Semantics-based Reconciliation of Divergent Replicas in Advanced Concurrent Engineering Environments

Vitaly Semenov[1]

Institute for System Programming RAS, Moscow, Russia.

Abstract. A novel method for semantics-based reconciliation of long-lived transactions in concurrent engineering environments is described. The reconciliation is a key element of recent optimistic replication technologies facing the challenges of diverging replicas, conflicts between concurrent operations and disturbing consistency. The research presented addresses the general problem of semantically consistent and functionally meaningful reconciliation and enables significant simplification and formalization of its solution. The advantages of the method are strong guarantees of semantic consistency of the convergent representation, capabilities to use it in autonomous and user interactive modes as well as avoidance of combinatorial explosion peculiar to many other methods. The particular application is presented to illustrate how the method can be effectively applied for collaborative software engineering in the scope of the emerging UML-driven methodology.

Keywords. Concurrent engineering environments, optimistic replication, reconciliation, EXPRESS, UML/OCL

1 Optimistic replication

Optimistic replication is the key technology in advanced distributed systems intended for effective collaboration of various stakeholders involved in joint engineering activities to be conducted concurrently [1].

Sacrificing underlying ACID (atomicity, consistency, isolation, and durability) principles the traditional database management systems rely on, optimistic replication enables higher availability and performance [2]. The advantages are achieved through reduced latency by letting users and applications share data simultaneously as well as through increased throughput by avoiding centralized query processing and remote network access. Instead of synchronous replica coordination deployed in pessimistic transaction models, optimistic methods let data be accessed without a priori blocking based on the assumption that occasional conflicts can be always fixed after they happen.

[1] Professor, Doctor's Degree in Computer Sciences, Institute for System Programming of the Russian Academy of Sciences; 25, Bolshaya Kommunisticheskaya str., Moscow, 109004, Russia; Tel: +7 (495) 9125317; Fax: +7 (495) 9121524; Email: sem@ispras.ru; http://www.ispras.ru

Often, human activities like cooperative engineering or software development require people to work asynchronously in relative isolation. It is better to allow them to update data independently and repair occasional conflicts after they happen than to lock the data out while someone is editing it [3]. CVS, Lotus Notes, Synergy, subversion [4] are some applications that exploit optimistic replication for collaboration purposes.

These benefits, however, come at a cost, since there is a principal trade-off between data availability and consistency common for all distributed systems. Optimistic replication faces the challenges of diverging replicas, conflicts between concurrent operations and disturbing consistency. These issues are especially critical for advanced multi-modal collaborative environments enabling long-lived transactions, supporting semantically rich operations and managing complex scientific and engineering data [5] like those defined by ISO 10303 — Standard for Product Data Representation and Exchange (STEP) [9] and by Model Driven Architecture initiative by Object Management Group (OMG MDA) [10].

Such data are usually multidisciplinary, highly scaled, and too complicated the diverged replicas to be managed and agreed manually. Being specified in the EXPRESS [11], the Unified Modeling Language and Object Constraint Language (UML/OCL) [12], the data models may contain definitions of hundreds and thousands of data types and constraint sorts. This makes realization of the reconciliation and replication approach in whole a nontrivial task. Creation of application-specific solutions is not satisfactory because of complexity and variety of the actual product models.

The circumstances above inspired us to develop a systematic semantics-based reconciliation method that would incorporate three basic principles making it quite universal and constructive for a lot of problems and applications:

— *support of emerging STEP and OMG MDA standards* to use general and popular object-oriented modeling languages like EXPRESS, UML/OCL as well as to employ standard product models developed by wide academy and industry communities for key engineering domains;

— *formal semantic analysis of concurrent transactions* covering both underlying model data structures and algebraic constraints imposed upon them;

— *logic (poly-syllogistic) deduction* targeted on construction of most promising reconciliation schedules that would incorporate as many as possible operations from concurrent transactions and would maintain consistency, correctness, and meaningfulness of the resulting convergent data representation.

