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

Implementation of a Development System For Compensation Based Extended Transactions

Dollinger Robert, Hale Aurelian, Madularu Cristian

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

Abstract – Many Extended Transaction Models (ETMs) have been proposed in order to overcome the limitations of the classic Flat Transaction Model. Some of these models have been materialized in the form of function libraries or in prototype implementations. Our point of view is that the success of an ETM equally depends on the availability of a complete and easy to use Development System that should facilitate application development in the paradigm of the given ETM. This paper presents the most important implementation issues of a ComET (Compensation based Extended Transaction) model based Development System (ComET-DS). The ComET model is more than a simple extension to the classical transaction concept and it provides a programming model that seems to be well suited for long-lived and complex computations. This programming model consists of a coherent control mechanism that is able to coordinate the execution of several simple transactions. The ComET-DS provides several development tools, like the graphical script editor and the runtime script debugger and tracer, that are adapted to this programming model.
Keywords: extended transaction models, complex computations, implementation, development system.

1. INTRODUCTION

The limitations of the classic Flat Transaction Model stimulated an intense research activity that resulted in many Extended Transaction Models (ETMs). Some of these models have been materialized in the form of function libraries or in prototype implementations [1][2][7]. However, at this moment, there is no available system that would provide the functionality and tools to develop applications based on one of the existing ETMs. Our point of view is that the success of an ETM equally depends on the availability of a complete and easy to use Development System that should facilitate application development in the paradigm of the given ETM. This paper presents the most important implementation issues of a ComET (Compensation based Extended Transaction) model based Development System. The ComET model goes beyond the simple extension of flat transactions and provides a programming model that seems to be well suited for long-lived and complex computations. This programming model is closely related to the ConTract model, from which it is inspired [4][5][6][7][8]. The ComET programming model is based on a two-level solution for maintaining database consistency: the lower level consists of ACID transactions (called here steps), while the

higher level consists of a control mechanism that is able to provide ACID like properties to the entire application according to a flow of control expressed in a simple script language. This control mechanism is based on a few concepts like: forward recovery, compensating transactions, invariants and script execution log [3]. ComET separates the task of application development in two distinct activities: (1) development of transaction steps and their compensating transactions at the level of the database server, (2) development of the ComET script expressing the higher level logic of the application. The ComET Development System (ComET-DS) is aimed to provide functionality and assistance needed in all phases of the development of ComET scripts: editing, testing and debugging. The ComET-DS provides several development tools, like the graphical script editor and the runtime script debugger and tracer that are adapted to this programming model.

2. SYSTEM ARCHITECTURE

The implementation of the ComET-DS model follows a strictly modular design. Its structure is presented in figure 1.

