Using Triggers in a Software Configuration Manager

R. Casallas
Rubby.Casallas@imag.fr
Adele Team: anonymous ftp, ftp.imag.fr:pub/ADELE
LGI. BP 53X, 38041 Grenoble Cedex France

Abstract

Software Configuration Management is the discipline of controlling the evolution of complex systems [Tichy88]. Most of current commercial Software Configuration Managers (SCM) propose a unique pre-defined policy for the management of software. This is quite restrictive because the Software Process is highly evolutive; it must be adapted to each organization (its own policies and constraints) and evolve in parallel with changes in the organization.

This paper presents the integration of active capabilities in a SCM as a solution to this problem. The use of a triggering mechanism, to implement these active capabilities, shows that it is possible to model and support a wide range of software management strategies. Therefore, the system is highly adaptable and can be customized to users specific needs.

This work has been realized in the Adele Configuration Manager. The Adele data model is an object-association, the objects represent software components and the associations are relationships between them. Both are stored in a multi-user and multi-version repository. The Trigger mechanism is not only used for integrity enforcement of data, but also, to support the definition, execution and control of software management policies.

Keywords: Configuration Management, Active Databases, Triggers, Software Process, Cooperative work.

1. INTRODUCTION

Software Configuration Management (SCM) is needed to manage and maintain large and constantly evolving software products [Tichy88], [Dart91]. Software products evolve to cope with new requirements, to enhance their functionalities, to adapt to new machines, new operating systems, new customers, or simply to fix bugs. This evolution, the Software Process, is the result of a large set of concurrent activities, performed by teams, over shared resources [Feiler91, Belkhatir93].

The Software Process is usually complex; it involves, during a long period of time, several activities performed concurrently by numerous agents over shared entities. A formalism and mechanism to model and automate a wide range of software management policies is clearly needed in this context. Most current commercial SCM [Sun88], [Rochkind75], [Munch93] [Adams93] [Cagan93] propose a unique pre-defined policy for the management of software. This is quite restrictive because most organizations (1) have to manage huge amount of existing software and (2) have existing people, methods, tools and know-how which constitute most of its capital.

1. Adele is distributed by Verilog Inc., 3010 LBJ freeway, suite 900 Dallas, Texas U.S.A.
   Verilog SA 52 ave. Aristide. Brian 92220 Bagneux France
In practice it is not feasible, and often not even desirable, to change the software product and/or the company methods to cope with the requirements of a SCM. Instead, the SCM must be programmed and customized to adapt itself to the company. The solution to this problem is to integrate in the system an interpreter of an explicit formalism in which the company characteristics and needs can be described such that the SCM policies can be automated and enforced. Such an environment is called a Process Centered Environment. We propose the building of such an environment based on a reduced number of concepts and mechanisms: a triggering mechanism integrated in an object-association data model.

Software artifacts can be shared by different activities, e.g., development, validation, documentation, etc. The behavior of the components can not be statically defined, because it depends on the activity where they are used. For example,

- when a module is in a development activity, edition is allowed, and compilation uses specific flags and libraries.
- when a module is in a validation activity, edition is not allowed, and compilation uses different flags and libraries.
- when a module belongs to a released configuration, edition is not allowed, and specific consistency constraints and actions are defined.

Adele is an open Software Engineering Environment that supports Software development and evolution in a multi-user and multi-version context. The Adele kernel is basically composed of: an object manager, transactional support, version and trigger mechanisms. Adele is a highly customized and adaptable system. All other basic SCM services like Configuration Management, Work Space control, change control and so on have been easily built using these basic services. Users can define and implement their own software management strategies and product model in exactly the same way.

The active capabilities have been integrated in Adele by means of Triggers. Triggers can be defined both on object types and on association types. Triggers on association types are a way to extend and redefine the behavior of objects; they support the implementation of active behavior. This is an original aspect not supported by other systems.

The rest of this paper is structured as follows. The second part presents the basic concepts of the Adele Data Model and the query language. The third part presents how active capabilities have been integrated into Adele. This paper concludes with a comparison with other work in the database and software engineering domain and with future perspectives.

2. THE CORE DATA MODEL

SCM have to represent the resources managed by the software process: users, activities, files, documents, policies, constraints, etc. Although these resources are heterogeneous, all of them require the same kind of basic management operations, for instance they need:

- symbolic unambiguous identification
- to be managed either as a whole, or each one of its component individually
- to hold different kinds of associations with other resources, etc.

Most of these requirements are simply satisfied using an object representation formalism, since expressiveness and extensibility are facilities offered by this kind of representation. Furthermore, the object-oriented approach allows the SCM to represent and manage in an uniform manner all types of resources. We believe that data modelling, in Software Engineering, must make explicit at a conceptual level the associations between objects.
Indeed, as will be shown later, our system make heavy use of associations. For that reason the Adele data model extends the object formalism with association, objects and association being treated in almost an homogeneous way.