This paper focuses on basic principles of the reconciliation method. Section 2 presents a general organization of a reconciliation process and shortly concerns a classification of data and constraints occurred in practice. In section 3 explanations how formal analysis of data models can facilitate establishing relations among operations in concurrent transactions and how logic deduction can be applied for their consistent and meaningful reconciliation are provided. Section 4 presents an application developed for the reconciliation of UML diagrams in the scope of the emerging model-driven software engineering methodology. In conclusions we mention some adjacent problems and outline promising directions for future investigations.

2 General reconciliation process

According to the optimistic replication, applications running within distributed systems are either in isolated execution phase or in the reconciliation phase. At isolated phase applications proceed on local replicas of the shared data bringing them to some tentative states. All the actions undertaken are sequentially recorded in transaction logs. Actions are assumed to be deterministic and reversible. Replaying a log against the initial state results in the same final state and, vice versa, being reversed the log reproduces the initial state from the final one. We suggest that transaction logs are correct in that sense they can be correctly replayed and correspond to consistent and meaningful tentative states. It is a rather reasonable assumption motivated by correctness of the applications running on local machines and proceeding on local data replicas.

A typical repertoire of the actions for the object-oriented data involves operations with extended read/write semantics like creation of new object, changing an object class, modification of object attributes, reassignment of object associations and aggregations, and deletion of an object. Object-oriented modeling languages like EXPRESS, UML/OCL provide a wide range of declarative and imperative constructs to define both data structures, derived properties and algebraic constraints. In particular, strings, enumerations, collections, object types, and classes can be defined within the specified data model. The constraints can restrict acceptable widths of binaries and strings; cardinalities of collections; multiplicities of associations and compositions; non-mandatory attributes; uniqueness of attributes and their aggregate groups; value domains of separate attributes, objects and whole object populations.

We suggest that general reconciliation process incorporates following seven stages presented by the figure 1.

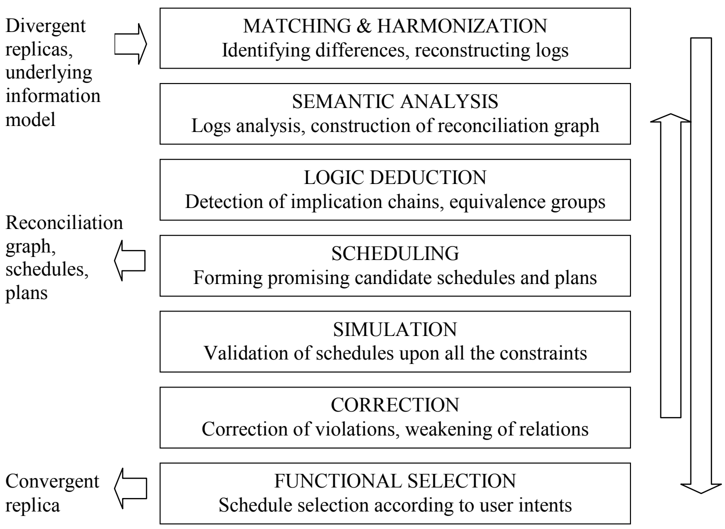

Figure 1. Basic phases of the general reconciliation process

At the *matching and harmonization* stage the divergent data replicas are compared with each other to identify and to harmonize differences and to correctly reconstruct logs. The actions contained in transaction logs are joined at the *semantic analysis* stage and analyzed against all the preconditions and constraints defined by the data model. Dependence and precedence relations are established among the actions. *Logic deduction* is applied to determine transitive closures of the relations and to form implication chains and equivalence groups for the actions. At the s*cheduling* stage alternative solutions are explored to satisfy the representativeness requirement for the consolidated schedule and the consistency requirement for the convergent data representation. The *simulation* stage is necessary to verify whether the built schedules really satisfy all the conditions. If no acceptable schedules have been found, *correction* may be constructive to weaken the established relations and to bring the resulting data representation into consistent state. Correction can be executed iteratively in feedback cycle as shown in the figure 1. Finally, at the *functional selection* stage the schedules being successive simulation outcomes are chosen in accordance with functional requirements and user intents.

