contexts will need to be refined into lower-level ones and eventually defined using the actual order amounts. We call such defined contexts *base contexts* (note the "B" label on the leaf-level contexts in Fig. 3).

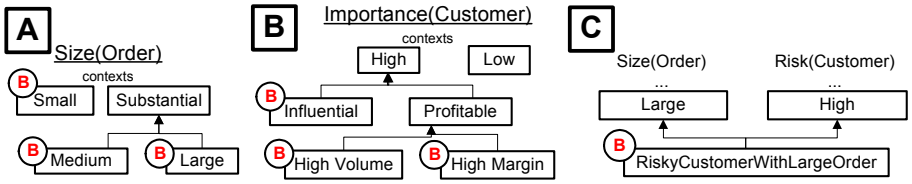

Fig. 3. Order size (A) and Customer importance (B) context hierarchies and multiple inheritance (C)

A context must be defined through a *definition*, a Boolean formula, which is specified using the expression of the type *Dimension(Entity(-ies),Context)* $\stackrel{\text{def}}{=}$ *definition*. If it holds (i.e., the domain is currently in the state defined by the context), we call that context *active*. For example, large orders may be defined as the ones over $1000. Thus, formally: $\forall$*Order size(Order, large)* $\stackrel{\text{def}}{=}$ $\exists n$ *size(Order, n)* $\wedge$ $n \geq$ *$1000*. As mentioned before, contexts may have concrete definitions or may be defined through their descendant contexts: $\forall$*Order size(Order, substantial)* $\stackrel{\text{def}}{=}$ *size(Order, large)* $\vee$ *size(Order, medium)*. There should be no cycles in context dependencies.

Contexts may be derived from several direct ancestors, thus inheriting their effects on the goal model. In Fig. 3C, we create a new context by deriving it from the contexts *size(Order,large)* and *risk(Customer,high)*. This produces a new context dimension with both context entities becoming its parameters. We also need to provide the definition for the new context, i.e., to specify when it is active: *sizeRisk(Customer, Order,riskyCustomerWithLargeOrder)* $\stackrel{\text{def}}{=}$ *size(Order,large)* $\wedge$ *risk(Customer, high)*. Thus, it is active precisely when the customer is risky and the order is large.

While context refinement hierarchies provide more flexibility for handling contexts, their design should not be arbitrary. When developing context hierarchies in our approach, care must be taken to ensure that they are not unnecessarily complicated, i.e., that the contexts are actually used in goal models.

4.2 Modeling the Effects of Contextual Variability on Goal Models

In Section 4.1, we discussed the modeling of domain characteristics using contexts. Here, we show how the effects of domain variability on requirements goal models can be captured. The idea is to be able to model the effects of all relevant contexts (i.e., the domain variability) conveniently in a single model instance and to selectively display the model corresponding to particular contexts. We use *contextual tags* (as in Section 3) attached to model elements to visually specify the effects of domain variability on goal models. While context definitions and inheritance hierarchies make up

the domain model, we need to specify how contexts affect the graphical models, i.e., which elements of the models are *visible* in which contexts.

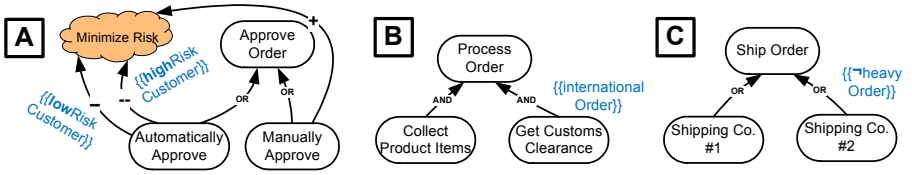

Fig. 4. Specifying effects of domain variability using contextual tags

Effects of contexts on goal models. Domain variability can influence a goal model in a number of ways. Note from the following that it can only affect (soft)goal nodes and contribution links. Domain variability affects:

- *The requirements themselves*. (Soft)goals may appear/disappear in the model depending on the context. For instance, if a customer is willing to share personal details/preferences with the seller, the vendor might acquire the goal Up-sell Customer to try and sell more relevant products to that customer.
- *The OR decomposition of goals*. New alternative(s) may be added and previously identified alternative(s) or may be removed in certain contexts. For example, there may be fewer options to ship heavy orders to customers (Fig. 4C).
- *Goal refinement*. For example, the goal of processing an international order is not attained unless the customs paperwork is completed (Fig. 4B). This, of course, does not apply to domestic orders.
- *The assessment of various choices in the goal model*. E.g., automatic approval of orders from low-risk customers may hurt ("–") the Minimize Risk softgoal, while doing the same for very risky ones will have a significantly worse ("--") effect on it (Fig. 4A).