This section presents the basic concepts of the Adele object-association data model. We also give some features of the query language.

2.1 BASIC CONCEPTS

Objects and associations. Objects represent software components and associations represent directed relationships between them. Despite the high grain variability of software components, all entities are represented in a uniform way. Objects can represent files, activities, functions as well as simple values like strings or dates. Associations are independent entities (external to objects) because they model relationships with different semantics such as derivation, dependency or composition. An association is established from an origin object to a destination object.

Object and Relation Types. A Type describes the common structure and the behavior of the instances of that type. The structure corresponds to a set of attributes modelling properties of the instance. Relation Types have additional information related to the domain of the relation, i.e., a constraints defining between which object types the association can be established. The behavior is encapsulated into operations (methods) and extended by means of triggers.

Object/Relationship identity. External names must be assigned to objects at creation time. Named objects are persistent roots. There is a difference between physical object identification and logical object identification. As in [Kifer92], physical object identification is an implementation notion and the user refers to objects via logical object ids. An association is also identified by a unique name. This name comes from the concatenation of the Origin object name (O), the Relation type name (R) and the Destination object name (D). It is designed by the string: O|R|D.

Attributes. Attributes model the properties of the instances. An attribute is defined by a name (identification), its domain type (integer, date, boolean, string, enumeration) and other information such as the default value, the initial value, if the attribute can be multi-valued or not, if its value is dynamically computed, etc.

Multiple Inheritance. Types are organized in a type hierarchy describing specialization. A type can have one or more super types (i.e multiple inheritance). The subtype relationship is a relation of inclusion, i.e., if type T1 is a subtype of type T0 then the extension of T1 is included in the extension of T0.

Methods. Methods are used to model the behavior of the instances. They are defined by a name, a signature and a body or implementation. Methods are inherited along type hierarchy and their implementation can be redefined. The implementation is written using the Adele language. It is an imperative language which allows other methods and basic operations to be called using basic conditional and repetitive instructions.

Triggers. Triggers are inherited along the type hierarchy but unlike methods, they can not be overloaded. A complete presentation is contained in section 3.

1. When confusion does not arise, we use the term "instance" to refer either to an object or an association and "type" to refer either to an object type or a relation type.
**Versioned Objects.** The Adele Version Model is based on the branch concept. A branch models the evolution of an object. It is a sequence of revisions where each revision contains a snapshot of the object. Different kinds of derivation graphs can be defined. Arbitrary versions and composite objects are created and managed using explicit relationships between the different components [Estublier94]. An in-depth description of the version model is beyond the scope of this paper.

### 2.2 Query Language

The object manager offers (1) basic operations to manipulate objects and associations: instance creation and destruction, attribute update, version manipulation, configuration construction, and (2) a language for querying the Adele database.

Objects and the associations between them can be seen as graphs: objects are the nodes and associations the arcs. The query language uses this fact to reach the instances through navigation on graphs. It also offers a filtering mechanism for selecting the instances on which the operations will be applied.

Querying the database is realized using path expressions. They are defined by a navigation expression and a selection expression:

```
path ::= navigation [( selection )]
```

**Navigation.** Navigation allows to reach the instances starting from a set of instances and following a path through the graph. A navigation expression defines a multiple path on a graph. As associations are directed, navigation can be done from the origin object to the destination object (noted \( \rightarrow \)) and in the other direction, from the destination to origin (noted \( \leftarrow \)):

```
navigation ::= nodes -> arcs | nodes <- arcs | navigation path
```

**Nodes.** It denotes the set of objects from which the navigation starts and **arcs** the set of associations to cross. This means that the navigation is done on various paths at the same time. For example:

```
nodes = \{ O_1, O_2, O_3, O_4 \}  \ 
D_6 <- arcs = \{ O_3, O_4 \}
```

```
nodes -> arcs = \{ D_1, D_2, D_3, D_4, D_5, D_6 \}
```

```
O_0 -> R_0 > R_1 > R_2 = \{ D_1, D_2, D_3, D_4, D_5, D_6 \}
O_0 -> R_0 > R_7 < R_3 = \{ O_3 \}
```

**Selection.** A selection allows to filter the set of instances reached by the navigation. It is a logical expression where the variables are instances attributes. The expression is evaluated for each instance in the set returned by the navigation. The result of this evaluation is the instances for which the expression is true. If there is not a selection expression, the initial set is returned.

**Substitution** allows to access the values of instances attributes. The syntax is:
3. ACTIVE CAPABILITIES

3.1 MOTIVATION

Software Processes involve, during a long period of time, several concurrent activities performed by numerous agents over shared resources. In order to support cooperative work among these activities, a SCM must allow to model and automate a wide range of software management policies.

