Tempo:
Enhancing OO Paradigm for Modeling Software Engineering Processes

Noureddine Belkhatir Jacky Estublier Walcélio L. Melo
Laboratoire de Genie Informatique
BP 53X, 38041 Grenoble Cedex France
{belkatir, estublier, wmelo}@imag.imag.fr

Published in the *Proc. of 8th Int'l Soft. Process Workshop*, Berlin, Germany, Feb. 1993. IEEE CS Press.

1 Position

Large scale software development and maintenance involves large teams working simultaneously, often in different geographically distributed sites. This situation raises high demanding requirements on SEEs. To attain such requirements, various researches have been done in the domain of data integration and centralized control using integrating platforms. These platforms provide support for product structuring, versioning, software configuration processes, and other engineering processes. From this experience, the software engineering community agrees that the entity-relationship-attribute data model (ERA) extended with Object-Oriented (OO) concepts is, now, the most adequate for modeling software artifacts. Standards interfaces such as PCTE ECMA and CAIS, and by various SEEs like ADELE, ATIS, EIS and PMDB+ are the result of this trend.

Typically these SEEs do not provide enough services to support software processes. In current SEEs, embedded processes control the use of system capabilities, coordinate concurrent activities, and enforce team communication policies; but in a quite inflexible way because the policies supported by the system cannot be tailored. SEE activities are carried out in poorly managed or loosely controlled working areas implemented above a standard file system, e.g, Unix.

These is a large consensus to say that SEE should provide explicit support for the expression and control of software processes. Software processes should be explicitly represented and, in part, automatically executed by a SEE.

Our position is summarized by two observations:

1. O.O technology is a useful and realistic approach to software process modeling. Software process models can be based on the objects involved in the software process since they provide operations (methods), and can be under the control of a configuration manager. Each controlled and identifiable activity in the software process can also be modeled as an active object.
2. OO technology alone is not sufficient to provide all the required features to support software process.

This position paper will try to show some of limits of the O.O. technology and how they can be overcome. Our position is based in our experience with Adele[1] and our work enhancing Adele to better support software processes.

2 Discussion

This position is based on the hypothesis that a SEE platform should be designed as a **process engine** able to interpret a language in which "any" particular software process formalism can be translated. Such a language must be able to model the static information (the passive objects) used during the software process, and the active entities (the software processes).

From that hypothesis, we shall highlight requirements that O.O. paradigm should address for supporting software processes: 1) multiple structuring 2) Traceability and time constraint management.

2.1 Multiple structuring

The only structuring concept supplied by the O.O. paradigm is the class concept. This model supports only one object behavior description, which can be refined by specialization using the class structure. All applications are supposed to agree in that structure, and by consequence in that behavior.

In our case, we would like that each process model has its own data and behavior description (the model) of the objects it manages. Object models are context dependent; for instance an object is seen and behave differently when managed by an implementation process, a test process or a validation process [2].

Unfortunately, to fulfill this requirement, the basic O.O. models must be extended. This extension has been realized in the Adele project at two levels: extending the Adele kernel to provide the needed basic mechanisms, as discussed in [2], and developing a high level language, TEMPO [4], on top of Adele kernel in which the description of an object depends on its **role** [3] in a process.

In TEMPO, a process is a set of roles, and each role is the set of object instances sharing the same static and dynamic description in that process (properties, methods, constraints, access rights, etc.). Thus a role defines the model of objects playing that role. In a given process, each object instance plays a single role, but different roles may use objects of the same type.

TEMPO associates a **work environment** (WE) to each software process step. A work environment is the place where the process performs (tools, automated processes and project members). It looks natural that the characteristics and behavior of manipulated resources depend on the WE type in which they are used. A given object can simultaneously be part of different WEs, and play a different role in each WEs. For instance a piece of program can be part of a development WE, a test WE and a validation WE.

2.2 Roles and classes

Roles and classes look similar; it rises the question: can roles be implemented in term of classes and sub-classes?, or better saying, is the concept of role needed at all?

