
A&QT-R 2002 (THETA 13)
2002 IEEE-TTTC International Conference on Automation, Quality and Testing, Robotics
May 23-25, 2002, Cluj-Napoca, Romania
- -

A Transaction Model for Long-Lived and Complex Computations

Dollinger Robert

Technical University of Cluj, Department of Computer Science,
Email: robert.dollinger@cs.utcluj.ro

Abstract - The classical transaction concept needs to be reconsidered because of its limitations in dealing with long-lived and complex computations. Managing such tasks requires more systematic approaches, based on a broader perspective of the entire programming model, rather than simple extensions of the classical transaction model. This paper presents the ComET (Compensation based Extended Transaction) model that aims to provide a higher extent control mechanism based on a few basic principles: (1) large computations are divided into several smaller transactional processing units (called steps) that are executed in a controlled and coordinated way, (2) the effect of any step may be compensated at any moment before the termination of the computation by executing a compensating step (transaction), (3) coordination and execution control of steps is done at the semantic level by using invariants, (4) the higher level logic of the application is captured into a simple and effective script language code.
Keywords: extended transaction models, complex computations, compensating transactions, invariants.

1. INTRODUCTION

Transactions have been proven useful in the field of database systems and are a common part of many important applications. However, there are many fields where classical transactions are not suitable to model the actions of the 'real world', and need to be reconsidered, as it is in the case of long-lived and complex computations. There have been several attempts to modify and extend the transaction concept [2][3]. One of the directions was to extend in different ways the transaction concept, thus producing models that where referenced in early literature as extended transaction models and, later, advanced transaction models. Initially, these models where developed on an ad-hoc basis as a response to the demands of some specific applications. Among the efforts to provide a unifying view of extended transaction models, the ACTA formalism proposed by Chrysantis and Ramamritham [1], can be recorded as one of the most remarkable. ACTA is a theoretical framework for describing extended transaction models and a valuable tool for reasoning about their possibilities and limitations. Another direction emerged from the observation that simply extending the classical transaction model does not solve the problem because long-lived and complex computations cannot match the requirements of ACID transactions, not even partially. Managing such tasks requires more systematic approaches, based on a broader perspective of the entire programming model. One of the most representative

approaches that goes beyond the simple extension of ACID transactions, is the ConTract model [4][5][6][7][8]. ConTracts cover most of the requirements of workflow management, thus providing a run-time system that can be the basis for the construction of general purpose workflow management systems. The approach we present in this paper is closely related to the ConTract model from which it is inspired. However, our model, which we called ComET (Compensation based Extended Transaction model), differs in several respects from the original model both in conception and implementation solutions, as it will be pointed out.

2. BASIC PRINCIPLES OF THE ComET MODEL

Transactions have been adopted as a means for translating the structural constraints of a database into dynamic, behavioral constraints. Transactions impose constraints on the grouping and order of data modifications in order to preserve database consistency, which at low level is much more cost effective than the brute verification of structural constraints satisfaction. ComET adopts a two-level solution for maintaining database consistency. It maintains ACID transactions at low level, while defining a control mechanism above ACID transactions at the higher level of the whole (complex) application. This latest mechanism is based on the enforcement of structural constraints. Complex computations are divided into several units of execution, called steps that are basically implemented as simple transactions at the database level. The higher-level mechanism ensures the controlled and coordinated execution of the steps in order to provide ACID like properties to the entire application according to the flow of control expressed in a simple script language. Thus a ComET consists of a set of transactions expressing the data manipulation operations at the database level and a script expressing the higher-level application logic. The script language defines the usual control flow instructions like: sequence, branching, loop and other. However, note that the execution of the script is constrained by the ComET high-level control mechanism in order to ensure database consistency. This mechanism is built on several concepts:

- forward recovery – a ComET is not rolled back after a failure during its execution. As much as possible of the work already done, is preserved by resuming execution from a preceding point that is closest to the point of failure. This could be the last committed transaction in the case of a system failure or a point determined by backtracking if the execution of a step failed;

- compensating transactions – the effects of a step execution are not necessarily permanent once the corresponding transaction is committed. Each transaction within a ComET must have a compensating transaction. The effects of a transaction can be cancelled any time before ComET termination by executing its compensating transaction. Once requested, a compensating transaction is guaranteed to commit;

- invariants – are predicates that express structural constraints at the level of data shared among several concurrent ComETs. Each step verifies a set of **entry invariants** before executed and generates a set of **exit invariants** when terminating;

- individual log – each ComET carries with him an individual log where its history of execution and current status is recorded. Using the data recorded in the log, the system can take appropriate decisions when recovering after a failure.

3. THE ComET MODEL AND THE ACID PROPERTIES

Each of the above mentioned features contribute in its own way to the achievement of ACID like behavior of ComETs. In this respect one can make a parallel between ACID transactions and ComETs:

Atomicity - A transaction is executed as an atomic unit of work according to the "all or nothing" rule. If successful, all its effects become permanent else all the effects are rolled back. The fundamental deviation from classical transactions is that ComET gives up atomicity at the script level because these properties are incompatible with the needs of long duration activities. The ComET model replaces the classical atomicity property with a more relaxed concept: a ComET can be interrupted explicitly and continued by the user after an arbitrary delay. And more important, a crash along the way does not initiate rollback. Rather the system initiates roll forward recovery, maybe along a different path than the one taken before. Thus, any ComET execution ends by a consistent database state.