3 Semantic analysis of concurrent transactions

Reconciliation should combine the initial logs in some way to produce a new log, which can be replayed to bring the replicated data from its last common consistent state to a new common consistent state. Ideally, the reconciled schedule would contain all the actions and satisfy all the constraints and user intents. Nevertheless, careful analysis of semantic relations among actions of concurrent transactions has to be conducted to guarantee that in general case.

The tentative log $T = T^{'} \cup T^{''}$ is a set of actions formed by union of the reconciled transactions $T^{'}$, $T^{''}$. In other words, if x is an ancestor state of the data and $x^{'}$, $x^{''}$ are the divergent replicas of the data, then $x^{'} = T^{'}(x)$ and $x^{''} = T^{''}(x)$.

The dependence relations D are defined using boolean operations as follows. For actions $t_1, t_2 \in T$, if $t_1 \rightarrow t_2$ then the schedule must contain t_2 on condition that it contains t_1. If $\neg t_1 \rightarrow \neg t_2$ then the schedule must exclude t_2 on condition that it doesn't contain t_1. The relations are non-symmetric, reflexive and transitive. We find also reasonable to utilize the symmetric implication relations $t_1 \rightarrow \neg t_2$, $\neg t_1 \rightarrow t_2$ as well as the relations induced by equivalence operation $t_1 \sim t_2 \equiv t_1 \rightarrow t_2 \wedge t_2 \rightarrow t_1$ and by exclusive disjunction operation $t_1 \oplus t_2 \equiv t_1 \rightarrow \neg t_2 \wedge \neg t_1 \rightarrow t_2$.

In some cases multiple dependency relations should be employed for the comprehensive semantic analysis. They can be represented in a general form by characteristic boolean functions $D(t_1, t_2, t_3, \ldots)$ and corresponding truth tables [13].

As an example, the relation $D^{(n:m)}(t_1^{+},\ldots, t^{+}_{I+}, t_1^{-},\ldots, t^{-}_{I-})$ should be taken into account if some cardinality constraint is imposed upon a collection and transaction actions are capable to insert new elements in the collection and remove the elements from it. The relation takes place if only $n \leq card(\ T^{+}\ ) - card(\ T^{-}\ ) \leq m$, where $T^{+} = \{\ t_i^{+}, i \in (1, I^{+})\ |\ t_i^{+} = \ true\ \}$, $T^{-} = \{\ t_i^{-}, i \in (1, I^{-})\ |\ t_i^{-} = \ true\ \}$, and the function *card*() returns the cardinal number of action subsets T^{+}, T^{-}.

The characteristic functions of the cardinality relations $D^{(0:0)}(t_1^+, t_2^+, t_1^-, t_2^-)$, $D^{(1:1)}(t_1^+, t_2^+, t_1^-, t_2^-)$ and $D^{(2:2)}(t_1^+, t_2^+, t_1^-, t_2^-)$ are given by trivial truth tables [8]. The functions of derived cardinality relations can be expressed using the following recursive identities:

$D^{(n:m)}(t_1^+,\ldots, t_1^-,\ldots) \equiv D^{(-m:-n)}(t_1^-,\ldots, t_1^+,\ldots)$,

$D^{(n:m)}(t_1^+,\ldots, t_1^-,\ldots) \equiv D^{(n:n)}(t_1^+,\ldots, t_1^-,\ldots) \vee \ldots \vee D^{(m:m)}(t_1^+,\ldots, t_1^-, \ldots)$

Often, if some constraint $c(x_1, x_2,\ldots)$ is given by a predicate function of several variables and it is required to satisfy it, the algebraic dependence relation $D^c(t_1', t_2', \ldots, t_1'', t_2'',\ldots)$ can be set among the transaction actions, which modify the variables of the given constraint. The relation takes place if the resulting schedule contains all the modification actions belonging either to one or another transaction $D^c(t_1', t_2',\ldots,t_1'', t_2'',\ldots) \equiv t_1' \oplus t_1'' \wedge t_1' \sim t_2' \wedge t_2' \sim t_3' \wedge\ldots\wedge\ t_1'' \sim t_2'' \wedge t_2'' \sim t_3'' \wedge\ldots$ Being established the relation guarantees the algebraic constraint will be satisfied.