Effects identification. The activities of developing contextual models and the identification of the effects of contexts on goal models need to proceed iteratively. While it is possible to attempt to identify all the relevant context entities and their dimensions upfront, it is very likely that certain important dimensions will be overlooked. For example, after the modeler refines the goal Package Order enough (see Fig. 1), he/she will elicit the goal Package Product. Only after analyzing which properties of a product can affect its packaging, will the modeler be able to identify the dimension *Fragility* as relevant for the context entity *Product*. Therefore, to gain the maximum benefit from the approach, the activities of context modeling need to be interleaved with the development of context-enriched goal models. Thus, the context model will be gradually expanded as the goal model is being created.

In our approach, when refining a goal, we need to identify the relevant context entities and their context dimensions that may influence the ways the goal is refined. There are a number of ways such relevant context entities can be identified. For example, in some versions of the goal modeling notation, goals have parameters (e.g., Process Order(Customer,Order), as in [12]), which are clearly context entities since their properties influence the way goals can be attained. Alternatively, a variability frame of a goal [14] can be a powerful tool for identifying relevant context entities and dimensions for a goal. We can use a table to document *potentially* relevant context entities (columns) and their dimensions (rows) for goals. While certain entities

and/or dimensions currently may have no effect on the refinement of the goal, it is still prudent to capture them for traceability and future maintenance. For instance, below is the table where we identified order size and destination as well as customer importance as dimensions affecting the goal Apply Discount.

Apply Discount	Entity: Order	Entity: Customer
Dimensions	Size, Destination	Importance

Specifying the effects of contexts on goal models. Tags are mnemonic names corresponding to contexts. For example, *largeOrder* may be the tag for the context *size(Order,large)*. Contextual tags are applied to model elements to specify the effects of domain variability on goal models – i.e., to indicate that certain contexts are required to be active for those elements to be visible in the model. As in Section 3, we have sets of alternative tag assignments and all the tags within any such assignment must be active for the model element to be visible. E.g., the set of tags *{{largeOrder},{importantCustomer,mediumOrder}}* attached to the goal Apply Discount indicates that ether the order has to be large or there must be an important customer with a medium-sized order to apply a discount. *Not* (¬) can be used with tags to indicate that the corresponding context cannot be active if the node is to be visible (see Fig. 4C).

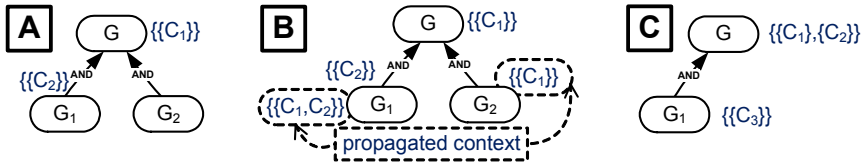

Fig. 5. Contextual tag assignment examples

By default, model elements are said to be contained in the *default context*, which is always active (*{{default}}*). To specify that a goal G must only be achieved when the context C_1 is active, we apply the tag $\{\{C_1\}\}$ to G (Fig. 5A). If we want a goal to be achieved when either of contexts is active, several sets of tag assignments must be used. E.g., the tag $\{\{C_1\},\{C_2\}\}$ applied to G (Fig. 5C) indicates that $C_1 \vee C_2$. When a set of tags is applied to a goal node G, it is also applied (implicitly propagated, see Fig. 5B) to the whole subtree rooted at that goal. The hierarchical nature of goal models allows us to drastically reduce the number of contextual tags used in the model. Tag sets are combined when used in the same goal model subtree. E.g., if a tag set $\{\{C_2\}\}$ is applied to the node G_1 in the subtree of G (Fig. 5B), then G_1 (and thus the subtree rooted at it) is to be attained only when *both* contexts corresponding to C_1 and C_2 are active, which is indicated by the tag $\{\{C_1,C_2\}\}$ (i.e., $C_1 \wedge C_2$). The tags applied to G and G_1 (Fig. 5C) when combined produce $\{\{C_1,C_3\},\{C_2,C_3\}\}$ since $(C_1 \vee C_2) \wedge C_3 = (C_1 \wedge C_3) \vee (C_2 \wedge C_3)$. The above also applies to softgoals.