Consistency - A transaction is a correct transformation of a system state. In other words, a transaction is a correct program that translates the database from a consistent state to another consistent state. ComET maintains system integrity not only at the level of single transactions, but extends it on the much larger scale of complex computations. Within a ComET execution any step that produces an inconsistent database state will be automatically cancelled by the execution of its compensating transaction.

Isolation - Concurrent transactions are isolated from updates of incomplete transactions, thus creating the illusion that each transaction is the only one executed by the system. This property is often approximated by the serializability concept. The isolation of shared data in long-live computation would reduce the system performances. Isolation in the ComET model is achieved at the semantic level by using invariants (predicates shared at data level) to control access to data. These invariants must be maintained by the system during the whole ComET execution.

Durability - If a transaction is successfully completed, its effects become permanent in the database even in case of a system failure. It is the task of the system to assure consistency by automatically compensating failed transactions and resuming successful transactions according to the existent data in the ComET's log. Effects of a ComET step are durable and can be cancelled through the execution of a compensating step.

4. A ComET EXAMPLE

ComETs are suitable for the modeling of long-lived actions on the basis of ACID transactions. The execution of a ComET is guaranteed to terminate in a finite amount of time. A ComET either terminates successfully, or the original state of the ComET, or a logical equivalent state, is reconstructed via compensating actions. That is a ComET, just like any ACID transaction, accepts only the original (or a logical equivalent) state or the state after the successful execution as a final state. In order to illustrate the basic mechanisms of the ComET model we use the example of an Internet based e-shopping application (Fig1).

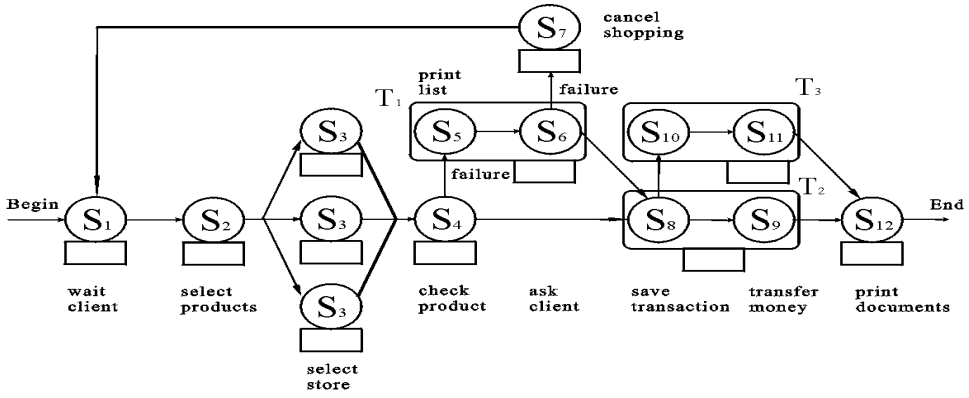

Fig. 1. A sample script "Electronic Commerce" - graphical representation

The e-shopping application is based on a scenario consisting of a number of steps representing the elementary actions of the scenario. Each of these steps is implemented as an independent ACID transaction. The steps are described as follows:

S_1 – Wait Client: wait for a client to log-in and order one or more products;
S_2 – Select Products: takes all products registered in the database;
S_3 – Select Store: for each selected product, select a store that offers it;
S_4 – Check Product: check if the selected product is available in the stock of the selected store. If the product is available go to step S_8, otherwise select next store or go to step S_5 if it is the last one;
S_5 – Print List: print the list of not existent products;
S_6 – Ask Client: ask the client if he(she) accepts to buy only the available products. If not, go to step S_7 – Cancel Shopping;
S_7 – Cancel Shopping: cancel the shopping activity;
S_8, S_{10} – Data Saving: save the transaction between the client and the store(s) in the database;
S_9, S_{11} – Money Transfer: transfers the money from the client account to the store(s) account;
S_{12} – Print Documents: print the documents.

Selecting products, selecting stores, transferring money, etc., are typical activities that can last a long time and sometimes need more than one session to be completed. Obviously, it is not possible to do the whole shopping procedure within one transaction.

5. EXECUTION SCENARIOS FOR ComETs

5.1. CONTROL FLOW

ComETs define large computations by combining existent actions (steps). Each step is defined, implicitly, as an atomic unit of work and coded independently without considering its concatenation and combination with other steps into larger units of work (ComETs) later on. It is the script programmer who is responsible for assembling the steps into a logically coherent computation. This is done via a simple script language that provides the basic flow control structures. Basically, there are two options for the processing of the script instructions: compiling or interpreting. Compiling is more

efficient, but more complicated and less flexible. Since script instructions express high granularity, but simple operations, speed is obviously not an issue. Therefore the interpretative option seems to be a reasonable one in this case due to its simplicity and flexibility. Our solution is a mixed one that offers the advantage of an intermediate representation that can be easily processed by the interpreter. The source code is scanned and parsed in order to eliminate syntax errors. The result is a representation into an internal code. When interpreted, each instruction is materialized as a node into a tree structure. This node contains all the necessary information for the correct (re)execution of the corresponding instruction.