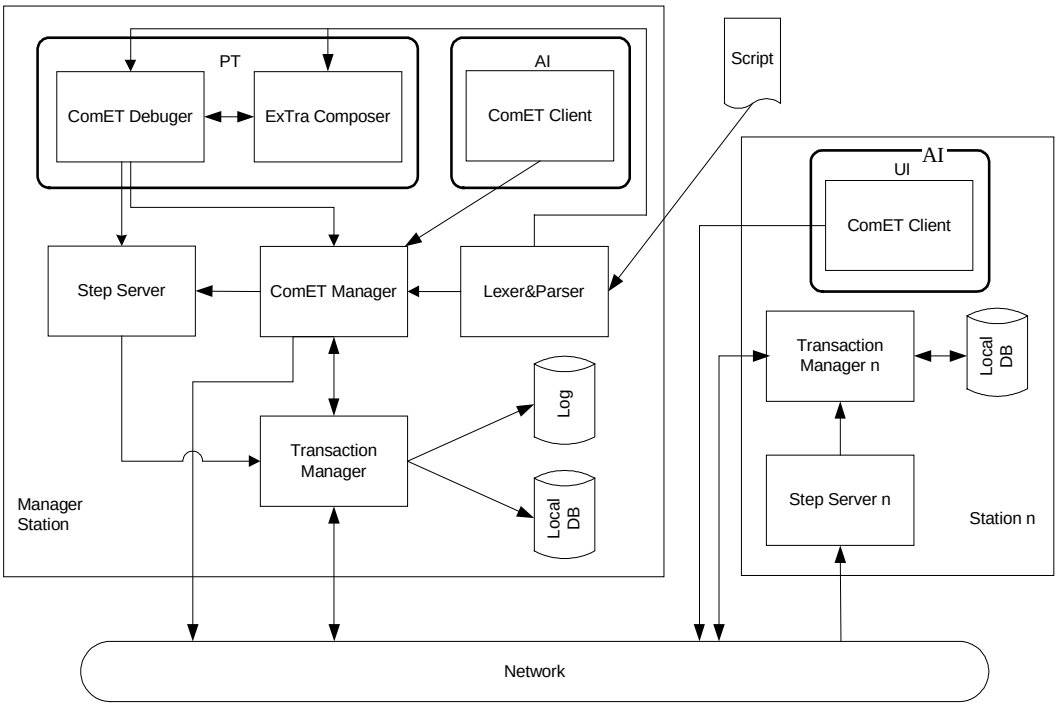

Figure 1 ComET-DS Modular Structure

The system is built around three main modules: ComET Manager, Step Server and ComET Client, implemented as DCOM services to provide system communication over the network. The other components, Lexer&Parser, ComET Debugger, ExTra (Extended Transaction) Composer and Transaction Manager, have the role to serve these services to accomplish the tasks they were build for. The Application Interface (AI) is based on a ComET Client, which is a DCOM used by applications to interact with the ComET system. It can be used to launch and execute scripts by connecting to the ComET Manager from any station in the network, setting input data and controlling step by step execution.

2.1. THE ComET MANAGER

The main component of the system is the ComET Manager that will receive and manage all user requests for script execution and control. User requests can came from two different directions: PT(Programming Tools) and AI(Application Interface). The ComET Manager guarantees the reliable execution of a started ComET and is responsible for the forward recovery after a crash. The ComET manager activates the script interpreter (Lexer&Parser) and performs the management of a global log used to resume the application in case of a system failure. Another function of the ComET manager is to guarantee for the execution and commitment of the compensating steps in case of a system failure or in case a step had failed during its execution. Script interpretation is done in two phases: a compile phase and an execution phase. A script has four main sections: (1) *step description section,* (2) *flow control section,* (3) *context description section* and (4) *atomic units of work section.*

2.2. THE TRANSACTION MANAGER

This component runs the protocol to implement the low level transactional semantics. It includes the database that stores the data needed by the application and the implementation of all the defined steps along with their compensating steps in the form of stored procedures. It can be any Database Management System (DBMS). In our approach we use for this component the Microsoft SQL Server. Bellow is a sample of a step implementation and its corresponding compensating stored procedure:

```
CREATE PROCEDURE spMoneyTransfer
       @source int,
       @dest int,
       @sum float
AS
IF (SELECT Sum
FROM ClientAccount
WHERE ClientCode = @source) > @sum
   BEGIN
        UPDATE ClientAccount
        SET Sum = Sum - @sum
        WHERE ClientCode = @source

        UPDATE StoreAccount
        SET Sum = Sum + @sum
        WHERE StoreCode = @dest

        PRINT 'Transfer completed!'
        RETURN 1
   END
RETURN 0
```

```
CREATE PROCEDURE spMoneyTransferComp
       @source int,
       @dest int,
       @sum float,
       @err int
AS
IF (@err = 1)
   BEGIN
        UPDATE ClientAccount
        SET Sum = Sum + @sum
        WHERE ClientCode = @sursa

        UPDATE StoreAccount
        SET Sum = Sum - @sum
        WHERE StoreCode = @dest
        RETURN 0
   END
ELSE
   PRINT 'Accounts have not been
updated!'
   RETURN 1
```