The precedence relation P is defined as follows. For actions $t_1, t_2 \in T$, if $t_1 \angle t_2$ then action t_1 must appear before (not necessarily immediately before) action t_2 in any schedule that contains both t_1 and t_2. The precedence relation is non-symmetric, non-reflexive, but transitive.

Once the relations are established among actions, the logic deduction can be applied to form a resulting schedule that would satisfy all the action preconditions and the semantic constraints. Early we presented a method exploiting a graph representation for the introduced semantic relations and a poly-syllogistic deduction on the graph to form consistent schedules. The method enables to determine implication chains, equivalence groups, and conflicts by transforming of the semantic relations into logic ones and by computing their transitive closures. An important feature is that the method tends to consolidate within the final schedule as many actions as possible (see the published works [7, 8]).

4 Reconciliation of UML models

For illustration let discuss the developed reconciliation method in conformity to collaborative software engineering in the scope of the emerging UML-driven methodology [10]. The UML has been widely accepted throughout the software industry and is successfully applied to diverse domains. The UML provides a variety of diagram types for describing dynamic behavior of software being developed, its static structures, functional requirements, and implementation features. The OCL is its textual extension intended for specifying constraints.

Using visual and textual notations provided by the UML/OCL languages, software engineers (e.g. analytics, architects, requirement engineers, designers, developers, testers, etc.) can effectively collaborate by sharing and exchanging the models. But possible conflicts in the model replicas caused by asynchronous work on them must be identified and resolved consistently. Because of high complexity of the underlying UML/OCL metamodel, it cannot be done manually without a risk to disturb the consistency. Nevertheless, the presented semantics-based reconciliation method can be employed for such purposes.

The class diagram in the figure 2 presents a partial, simplified fragment of the metamodel responsible for visual representation of the UML state chart diagrams.

The class diagram illustrates a simple visualization scenario describing the generalizations, compositions and associations among Diagram, DiagramElement, Symbol and Line classes. The scenario indicates that Diagram instances consist of elements of Symbol or Line types. The concrete classes StartSymbol, TaskSymbol, DecisionSymbol, CommentSymbol and some other allied classes are specializations of the abstract Symbol class. The classes FlowLine and CommentLine are specializations of the Line class. The associations From and To established at abstract level are inherited by the concrete classes in which the referenced association types and multiplicities are redefined.