4.3 Analyzing Context-Dependent Goal Models

In Section 3, we presented a generic formal framework for handling context-dependent models. It provides the basis for managing model variability due to external factors such as domain assumptions, etc. Here, we show how the formal framework can be used together with requirements goal models to analyze domain variabil-

ity in requirements engineering. In order to use the framework with goal models, we need a procedure that processes these models together with context inheritance hierarchies and generates the required sets and facts for the formal framework to operate on.

There are several steps in the process of generating the formal framework for goal models. First, we create the *parent* facts that model the tag inheritance hierarchy based on the context hierarchies described in Section 4.1. Similarly, definitions of the contexts will be assigned to the corresponding contextual tags and will be returned by the *active(context)* function for evaluation to determine if these tags are active.

We then state which elements of goal models we consider context-dependent. In general, the set $\mathbf{T}^{\mathbf{C}}$ = {G (goals), S (softgoals), R (contribution links)}. Below is the algorithm that completes the creation of the formal framework: it traverses the goal model and generates *taggedElement* instances corresponding to the context-dependent elements of the model along with the sets of tags assigned to these elements.

Algorithm 1: *Formal model generation*
Input: a set O of root (soft)goals of a goal model
Output: a formal model in the notation described in Section 3

```
1:  procedure generateFormalModel(O)
2:    for each e ∈ O do
3:      processNode(e, {{default}})
4:    endFor
5:  endProcedure
```

Algorithm 2: *Traverse goal model*
Input: element e and its parent context pC
Output: *taggedElement* entities in the formal model

```
01: procedure processNode(e, pC )
02:   newContext ← ∅
03:   if context annotation A exists for e then
04:     if pC = {{default}} then
05:       newContext ← A
06:     elseIf //parent context is not default
07:       for each K1 ∈ pC do
08:         for each K2 ∈ A do
09:           newContext ← newContext ∪
                                    {K1 ∪ K2}
10:         endFor
11:       endFor
12:     endIf //default context
13:   elseIf
14:     newContext ← pC
15:   endIf //annotation
16:   newTaggedElement(e, newContext)
17:   for each child contribution link l of e do
18:     processLink(l, newContext)
19:   endFor
20:   for each child (soft)goal node c of e do
21:     processNode(c, newContext)
22:   endFor
23: endProcedure
```

The procedure *generateContextModel* takes the set of root (soft)goals as the input and calls the procedure *processNode* on the (potentially) many (soft)goal trees that comprise the goal model. *processNode* has two parameters: the node e being processed and the set of tag assignments from the parent node, pC (parent context). Since we start from the root goals, initially pC has the value *{{default}}*. Within the *processNode* procedure we first check if the node e has a set of context tags A attached. If it does, it means that we must combine the parent context pC with A to produce the complete set of tags for e. If pC is the default context, it will simply be replaced by the tag set A. Otherwise, both pC and A are combined (as described in Section 4.2) to produce the new set of tags for e (see lines 7-11). We create the *taggedElement* unit for n with the newly produced context in line 16. The softgoal contribution links emanating from n are processed by the *processLink* function that computes the tag assignment for the link in the same way we have done it for n. Note that *newContext* is

provided to *processLink* as it becomes its parent context. Then we recursively process all the child nodes of *n* providing *newContext* as their parent context.

After *generateFormalModel* and other mapping procedures have been executed, we have a formal context framework that can be used to produce the set of elements visible in the model in the current context. Below we show the analysis that can be done on context-enriched goal models with the aid of our approach. Fig. 6A shows a fragment of the process Supply Customer for calculating shipping charges. Influential customers are not charged for shipping, so the context *{{¬F}}* (see the legend in Fig. 6 for abbreviations) is applied to it. We apply discounts only for important customers or for substantial orders, so Apply Discount is tagged with *{{I},{S}}*. [Provide] Large Discount is tagged with *{{I},{L}}*: it applies to large orders or to important customers. Finally, Medium Discount applies to international orders only. Fig. 6B shows a fragment of the formal model generated by the algorithm presented earlier (the inheritance hierarchies are based on those in Fig. 3). Note that the influential customer context tag (*F*) is found to be abnormal w.r.t. important customer (*I*) in the subtree Apply Discount. The sets of tags for each node are also calculated (Fig. 6B). By using context definitions (not shown), we can determine which contextual tags are active and thus affect the model. Suppose that we are in the context of a large international order (Fig. 6C). ¬*F* is active, so Charge for Shipping is visible. Apply Discount is too since a large order *is-a* substantial order and so both tags in *{¬F,S}* are active. Similar reasoning reveals that the remaining nodes are also visible. Note that we have *bound contextual variability* in the model by stating whether each context is active or not and by producing the corresponding version of the model. This process does not remove *non-contextual* variability from the model as shown in Fig. 6C where two choices for applying the shipping discount remain. The selection among them can be made using the conventional goal model analysis techniques (e.g., [18]).