Activities share data but they do not use them in the same way. Each activity has its own perception of the resources and has specific requirements for their manipulation. This characteristic is called the contextual behavior of data. The SCM should, ideally, respect and even enforce these different perceptions, by providing for each activity the adapted viewpoint of data.

An approach to deal with the contextual behavior problem is to use a system having active capabilities. An "active system" allows to automatically fire activities in reaction to particular events and circumstances. This active behavior can be specified by means of triggers. As we will see latter, the trigger mechanism is powerful enough for allowing to model software processes with different scenarios and control policies.

In this section we present the trigger mechanism in the Adele system. This mechanism has been built in the kernel of the system and takes advantage of the object-association data model to reduce the search space of triggers to be activated. Indeed, triggers being defined both on objects and associations, the activation process is performed by propagation through the graph composed by the instances.

We will first show the trigger definition language and how it is integrated within the data model. Then we will present the trigger execution model. Two complete examples are given in the fourth section.

3.2 TRIGGER DEFINITION

Triggers are defined within types and they are inherited along the type hierarchy. A trigger definition is as follow:

```
OBJTYPE TIS::

   Activation point
   ON eventname DO
   Action

END T;
```

In the case of relation types there are two additional definitions: triggers related to events raised in the object origin of the associations and triggers related to events raised in the object destination of the associations.
RELTYPE R IS:

<table>
<thead>
<tr>
<th>Activation point</th>
<th>Triggers related to relationships of type R</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON eventname DO</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Action</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Activation point</td>
<td>Triggers related to origin objects</td>
</tr>
<tr>
<td>ON ORIGIN eventname DO</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Action</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Activation point</td>
<td>Triggers related to destination objects</td>
</tr>
<tr>
<td>ON DEST eventname DO</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Action</td>
<td></td>
</tr>
<tr>
<td>END R;</td>
<td></td>
</tr>
</tbody>
</table>

**Activation point.** When a method is called, the system opens a transaction to execute it. A method execution always raises events at some specific points of its associated transaction. These points are called activation points and can be: at the beginning of the transaction (PRE triggers), before the transaction commit (POST triggers), after commit (AFTER triggers) and after rollback (ERROR triggers).

PRE triggers are activated before the method. They enable the system to check objects and the system state before operation execution, to extend the method by prologues and so on.

POST triggers are executed after the method, but before the validation of the corresponding transaction. Their aim is to analyze the database results, to assert the new state of the database, to undo (rollback) the modifications performed by the operation, to add more computation, to chain with the next action to execute and so on.

AFTER triggers are executed after the validation of the transaction. They can be used to notify the validated operation and to take historical information, to chain with further actions and so on.

ERROR triggers are executed if the transaction is aborted, as an exception mechanism. They allow to take an alternative action.

**Eventname.** It is the name of an eventmask. An eventmask is a condition to filter the system state. It is a first order logical expression where variables are related to objects or association attributes using the query language presented in the section 2.2.

The priority level determines the execution order when multiple triggers are simultaneously fired.

```
EVENTMASK eventname = Condition, PRIORITY nn;
```

**Action definition.** Any program in the Adele Language, as the method implementation.

### 3.3 TRIGGER EXECUTION MODEL

When an event is raised, the system fires the action of the triggers for which the condition of the eventmask is true.

**Triggers eligible for activation.** The instance on which a method has been invoked is called the reference instance. This instance plays a central role because it determines the triggers to be activated. There are two cases: the
reference instance is either an association or an object.

**Reference instance is an association:** In the following figure, we have an association called: O1\(\triangleright\)R\(\triangleright\)O2.

![Diagram of association O1\(\triangleright\)R\(\triangleright\)O2]

If a method is applied on this association, the system searches the triggers for activation into the ON clauses of the definition of the R relation type.

**Reference instance is an object:** The eligible triggers are searched, in its object type definition and also in the relation type definition of the associations for which this reference instance is currently either the destination or the origin object. For example, in the following figure, the reference instance O is the origin of O1\(\triangleright\)R\(\triangleright\)O1, and the destination of O2\(\triangleright\)R2\(\triangleright\)O2. The triggers will be searched in:

- definition of the object type T0,
- definition of the relation type R1 (ORIGIN clause)
- and, definition of the relation type R2 (DEST clause)

![Diagram of triggers O1\(\triangleright\)R\(\triangleright\)O1 and O2\(\triangleright\)R2\(\triangleright\)O2]

This propagation mechanism is used to model contextual behavior. Indeed, the behavior of an object can be extended and/or modified dynamically, depending on its associations.

**Trigger execution.** At each activation point, the system searches the set of eligible triggers. These triggers are ordered according to priority level of the eventmask that fired them. They are executed sequentially in this order. The execution of the triggered actions can create new sub-transactions because other triggers can be activated in reaction to new events. These new sub-transactions are nested in the initial one.