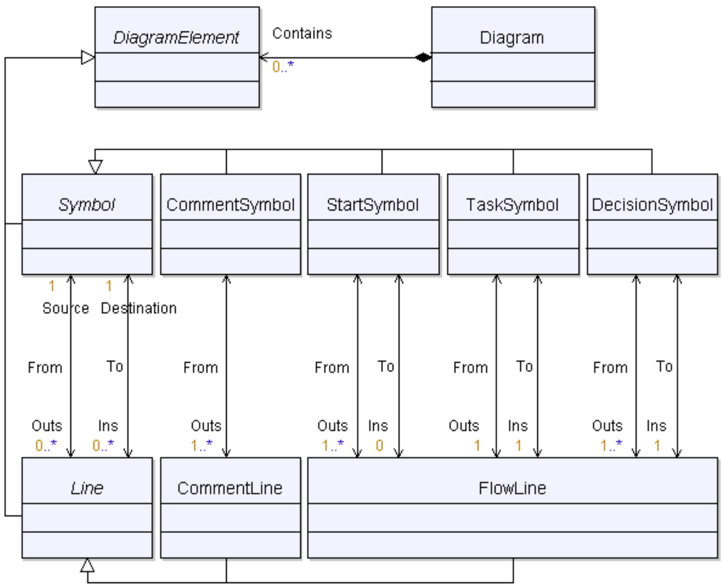

Figure 2. The metamodel representation of the UML statechart diagrams

The figure 3 presents replicas of the concurrently developed state chart diagram — common ancestor version (a), versions obtained in the first and the second transactions (b), (c), and a resulting version (d). In the version (b) new transition chain has been added consisting of Signal3 ∈InputSignalSymbol, Task1(i = 20) ∈ TaskSymbol, State2 ∈StateSymbol and connecting them Line1(Idle -> Signal3), Line2(Signal3 -> Task), Line3(Task -> State2) ∈FlowLine. The previous State1 ∈ StateSymbol has been deleted, and output of the Task (i = 10) ∈TaskSymbol via Line4 ∈FlowLine has been reconnected.

The first transaction log can be represented as follows: $T^{'} = \{t_1^{'} =$ *new* (Signal3), $t_2^{'} =$ *new* (Task1), $t_3^{'} =$ *new* (State2), $t_4^{'} =$ *new* (Line1), $t_5^{'} =$ *new* (Line2), $t_6^{'} =$ *new* (Line3), $t_7^{'} =$ *wr* (Line4.Destination, State2), $t_8^{'} =$ *del* (State1)$\}$, where symbols *new*, *wr*, *del* denote creation, modification and deletion actions. Here action $t_4^{'} =$ *new* (Line1) implicitly assumes the setting of mandatory Source and Destination associations: *wr* (Line1.Source, Idle), *wr* (Line1.Destination, Signal3).

Then the following relations can be established among actions. Implications $t_1' \rightarrow t_4'$, $t_1' \rightarrow t_5'$ take place as the signal3 must have input and output. In fact, more strong equivalence relation takes place because of Source and Destination associations for newly created flow lines must be set. Thus, to guarantee the semantic correctness $t_1' \sim t_2' \sim t_4' \sim t_5' \sim t_6'$ must be satisfied. In order to be able to reply the log, the partial ordering $t_1' \angle t_4'$, $t_1' \angle t_5'$, $t_2' \angle t_5'$, $t_2' \angle t_6'$ must be kept.