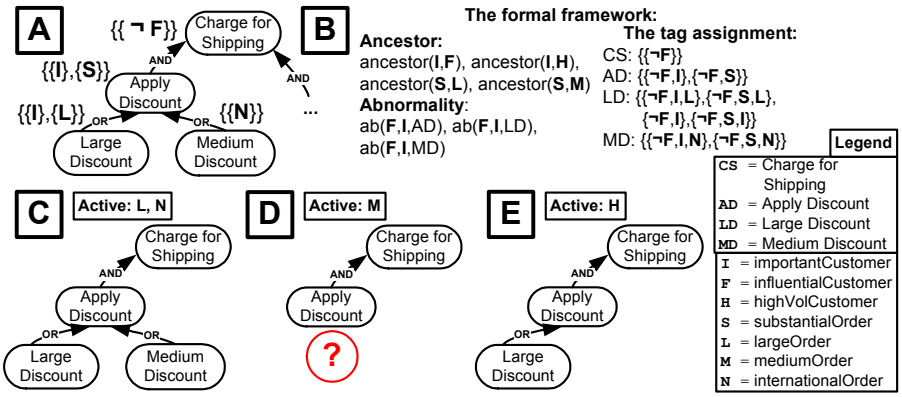

Fig. 6. Analyzing the effects of domain variability on goal models

Fig. 6D shows the model in the context of a medium order. Here, Charge for Shipping is visible again as is Apply Discount since medium orders are substantial orders. However, there are no combinations of active tags (see Fig. 6B) that make the other two goals visible. The analysis reveals a problem with the resulting model since no refinement of a *non-leaf* goal Apply Discount is available and thus any goal depending on it will not be achieved. One solution is to tag Medium Discount with *{{N},{M}}*

instead of *{{N}}*. Finally, Fig. 6E shows the model resulting from the context *high-VolumeCustomer* being active. Since these customers are important customers, they are given large discounts.

5 Discussion and Future Work

Our formal framework presented in Section 3 only deals with the visibility of context-dependent model elements. It does not guarantee that the resulting model is well-formed (e.g., as in Fig. 6D). So, we need additional formalization for each modeling notation to construct and verify model variants given the sets of elements visible in specific contexts. Thus, our framework represents the generic component for reasoning about contextual variability upon which complete solutions can be built. An example of such a solution is our approach to context-enriched goal models, where, unlike in most goal-based RE methods, we always do goal refinement *in context*.

The hierarchical nature of goal models helped us to reduce the number of tags and to simplify the creation of context-enriched models. Other modeling notations can also benefit from the same idea. We have dealt with limited non-monotonic inheritance and are also exploring ways of modeling richer notion of context inheritance.

We do not capture relationships among contexts other than inheritance. In future work, we would like to be able to recognize which contexts are compatible and which are in conflict, to handle different contexts with different priorities and in general to be able to choose whether and under what circumstances to recognize the effects of contexts on requirements. We are looking into developing or adopting richer context modeling notations to help in analyzing and documenting domain variability in RE.

Recently, context-based approaches for designing adaptive software have been growing in popularity (e.g., [17]). While high-variability goal models have been proposed as a vehicle for designing autonomic software [11], that approach did not consider the effects of domain variability on requirements and on the adaptive systems design. Thus, we are augmenting the approach of [11] with the context framework presented here to support both intentional and domain variability. We are exploring ideas like [17] for introducing context-based adaptation into the approach. Also, for adaptive systems design, we need to consider advanced context issues such as context volatility, scope, monitoring, etc., some of which were identified in [9]. We plan to further assess the approach using case studies in the area of BP modeling. While the complexity is the inherent property of many domains, the emphasis in the future work will be on improving the methodology to help reduce the complexity of context-enriched goal models by guiding the development of context hierarchies and by focusing only on relevant domain properties as well as on fully automating the generation of goal model variants for specific contexts. We are applying our framework to the problem of BP design and reconfiguration, further extending the method of [12].