2.3. THE STEP SERVER

The Step Server deals with the entire house keeping related to step execution. The execution of steps is done asynchronously with respect to script execution. It communicates directly with the ComET Manager to obtain a balanced system load and to meet system performance requirements, and with the Transaction Manager, calling the stored procedure implementation of the steps. The Step Server is also responsible with the interpretation, evaluation and validation of invariants (entry and exit invariants). Execution of each step is done according to the following algorithm:

get step information;
construct, evaluate and validate the entry invariants for the current step;

if (entry invariant = TRUE)
3.1. **then** execute the stored procedure associated to the step and go to 1;
3.2. **else** execute backward compensation of last committed steps until the closest point of alternative available choices is reached;
construct, evaluate and validate the exit invariants;
if (exit invariant = TRUE)
5.1. **then** go to 6;
5.2. **else** execute the compensation stored procedure of to the current step and execute backward compensation of last committed steps until the closest point of alternative available choices is reached;
save the step context to the global context an go to 1.

3. THE GRAPHICAL SCRIPT EDITOR (ExTra Composer)

The graphical script editor provides two main functions: (1) script design and generation through a user-friendly graphical interface (direct engineering) and (2) reverse engineering of an existing script. The graphical editor is a powerful tool for easy and effective developing of ComET scripts. It provides the functionality needed to describe all the components of a script: global variables and synonyms definition, step definitions, control flow, context etc. Each script element is associated a representative graphical symbol. These symbols are interconnected according to the script structure and control flow. One can move around, resize, modify or delete any of these symbols. The properties of a script element can be set or modified by double clicking the corresponding symbol and editing the properties exposed by the interface. Figure 2 shows a small example of a visual representation and script generation.

