Distributed Constraints Maintenance in Collaborative UML Modeling Environments

Haifeng Shen, Siyuan Liu, Steven Xia, and Chengzheng Sun
School of Computer Engineering
Nanyang Technological University, Singapore 639798
Email: {ashfshen, LIUS0012, stevenxia, czsun}@ntu.edu.sg

Abstract

Constraints maintenance plays an important role in keeping the integrity and validity of models in UML software modeling. Constraints maintenance capabilities are reasonably adequate in UML modeling applications, but few work has been done to address the distributed constraints maintenance issue in collaborative UML modeling environments. In this paper, we propose a novel solution to the issue, which can retain the effects of all concurrent modeling operations even though they may cause constraints violations. We further contribute a distributed constraints maintenance framework, in which the solution is encapsulated as a generic engine to be mounted in a variety of single-user UML modeling applications for supporting distributed collaborative UML modeling and distributed constraints maintenance.

Keywords: collaborative UML modeling, constraints maintenance, model consistency.

1. Introduction

Constraints are commonly used in interactive modeling applications to describe semantic restrictions imposed on models [1], [4]. Software modeling in UML (Unified Modeling Language) [5] is a typical example where constraints play a key role in ensuring the correct semantics of a model as well as complying with the rigid syntax enforced by the programming language to which the model is eventually transformed.

To keep the integrity and to verify the validity of a UML model by maintaining predefined constraints, constraints definition and validation capabilities have been integrated in most single-user UML modeling applications to warn users of potential constraints violations, or prevent users from performing operations that cause constraints violations. However, these capabilities do not address the distributed constraints maintenance issue in collaborative UML modeling environments, where multiple users take part in a common modeling task.

Consider an example shown in Figure 1, where two users collaboratively build an UML model with the Java multiple generalizations constraint (i.e., a class cannot be a subclass of multiple classes) enforced. To allow free and concurrent modeling operations from the two users, a replica of the model is distributed to each site. One only interacts with her/his own replica and operations made on one replica are propagated to the other site to keep these two replicas consistent.

![Figure 1. Distributed constraints maintenance issue](image)

Initially, the model consists of three classes part_time_student, student, and staff. The user at site 0 interacts with her/his replica at site 0, adds a class part_time_student to the model, and propagates the change to site 1. But the user at site 1 is not aware of this modification. She adds a class part_time_student to her site and propagates the change to site 0. Thus, the constraint violation is detected at site 0 and the user is informed of the violation. She then discards the new class part_time_student that is propagated to site 0 due to constraint violation.
work, referred to as SIGEMA (Seamlessly Integrating Generic Engine into Modeling Application), has the following characteristics. First, it leverages single-user UML modeling applications to support collaborative modeling and their constraints validation modules to support distributed constraints maintenance. Second, it offers a generic distributed constraints maintenance engine, which can be mounted in a variety of single-user UML modeling applications. Last, the engine lies on a cornerstone algorithm, which not only allows free and concurrent activities, but also maintains constraints consistently across all distributed sites even though they may cause constraints violations.

The SIGEMA framework consists of three layers, as shown in Figure 2. The top layer is SMA (Single-user Modeling Application), which provides conventional UML modeling capabilities. On the same layer, there is another component called CVM (Constraints Validation Module), which provides the constraints validation functionality. CVM can be an integral module of SMA or a separate software tool. For example, RSA has an integrated CVM, whereas Umbrello does not.

The bottom layer is the generic DCME (Distributed Constraints Maintenance Engine), which encapsulates a set of algorithms and data structures to solve the distributed constraints maintenance issue. The cornerstone of DCME is the COMIC (COncstraint Maintenance by PrioriTy Control) algorithm, which keeps the effects of operations that cause constraints violations in separate versions of a model. Multiple versions are maintained internally within operations involved in constraints violation and can be viewed one at a time. To achieve this, the engine provides two more components: PSAC

2. SIGEMA: A Distributed Constraints Maintenance Framework

We propose a distributed constraints maintenance framework to support collaborative UML modeling and distributed constraints maintenance. The framework, referred to as SIGEMA (Seamlessly Integrating Generic Engine into Modeling Application), has the following characteristics. First, it leverages single-user UML modeling applications to support collaborative modeling and their constraints validation modules to support distributed constraints maintenance. Second, it offers a generic distributed constraints maintenance engine, which can be mounted in a variety of single-user UML modeling applications. Last, the engine lies on a cornerstone algorithm, which not only allows free and concurrent activities, but also maintains constraints consistently across all distributed sites even though they may cause constraints violations.

The SIGEMA framework consists of three layers, as shown in Figure 2. The top layer is SMA (Single-user Modeling Application), which provides conventional UML modeling capabilities. On the same layer, there is another component called CVM (Constraints Validation Module), which provides the constraints validation functionality. CVM can be an integral module of SMA or a separate software tool. For example, RSA has an integrated CVM, whereas Umbrello does not.