6 Conclusion

We have shown a method for representing and reasoning about the effects of domain variability on requirements goal models as well as the underlying generic framework for reasoning about visibility of context-dependent model elements. We use a well-understood goal modeling notation enriched with contexts to capture and explore *all*

the effects of domain variability on requirements in a *single* model. Given a particular domain state, a goal model variation can be generated presenting the requirements for that particular domain variation. We propose the use of context refinement hierarchies, which help in structuring the domain, in decoupling context definitions from their effects, and in incremental development of context-enriched goal models.

Taking domain variability into consideration allows us, in conjunction with the approach of [14], to increase the precision and usefulness of requirements goal models, by explicitly capturing domain assumptions and their effects on software requirements.

References

1. P. Bouquet, C. Ghidini, F. Giunchiglia, E. Blanzieri. Theories and uses of context in knowledge representation and reasoning. *Journal of Pragmatics*, 35(3):455-484, 2003.
2. P. Brezillon. Context in Problem Solving: A Survey. *The Knowledge Engineering Review*, 14(1):1-34, 1999.
3. P. Brezillon, L. Pasquier, J-C. Pomerol. Reasoning with Contextual Graphs. *European Journal of Operational Research*, 136(2):290-298, 2002.
4. C. Cappiello, M. Comuzzi, E. Mussi, B. Pernici. Context Management for Adaptive Information Systems. *Electronic Notes in Theoretical Comp Sci*, 146(1):69-84, 2006.
5. J. Castro, M. Kolp, J. Mylopoulos. Towards Requirements-Driven Information Systems Engineering: The Tropos Project. *Information Systems*, 27(6):365-389, 2002.
6. S. Ceri, F. Daniel, F. Facca, M. Matera. Model-Driven Engineering of Active Context-awareness. *World Wide Web*, 10(4):387-413, 2007.
7. A. Dardenne, A. van Lamsweerde, S. Fickas. Goal-Directed Requirements Acquisition. *Science of Computer Programming*, 20(1-2):3-50, 1993.
8. P. Giorgini, J. Mylopoulos, E. Nicchiarelli, R. Sebastiani. Reasoning with Goal Models. Proc. *ER 2002*, Tampere, Finland, Oct 7-11, 2002.
9. K. Henricksen, J. Indulska. A Software Engineering Framework for Context-Aware Pervasive Computing. Proc. *PERCOM'04*, Orlando, FL, March 2004.
10. D. Hong, D. Chiu, V. Shen. Requirements Elicitation for the Design of Context-aware Applications in a Ubiquitous Environment. Proc. *ICEC 2005*, Xian, China, August 15-17, 2005.
11. A. Lapouchnian, Y. Yu, S. Liaskos, J. Mylopoulos. Requirements-Driven Design of Autonomic Application Software. Proc. *CASCON 2006*, Toronto, Canada, Oct 16-19, 2006.
12. A. Lapouchnian, Y. Yu, J. Mylopoulos. Requirements-Driven Design and Configuration Management of Business Processes. Proc. *BPM 2007*, Brisbane, Australia, Sep 24-28, 2007.
13. D. Lenat. The Dimensions of Context-Space. Technical Report, CYC Corp. Available at: www.cyc.com/doc/context-space.pdf
14. S. Liaskos, A. Lapouchnian, Y. Yu, E. Yu, J. Mylopoulos. On Goal-based Variability Acquisition and Analysis. Proc. *RE 2006*, Minneapolis, USA, Sep 11-15, 2006.
15. J. McCarthy and S. Buvac. Formalizing Context (Expanded Notes). *Computing Natural Language*. A. Aliseda et al, Eds. Stanford, CA, CSLI Publications**:** 13-50.
16. R. Prieto-Diaz. Domain Analysis: an Introduction. *SIGSOFT Software Engineering Notes*, 15(2):47-54, 1990.
17. M. Salifu, Y. Yu, B. Nuseibeh. Specifying Monitoring and Switching Problems in Context. Proc. *RE 2007*, New Delhi, India, Oct 15-19, 2007.
18. R. Sebastiani, P. Giorgini, J. Mylopoulos. Simple and Minimum-Cost Satisfiability for Goal Models. Proc. *CAiSE 2004*, LNCS 3084, pp. 20-35, Springer-Verlag, 2004.