A role, as well as a class, is a set of instances sharing the same definition (static and behavioral). A given object instance can be simultaneously a member of different roles (classes). Both roles and classes can be seen as a viewing mechanism since a given object instance has a different description depending on the role (class) from which it is managed.

However the differences are the following: The association between an instance and its class(es) is statically defined at instantiation time, while an instance can be dynamically bound to an arbitrary role at any time. In an O.O. system the class definition is created first, and then the instances of the class; while in TEMPO, usually, the instances are created first, and are dynamically associated, for a while, to a (set of) role.

Since a given object instance can be simultaneously a member of different roles (classes) there are compatibility rules between those roles (classes) allowed to shared objects. For classes it is the *inclusion semantics* constraint which holds between a class and its sub classes (ISA relationship). For roles it is essentially a synchronization and coordination semantics. A wide range of such semantics exist, this is why the language does not impose any predefined semantics. Relationship between roles, and their semantics, is user defined.

The role concept is a generalization of the class concept, intended for another purpose: the control of the multiple views of objects instances and the coordination of their concurrent use. As a special case, classic O.O. systems can be implemented in term of roles, not the contrary.

2.3 Traceability and time Constraints

A software process executes, in general, for a long time. Since different processes execute simultaneously over the "same" objects, new difficulties need to be solved.

The long duration of processes (spanning over days, weeks and months), make it necessary to manage persistent objects (stored on persistent media, not main memory). This requirement can be fulfilled either by a standard file system, or by a data base. Unfortunately file systems do not provide attribute, relationships (out of links), recovery mechanism nor transaction facility. Clearly a data base is needed somewhere.

Sharing an object between processes, along with the long duration of processes, make unpractical the standard transaction mechanism as found in data bases. This is why we associate a Work Environment to each process; a WE is a sub data base.

The long life duration also needs a time management. Temporal constraints must be expressed. These constraints are interpreted by the system using two mechanisms: the versioning of object (each time an object is changed, a new version is created), and an explicit history mechanism (each time a given condition holds, predefined information is stored). The system is able to fetch the needed information in order to answer queries like: are all tests been executed, what was the result of a given action, what was the requirement which originated this change. The data base clearly needs to be designed to fulfill theses very special requirements [5].

Pure O.O paradigm lacks concepts to express constraints which are generally encapsuled in methods. We claim that some constraints need to be extracted from methods and associated with other concepts (roles in TEMPO). Constraints play different functions: 1) associated with objects, they define general integrity constraints applied to all object of the same type; 2) associated to a **role**, they control its contextual behavior.

3 Conclusion

This approach raises an interesting question: since roles are like dynamic types for objects, should the underlying system support classic O.O. description, or should roles control completely object description. If not, what is the boundary between class/role responsibility; how to manage eventual conflicts.

References

[1] N. Belkhatir, J. Estublier, and W. L. Melo. Adele 2: a support to large software development process. In M. Dowson, editor, *Proc. of the First Int'l Conf. on the Software Process*, pages 159–170, Redondo Beach, CA, October 21–22 1991. IEEE CS Press.

[2] N. Belkhatir, J. Estublier, and W. L. Melo. Software process model and work space control in the Adele/Tempo system. In L. Osterweil, editor, *Proc. of the 2nd Int'l Conf. on the Software Process*, pages 2–11, Berlin, Germany, February 1993. IEEE CS Press.

[3] N. Belkhatir and W. L. Melo. The object role software process model. In J.-C. Derniame, editor, *Proc. of the 2nd European Workshop on Software Process Technology*, volume 635 of *LNCS*, pages 150–152, Trondheim, Norway, 7–8 September 1992. Springer-Verlag.

[4] N. Belkhatir and W. L. Melo. Tempo: a software process model based on object context behavior. In *Proc. of the 5th Int'l Conf. on Software Engineering & its Applications*, pages 733–742, Toulouse, France, December 7–11 1992.

[5] W. L. Melo. *Tempo: Un environnement de développement Logiciel Centré Procédés de Fabrication.* Thèse de Doctorat, Université Joseph Fourier (Grenoble I), Laboratoire de Génie Informatique, Grenoble, France, 22 de Octobre 1993.