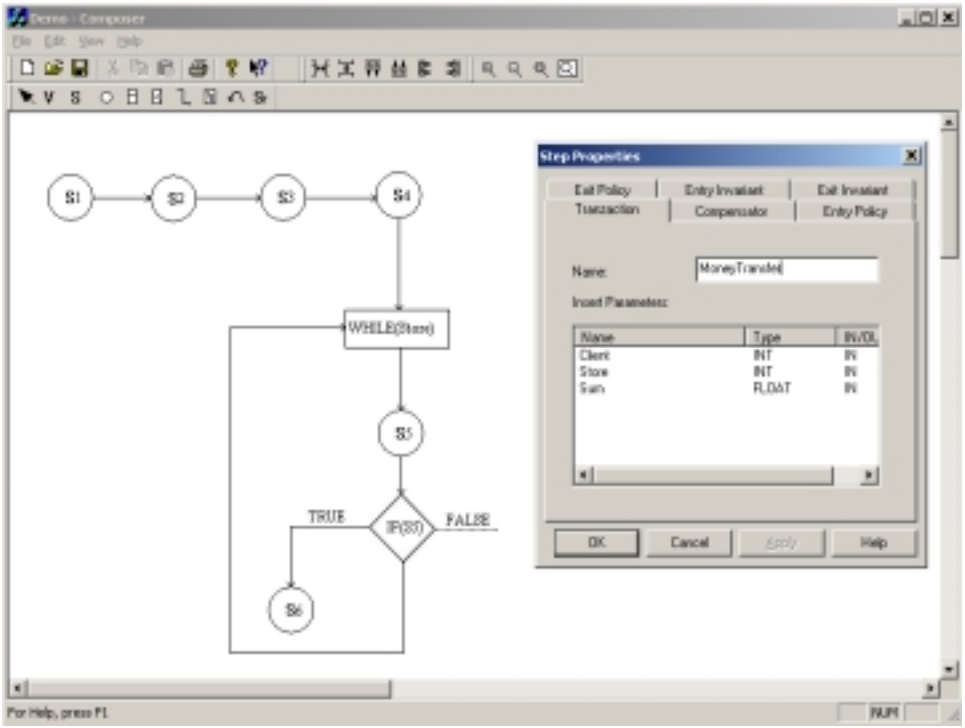

Figure 2 Graphical script-editing example

The first step the script-programmer should consider when defining a new script is the variables and synonyms definition. This is followed by the creation of the *steps* and of the control flow instructions. Finally, the CONTEXT section is automatically generated. Part of the generated script for the example in figure 2 is shown bellow:

```
STEP_DEFINITIONS:
STEP:
TRANZACTION:
        WaitClient(OUT_P INTEGER Client);
COMPENSATOR: DoNothing();
INVARIANTS:
        EXIT_I:
        EXITPOLICY:
        ENTRY_I:
        ENTRYPOLICY:

    ...

STEP:
TRANZACTION:
        MoneyTransfer(IN_P INTEGER
          Client, INTEGER Store, FLOAT Sum);
COMPENSATOR: MoneyTransfer ();
INVARIANTS:
        EXIT_I: ClientAccount < Sum
        EXITPOLICY:
        ENTRY_I: ClientAccount > Sum
        ENTRYPOLICY:
END_STEPS
```

```
CONTROL_FLOW_SCRIPT
S1: WaitClient(OUT_P Client);
S2: SelectProduct(IN_P Client;
                  OUT_P Product);
S3: UpdateClient(IN_P Client;);
S4: SelectStore(IN_P Product; OUT_P Store);

WHILE(Store)
{
   S5: UpdateDemands(IN_P Client,
                  Product;OUT_P bUpdate);
   if(OK[S5])
     S6: MoneyTransfer (IN_P Client, Store,
                                    Sum);
     Store = Store->NEXT;
}
END_CONTROL_FLOW_SCRIPT

CONTEXT
        S1->S2
     S1->S3
     S1,S3->S4
     S3->S5
     S1,S5->S6
END_CONTEXT
```

The reverse engineering component of the ComET composer offers the possibility to generate a graphical representation, through reverse engineering, of an already existing script.

4. SCRIPT DEBUGGER AND TRACER

This tool is developed to test the correctness of ComET scripts. It's a visual tool designed to offer support for checking *Script* correctness on runtime. It allows the user to visualize in a graphical way the current step of a script execution.
The script debugger and tracer provides functions like:

- interaction with the user for setting data input for the current step;
- setting breakpoints (F9) on any step of a script and watch step's variables after executing it, by expanding tree control knots;
- step by step execution(F10);
- executing script's steps from one breakpoint to another(F5);
- immediate invariant evaluation.

The steps of the executing script are visualized in a tree structure having associated a knot component to each step. These components will permanently be filled with current context values of each the corresponding step. The context can have different representation according to the step type. If type of the step is *while* or *if* then

the knots are generated dynamically and will be represented in the tree control as new instances of original knot, one instance for each iteration on the same step or new instance of it. This is a valuable feature that allows examining not only the current context of a step execution, but also its execution history. Also, when backtracking one can follow the dynamic changes in the tree control as each knot instance is deleted once the corresponding step is compensated for. This feature provides an intuitive representation of how things happen internally during execution of a ComET script. Also, one can set breakpoints from the tree control by right clicking on a knot and selecting the breakpoint option.

6. CONCLUSIONS

The most important contribution of this work consists in the development of a system architecture that allows a flexible, modular and extensible implementation of a ComET-DS. This modular structure makes easy further developments that would add new or modify existing functionality. The script development tools provided to the ComET application developers proved to be simple to use and effective in increasing the efficiency of the entire development process from design to testing and runtime debugging. Several test application where successfully developed in order to validate the system's effectiveness.

REFERENCES

[1] Anwar,E., Chakravarthy,S., Viveros,M. {1997} – An Extensible Approach To Realizing Advanced Transaction Models", in: *"Advanced Transaction Models and Architectures"*, Kluwer Academic Publishers, 1997, pp.259-276;

[2] Birilis,A., Dar,S., Gehani,N., Jagadish,H.V., Ramamritham,K. {1994} – "ASSET: A System for Supporting Extended Transactions, in: *Proc. of the ACM SIGMOD Int'l Conf. on Management of Data*, Minneapolis, USA, Minn. June 1994;

[3] Dollinger,R. {2002} - "Database Transaction Model for Advanced Applications", in: *Proceedings of IASTED International Conference on Applied Informatics*, AI'2002, 18-21 Feb.2002, Innsbruck, 2002;

[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 Computer Society;

[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. – "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