4. **EXAMPLES**

In this part we present two examples using the triggers in Adale. The first shows as the trigger are used to integrity enforcement and the second to model a synchronization protocol among concurrent activities.
4.1 INTEGRITY ENFORCEMENT

Assume that AdlConf3 is a configuration and its type is called TConf. TConf has, among its attributes, a state attribute which indicates the configuration state, e.g., stable, in-development, in-test, in-documentation. If the configuration is stable (state = stable), its components cannot be modified. Its components are objects (instances of heterogeneous types, e.g., programs, binaries, documentation, etc.) and are related to AdlConf3 via associations, instances of compconf relation type.

Triggers can be used to prohibit modifications on the configuration components. We can use the compconf relation type to define a trigger related to the destination objects, i.e., the configuration components. The transaction will be aborted if there is a modification attempt:

```
RELATIONTYPE compconf IS ...;
   DOMAIN (type == TConf) -> (!type == Object)
   POST
      ON DEST modified DO {
         IF (-10%status == stable) THEN
            ABORT;
         }
      }
   END compconf;
```

4.2 SYNCHRONIZATION PROTOCOL

Assume that a software product must be enhanced with new functionalities. Assume that each new functionality will be developed by a different user. They will share the initial software product or baseline. Each user will work in his own work space. If a user needs to change a module, a copy is realized into his work space to work in isolation. When the modification is finished, the baseline will be updated by the integration of the modified modules. We want to propagate this modification to all work spaces containing a copy of the same original module.

RELATIONTYPE propagate IS ...;
Using Triggers in a Software Configuration Manager

Synchronization protocol

DOMAIN
(type == Object) -> (type == Object)

POST

ON ORIGIN updatessource DO {
FOR d IN !O->propagate
IF (d <= !D) THEN
update (d, !D%source)
}

END propagate;

EVENTMASK updatessource = (!method = update and !attr = source), PRIORITY 1;

In the figure, the user in the work space 1 update a module using the method update on the module in the baseline:

update (M1, -MWS%source)

the reference object is M1. A cause of the association M1 propagate Mref, the system search in the relation type propagate, the triggers defined on the ORIGIN clause (M1 is the origin of the association). The action to fire is the block:

FOR d IN !O->propagate
IF (d <= !D) THEN
update (d, -!D%source)

All objects destination of the propagate relationships
to propagate the modification

5. CONCLUSIONS AND FUTURE WORKS

Most current commercial Software Configuration Managers [Ovum93] propose a unique pre-defined policy for software management. This is too restrictive, since the SCM must adapt itself to the company, and further, the Software Processes permanently evolves, to cope with the organization evolution in term of people, methods, tools and management policies.

We presented in this paper how the integration of active capabilities in a SCM is a solution to this problem. This active facility has been specified by means of triggers. The use of a triggering mechanism shows that it is possible to model and automate a wide range of software management strategies. Therefore, the system is highly adaptable and can be customized to specific users needs.

A triggering mechanism has been built into the Adele kernel to take advantage of its object-association data model. In this model, all software components are represented in a uniform way, using an object paradigm, and associations are modelled at conceptual level. Thanks to triggers defined in relation types, the behavior of the objects can be extended and modified to deal with a particular viewpoint of the data; it is the way object contextual behavior has been implemented.

The Adele system, in use for years, has proved its usefulness in the context of real applications. The Adele system has been used in real applications including the development of Adele itself. From the experience two main areas of improvement have been identified.

The first one concerns kernel improvement, extending the event expression by temporal features, even composition, enhanced transaction recovery possibilities, and providing a better integration with the version concept.
The second one deals with Process modelling and support. It has been found that triggers produce a fragmentation of the process into a large number of tiny fragments into the different types definitions, making it difficult to have a clear perception of a complete process description. The next task will be to make the use of the active capabilities easier by defining a high level Software Process language. This language will allow users to describe the different software management policies in terms of concepts more abstract than object, triggers and association, then translated into our triggering mechanism.

6. REFERENCES


Ovum93 Ovum. P. Ingram, C. Burrows, I. Wesley
Configuration Management Tools: a Detailed Evaluation

Paton93 N. Paton, O. Diaz, M. Howards Williams
Dimensions of Active Behaviour

Rochkind 75M. Rockkind
The Source Code Control System.

Sun88 Introduction to the NSE.

Tichy 88W. Tichy
Tools for Software Configuration Management.
International Workshop on Software Version and Configuration Control.
Grassau FRG. Jan 88. Published in Springer Verlag.

Wiebe93 D. Wiebe
Object-Oriented Software Configuration management.
Proc 4th Software Configuration Management workshop.