5.2. FAILURE SCENARIOS

There are several kinds of failures that may occur during a ComET execution:

System failure - in the case of a system failure the script execution is resumed from the point of failure. This is based on the data saved in a global script log. After a system failure execution resumes with the step next to the last committed step before failure. If the failure point is inside a loop, the next step of the current iteration is executed. For example after a system crash during step S8, when all available products and their source stores have been selected, the process is continued with step S8 (that has not been committed!), that is registering the client-store transactions.

Failure in the evaluation of an invariant - if at least one input invariant of a step is not satisfied then a backtracking procedure is initiated in order to compensate the executed steps in the reverse order of their execution. Steps are compensated until a WHILE node or the applications initial step is reached. A WHILE node represents a point where several choices are available for continuing the execution. When such a node is reached, the next alternative is taken, if any, resuming execution on a new path. If there is no alternative to be taken the backtracking process is continued. If the applications initial node is reached then the entire computation fails. Compensation starts at the previous step if the invariant that triggered it is an input invariant, and at the current execution step if the invariant is an output invariant. For example, if at step S4 the selected product is not available in the current store, then compensate for the last store selection and select the next store.

5.3. COMPENSATION

Updates of a ComET are externalized at the end of each step, and can be accessed by other ComETs. This makes impossible any unilateral rollback of a ComET execution. The effects of a ComET can only be cancelled by undoing its global effects explicitly via execution of **compensating steps**. For this reason, a so-called *compensating* action has to be provided for each step in the script (rectangular boxes in Fig. 1), which semantically undoes the updates of global (database, etc.) objects. To compensate, for example, for the money transfer (Step S_9), it is necessary to perform the reverse operation and to transfer the money back to the client account, from the store account.

5.4. SYNCHRONIZATION WITH INVARIANTS

Execution of steps from different concurrently executed ComETs is synchronized by **invariants**. These are predicate expressions associated with the data

accessed by steps. The script programmer is responsible for defining the strategy of how to manage these predicates.
There are two types of invariants:

-first, each entry point into a step is protected by a set of so-called *entry invariants*. Each of these is a predicate expression, typically based on shared data in the database that must evaluate to "true" in order to actually invoke the step procedure;

-second, before terminating, each step will set up a set of *exit invariants*. This process basically binds result values of a step to the global context variables in a set of predicate expressions, thereby establishing the fact that certain conditions where fulfilled during the step execution. These exit invariants will be referred and evaluated as input invariants by subsequent steps of the same or other ComETs.

6. CONCLUSIONS

The main contribution of this work is the design of a compact and modular system that provides a generalized control mechanism for the execution of transactional long-lived and complex computations. ComETs are designed to meet the requirements that result from dividing large applications into a set of related processing steps and defining appropriate consistency and reliability requirements for the execution of the whole activity. The system is characterized by the separation of control aspects into several orthogonal dimensions, which can be exercised independently by an application using declarative techniques.

REFERENCES

[1] Chrysantis,P.K., Ramamritham, K. {1994} – Synthesis of Extended Transaction Models Using ACTA, *ACM Trans.Database.Syst.*,19, 3(Sept.), 1994, pp.450-491;

[2] Elmagarmid,A.K. {1992} – Database Transaction Models for Advanced Applications, Morgan Kaufman Publishers, Inc., 1992;

[3] Jajodia,S., Kerschberg, L. {1997} – Advanced Transaction Models and Architectures, Kluwer Academic Publishers, 1997;

[4] Reuter,A., Schneider,K., Schwenkreis,F. {1997} – ConTracts Revisited, in: *"Advanced Transaction Models and Architectures"*, Jajodia,S. and Kerschberg, L., eds., Kluwer Academic Publishers, USA, 1997, pp.127-151;

[5] Reuter,A., Schwenkreis,F. {1993} – ConTracts – A Low-Level Mechanism for Building General-Purpose Workflow Management-Systems, in: *Bulletin of the Technical Commmittee on Data Engineering*, Vol.18, No.1, IEEE Comp.Soc.;

[6] Reuter,A.,Wachter,H. {1992} – The ConTract Model, in: "*Database Transaction Models for Advanced Applications"*, Elmargamid,A.K. ed., Morgan Kaufman Publishers, Inc., USA, 1992, pp.229-263;

[7] Schwenkreis,F. {1993} – APRICOTS – A Prototype Implementation of a ConTract System – Management of the Control Flow and the Communication System, in: *Proc. Of the 12th Symposium on Reliable Distributed Systems*, Princeton (NJ), 1993;

[8] Schwenkreis,F., Reuter,A. – The Impact of Concurrency Control on the Programming Model of ConTracts, available from: http://www.informatik.uni-stuttgart.de/ipvr/as/personen/{reuter,schwenk}.html