Reusing software processes

Jacky Estublier
L.G.I. BP 53x Grenoble 38041 France

It is advocated here that the most critical aspects for modeling and control, in a large Engineering Environment, are the inter/intra team communication and synchronization. A purely OO approach does not solve the problems found when multiple processes share objects. We propose a solution where the combination of an active relationship with sub-databases (Work Environments) provide the necessary flexibility; the semantics for communication and synchronization can be captured by active relationships.

Our approach is essentially based on three concepts:

- **Object Oriented** technology.
- **Work Environment**: A WE is a sub-database created for a user to fulfill an activity. It is filled with the relevant objects, the tool and the processes for the activity.
- **Active relationships**: An active relation type defines attributes, triggers and methods. A relation instance is a triple (Origin, Relation type, Destination). A relationship may execute methods or triggers when either its origin, destination or the relationship itself receives a message.

Since Adele relationships are binary, a method M on a relation type R may be defined in the following way: (1) “DEST M...” or (2) “ORIGIN M” or (3) “M...” which means that when message M is received by (1) the object DESTination (2) the object ORIGIN or (3) the relationship itself of a R relationship instance, we must execute the following code. If M is also defined in the receiving object, it is ignored; the M method is dynamically overloaded.

By definition, in a sub database, a relationship (and its methods) is visible only if its origin and destination objects are visible. Suppose the “test” method defined in relationship “test” between a test file and the module it tests. The method is available only if both the module and the test file are visible, which occurs only in the validation WE.

Object Orientation, Relationship and Work Environment concepts, all together, allow a separation of concern.

- **OO** defines the structural and behavioural properties of objects of the same class. All instances of a class are identical with regard to these properties which are true regardless of the context in which instances are used.
- **Active Relationships** define additional properties for the objects they relate; they allow addition, substitution and delegation of properties as well as information flow in both directions.
- **Work Environment** define the visible objects and relationships, and by the way, which are the characteristics (attributes, methods, triggers, constraints) relevant in a given WE. It should be noted that “tasks” are implemented as objects with relationships to its input/output. A task is active only in the WE that contain it.
This separation of concern has a large impact on the way processes are described. A relationship turns to define the dynamic semantics between entities or group of entities. In Adele they are powerful enough to define it completely, often regardless of the type of the related objects. Classic semantics, as found in SEE data bases, are also provided as standard. For instance the PCTE link categories such as composition, reference, implicit or stabilize are basic relation types in Adele (a single description line for each).

```
RELTYPE stabilize; ON DEST [ !modified = true ] DO ABORT;
```

Which means: when the DESTination of the relationship is modified "!modified = true" then undo "ABORT" the action that did the change. In Adele the relation roles type need not be defined (as here for relation stabilize), thus a stabilize relationship (or any of its subtype) may be instantiated between any pair of objects, as for instance between a configuration and its components.

Consider inter WE relationship. In Adele we can define "busy-propagate" relations between two copies of an object, with as semantic "when a copy is modified, propagate immediately the change to the other copy"; "conditional-propagate", "lazy-propagate", "auto-resynch" etc. Relation types can be defined with other semantics of the relationship between two copies of the same object.

An engineering environment evolves rapidly. Each object may be part of different activities (dynamic aspect) and be a component of different complex objects (static aspect). In the following example, suppose A and B are of the same type: C program. B not being used in any configuration has no consistency constraint to verify, but conversely it does not have the methods "archive" and "release", associated with configuration components, but it has the methods associated with the relation in_WE. ...

The behaviour of an object depends heavily of the context in which it is used. An OO approach cannot model this since all instances must have the same methods; the only possibility is to subtype C type, for all possible use of C instances, which, because of the combination of possibilities, is intractable. It would need dynamic type change and would impose type definition change each time a relation semantics is changed.