The bottom layer is the generic DCME (Distributed Constraints Maintenance Engine), which encapsulates a set of algorithms and data structures to solve the distributed constraints maintenance issue. The cornerstone of DCME is the COMIC (COncstraint Maintenance by PriOrity Control) algorithm, which keeps the effects of operations that cause constraints violations in separate versions of a model. Multiple versions are maintained internally within operations involved in constraints violation and can be viewed one at a time. To achieve this, the engine provides two more components: PSAC

2. SIGEMA: A Distributed Constraints Maintenance Framework

We propose a distributed constraints maintenance framework to support collaborative UML modeling and distributed constraints maintenance. The framework, referred to as SIGEMA (Seamlessly Integrating Generic Engine into Modeling Application), has the following characteristics. First, it leverages single-user UML modeling applications to support collaborative modeling and their constraints validation modules to support distributed constraints maintenance. Second, it offers a generic distributed constraints maintenance engine, which can be mounted in a variety of single-user UML modeling applications. Last, the engine lies on a cornerstone algorithm, which not only allows free and concurrent activities, but also maintains constraints consistently across all distributed sites even though they may cause constraints violations.

The SIGEMA framework consists of three layers, as shown in Figure 2. The top layer is SMA (Single-user Modeling Application), which provides conventional UML modeling capabilities. On the same layer, there is another component called CVM (Constraints Validation Module), which provides the constraints validation functionality. CVM can be an integral module of SMA or a separate software tool. For example, RSA has an integrated CVM, whereas Umbrello does not.

The bottom layer is the generic DCME (Distributed Constraints Maintenance Engine), which encapsulates a set of algorithms and data structures to solve the distributed constraints maintenance issue. The cornerstone of DCME is the COMIC (COncstraint Maintenance by PriOrity Control) algorithm, which keeps the effects of operations that cause constraints violations in separate versions of a model. Multiple versions are maintained internally within operations involved in constraints violation and can be viewed one at a time. To achieve this, the engine provides two more components: PSAC

2. SIGEMA: A Distributed Constraints Maintenance Framework

We propose a distributed constraints maintenance framework to support collaborative UML modeling and distributed constraints maintenance. The framework, referred to as SIGEMA (Seamlessly Integrating Generic Engine into Modeling Application), has the following characteristics. First, it leverages single-user UML modeling applications to support collaborative modeling and their constraints validation modules to support distributed constraints maintenance. Second, it offers a generic distributed constraints maintenance engine, which can be mounted in a variety of single-user UML modeling applications. Last, the engine lies on a cornerstone algorithm, which not only allows free and concurrent activities, but also maintains constraints consistently across all distributed sites even though they may cause constraints violations.

The SIGEMA framework consists of three layers, as shown in Figure 2. The top layer is SMA (Single-user Modeling Application), which provides conventional UML modeling capabilities. On the same layer, there is another component called CVM (Constraints Validation Module), which provides the constraints validation functionality. CVM can be an integral module of SMA or a separate software tool. For example, RSA has an integrated CVM, whereas Umbrello does not.

The bottom layer is the generic DCME (Distributed Constraints Maintenance Engine), which encapsulates a set of algorithms and data structures to solve the distributed constraints maintenance issue. The cornerstone of DCME is the COMIC (COncstraint Maintenance by PriOrity Control) algorithm, which keeps the effects of operations that cause constraints violations in separate versions of a model. Multiple versions are maintained internally within operations involved in constraints violation and can be viewed one at a time. To achieve this, the engine provides two more components: PSAC
(Priority Specification, Assignment, and Control) and AVSD (Alternative Version Selection and Display). PSAC determines how priority is specified and assigned to operations, and how operations depend on each other. AVSD determines all available versions and the operation to be processed by COMIC for displaying/hiding a specific version.

The engine also maintains two essential data structures HB (History Buffer) and COOT (Constraint Object Operation Table). HB is an array list that stores a sequence of executed operations in the descending priority order. COOT is a table in which a column corresponds to a constraint, a row corresponds to an object, and a cell stores the operations that may cause the constraint violations involving the object.

The middle layer is AEA (Application-Engine Adaptor), which addresses the uniqueness of individual SMA, e.g., constraints, API (Application Programming Interface) to intercept events, execute operations, and validate constraints, in order to bridge the gap between SMA and DCME. A core component is the CORD (Constraint Operation-type Relationship Description) table in which a column corresponds to a constraint, a row corresponds to a type of operations, and a cell describes whether the type of operations may cause the constraint violations. CORD is catered for each individual CVM and is the “dictionary” based on which COOT is generated and filled.