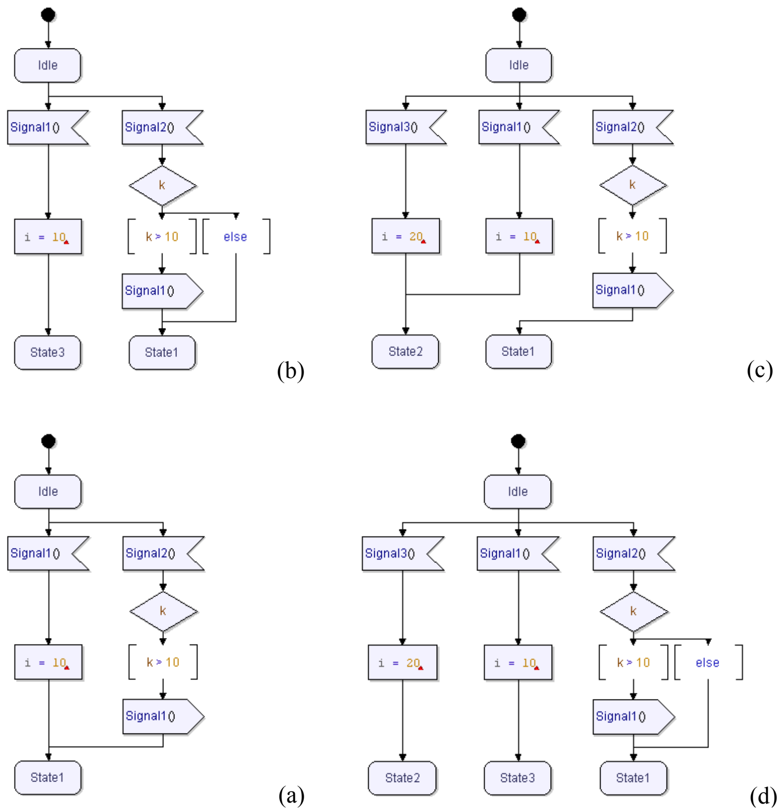

Figure 3. The original, divergent and resulting replicas of the statechart diagram

Similar semantic analysis can be performed for the second transaction and across concurrent transactions to detect possible conflicts. So, in the example considered there is a conflict between the actions *wr* (Line4.Destination, state2), *wr* (Line4.Destination, state3) in the first and the second transactions respectively. If the conflict is resolved by taking changes made by the second transaction, the result looks as shown at the figure 3. It is important that the semantically consistent and functionally meaningful result has been derived in a completely formal way using underlying data model and applying logic, poly-syllogistic deduction. Certainly, the example is not exhaustive. Nevertheless, it outlines the essential advantages of the developed method.

5. Conclusions

Thus, the method for semantics-based reconciliation of divergent replicas in advanced concurrent engineering environments has been presented. Its main advantages are a significant formalization making possible to employ it for sophisticated data models and applications, mathematically strong guarantees of the result correctness and meaningfulness, capabilities to use it in autonomous and user interactive modes as well as avoidance of combinatorial explosion peculiar to many other methods. In the future, some adjacent problems will be investigated. These are semantics-based matching of divergent replicas, building well-balanced reconciliation plans, adaptive correction of transactions as well as application-specific reconciliation solutions.

6. References

[1] Anderson T, Breitbart Y, Korth HA. Wool, replication, consistency, and practicality: are these mutually exclusive? In: Proceedings ACM SIGMOD, Seattle, WA, USA, 1998; 484-495.
[2] Cederqvist P, Pesch R, *et al.* Version management with CVS, 2001. Available at: http://www.cvshome.org/docs/manual. Accessed on: Feb. 26th 2007.
[3] ISO 10303. Industrial automation systems and integration — Product data representation and exchange, 1994.
[4] ISO 10303-11: 1994, Industrial automation systems and integration — Product data representation and exchange — Part 11: Description methods: The EXPRESS language reference manual.
[5] Model driven architecture: how systems will be built, 2006. Available at: http://www.omg.org/mda. Accessed on: Feb. 26th 2007.
[6] Ramamritham K, Chrysanthis P. Executive briefing: advances in concurrency control and transaction processing. IEEE Computer Society Press, 1997.
[7] Saito Y, Shapiro M. Optimistic Replication. ACM Computing Surveys 2005; 37(1): 42–81.
[8] Semenov V, Bazhan A, *et al.* Distributed STEP-compliant platform for multi-modal collaboration in architecture, engineering and construction. In: Proceedings of X International Conference on computing in civil and building engineering, Weimar, 2004; 318-319.
[9] Semenov V, Bazhan A, *et al.* Efficient verification of product model data: an approach and an analysis. In: Proceedings of 22nd Conference on information technology in construction, Dresden, 2005; 261-268.
[10] Semenov V, Karaulov A. Semantic-based decomposition of long-lived optimistic transactions in advanced collaborative environments. In: Proceedings of 6 European Conference on product and process modeling, Valencia, 2006; 223–232.
[11] Semenov V, Eroshkin S, *et al.* Semantic reconciliation of product data using model specifications. In: Proceedings of Institute for System Programming, Moscow, ISP RAS, 2007; 21-42.
[12] Unified Modeling Language (UML), Version 2.0. Available at: http://www.uml.org/#UML2.0. Accessed on: Feb. 26th 2007.
[13] Zakrevskij A. Recognition logic. Editorial Press. Moscow. 2003.