AEA includes two concurrent threads: LOH (Local Operation Handler) and ROH (Remote Operation Handler). LOH intercepts SMA’s local events (using SMA’s API) to create UML modeling operations, and then validates whether these operations will cause constraint violations (using CVM’s API). Operations that do not cause constraints violations are processed by COMIC and propagated to other sites, ROH receives remote operations, which are processed by COMIC for distributed constraints maintenance.

3. Distributed Maintenance of Java Transformation Constraints with RSA

A UML modeling application such as RSA provides a set of modeling operations as create a package or a class, create a generalization relationship from a child class to a parent class, delete a package, class, or generalization, and update a key of a package, class, or generalization from the old value to the new value. The definition of these operations can be found in reference [3].

Each operation is timestamped with an SV (State Vector) [2]. Operations priorities can be defined according to their total ordering relations defined as follows.

**Definition 1: Totally Preceding Relation**

Given two operations \( O_a \) and \( O_b \) generated at site \( i \) and \( j \) and timestamped by \( O_a.SV \) and \( O_b.SV \) respectively, \( O_a \) is totally preceding \( O_b \), denoted by \( O_a \Rightarrow O_b \), if:

1) \[ \sum_{k=0}^{|SV|-1} O_a.SV[k] < \sum_{k=0}^{|SV|-1} O_b.SV[k], \] or
2) \[ \sum_{k=0}^{|SV|-1} O_a.SV[k] = \sum_{k=0}^{|SV|-1} O_b.SV[k] \text{ and } i < j. \]

So, if \( O_a \Rightarrow O_b \), then \( O_a \) has a higher priority than \( O_b \) does, denoted by \( O_a \gg O_b \). Moreover, RSA has the following predefined Java transformation constraints for UML modeling:

1) **Circular generalization**: an inheritance cycle is not allowed among classes.
2) **Default package**: a class cannot be added directly into a model; it must be added into a package.
3) **Multiple generalizations**: a class cannot be a subclass of multiple classes.
4) **Top Classifier private**: a class cannot be set private if it is not a subclass of any other classes.
5) **Abstract method**: an abstract method must be defined in an abstract class.
6) **Duplicate name**: different objects in the same namespace must have identical names.
7) **Abstract and leaf**: a class cannot be set both abstract and private/leaf.
8) **Name correct**: an objects name must conform to the Java naming rule.

Consider a collaborative UML modeling session, where the shared model (initially consisting of three Java classes student, staff, and part_time_student) is distributed across four sites.

![Figure 3. Operation arrival order at each site](image)

The user at site 0 issued operation \( O_1 \) to create a generalization relationship from part_time_student to student. Concurrently, the user at site part_time_student issued operation \( O_2 \) to create a generalization relationship from part_time_student to staff and the user at site...
3 issued two operations $O_3$ and $O_4$ in sequence. $O_3$ was to create a generalization relationship from staff to part_time_student and $O_4$ was to delete this generalization relationship. At site 2, after $O_3$ propagated from Site 3 had been executed, the user issued operation $O_5$ to update staff to be private. The arrival order of the five operations at each site is shown in Figure 3. According to the above priority specification, $O_1 \gg O_2 \gg O_3 \gg O_5 \gg O_4$.

![Image](https://via.placeholder.com/150)

Figure 4. Three versions of the model

As shown in Figure 4, after all operations have been executed by the COMIC algorithm for distributed maintenance of the above Java transformation constraints at the four sites, PSAC can derive three versions of the model: $V_1 = [O_1, O_3, O_5]$, $V_2 = [O_1, O_3, O_4]$, and $V_3 = [O_2]$. Assuming $V_1$ is the displayed version, to display $V_2$, AVSD will determine that $O_5$ needs to be undone by COMIC; to display $V_3$, AVSD will determine that $O_1$ needs to be undone by COMIC.

4. Conclusions and Future Work

As collaborative UML modeling is increasingly desirable, distributed constraints maintenance becomes a challenging technical issue that hinders the usability of collaborative UML modeling environments. While existing environments either leave this issue to users or avoid this issue by preventing concurrency, we propose a solution that not only allows free and concurrent activities from distributed users, but also maintains constraints consistently across all distributed sites by retaining the effects of all concurrent operations even though they may cause constraints violations.

Building upon the cornerstone algorithm COMIC, we devised a distributed constraints maintenance framework SIGEMA, in which a generic engine DCME can be mounted in a variety of single-user UML modeling applications to support collaborative modeling and distributed constraints maintenance via application-tailored adaptors AEs. We are working towards allowing automatic maintenance of more application-built-in or user-defined constraints. Displaying alternative versions of a model is currently supported by undoing operations in chronological order and re-displaying a version is not supported. We are designing a user interface that can aggregate all versions of a model, which can be chosen for display at any time.

Acknowledgment

The work is partially supported by a Singapore ASTAR PSF grant (062-101-0034).

References


