Modeling of Software Process Assistance

N. Boudjlida - F. Charoy - J.C. Derniame - C. Godart*
CRIN, Université de Nancy I
Campus Scientifique, BP 239
F-54506 Vandoeuvre-les-Nancy
France

Abstract

Software project development can be improved if assistance is provided to ensure furtherance in development. This paper reviews different kinds of assistance one can expect from a software development environment. Then, we propose concepts for sofware process modeling and we show how the resulting modeling can be used to achieve the different kinds of assistance. We relate our proposal to the framework of our investigations, namely the ALF ESPRIT project.

Keywords : Software process modeling, assistance, object model, operator model, rules, ordering, model for assisted software process

*Proceedings of CASE'89, London (This work is partially sponsored by the Commission of the European Community under ESPRIT Project ALF Ref. No. 1520)

Modeling of Software Process Assistance N. Boudjlida - F. Charoy - J.C. Derniame - C. Godart
C.R.I.N.
Nancy I University - FRANCE

1 The need for a Model for Assisted Software Process

A current trend in software development is to try to provide the different "actors" in software development (managers as well as developers) with "sophisticated" assistance mechanisms [ALF 87] [Ambras 88] [Derniame 88a] [Kaiser 87] [Kaiser 88]. Assitance mainly aims at guiding in furtherance of the software development, warranting the appropriateness of an activity, taking the initiative to perform an activity instead of the user,... A step toward this aim is to provide a description of all what we know about software development activities [Osterweil 87] [Perry 88] and about how we would like them to be performed, and then to have accurate mechanism which use this knowledge for providing different kinds of assistance.

We shall call this description *Model for Assisted Software Process*, abbreviated into MASP further [Derniame 88a]. A MASP describes the way software processes have to be executed and the organization of the objects on which software processes act. So, the MASP is a static description of the system's expected behavior and not of its effective behavior.

This paper presents in the second section the main kinds of assistance we feel necessary in a software environment. The third section contains the presentation of the concepts we propose for software process modelling. The fourth section shows how we intend to use such a model to ensure the different kinds of assistance which are mentioned in the second section. The fifth section relates our proposals to the ALF [ALF 87, Benali 89] ESPRIT project which is the framework of our investigations.

2 Functionalities of third generation IPSEs.

This section proposes a representative set of software manufacturing assistance functions a third generation IPSE [Lehman 87] must provide with. We distinguish here different kinds of assistance with respect to the three main classes of roles in software manufacturing: assistance in the software development itself, assistance in project management and assistance in software process modeling.

2.1 Software development assistance.

A software developper wants an IPSE to assume the following functions:

- **To do work:** an IPSE must be able to execute process, to operate on the objects of the software in development. An IPSE must assume at least the classical Operating System role.
- **To take initiatives:** it is the main request of software development industrials. An IPSE must be able to take initiatives, i.e to execute and to combine operations without user's stimuli and user's interaction. Of course, some operations need user interaction (editing for instance), but many of them does not (syntactic control, statistical computing, ...).
- **To control:** on one hand, an IPSE must protect the user from his initiatives which come too early with respect the software state (a user must respect the software development process). On the other hand, it must protect him from going beyond its rights (a user must adopt a role and stay in this role until he explicitly changes his role).
- **To guide:** at any moment, an IPSE must be able to guide a user in his task. This kind of assistance has different facets.
  - *What can be done ?:* an IPSE must be able to say a user what operations can be performed, to propose to a user the set of operation he can undertake. The fact that an operation is undertakable or not depends on the state (the value) of the objects in the user environment.
  - *How does it work ?:* an IPSE must be able to provide a user with information which are more than explanation about a tool functioning (the classical help in operating system). The IPSE is expected to explain to a user how to get an object of a particular type. In other words, it must be able to find possible operation combinations which leads to this type of object.
  - *How to do it ?:* an IPSE must be able to deduce what operation combination has to be performed in order to produce a specific object of a specific type considering the actual state of the existing objects.
  - *What is the effect of it ?*: this function allows a user to know "a priori" the consequence of an operation, considering the existing objects and their relationships. For instance, the modification of a source text has some impact on all the source texts which use the modified one. So, an IPSE should provide with a kind of simulator which enables a user to measure all the possible consequences of one of his initiative.
- **To explain:** when the IPSE cannot perform or complete an operation, it must be able to explain to the user the reason of the failure and to advise him on the way to avoid this failure.

- **To allow user training:** the "what can be done", "how does it work ?", "how to do it?" and "explain" functions seem to be very suitable for user training to achieve efficient development.
- **To assume communication:** the development of a large software needs the interaction of a great number of people. Communication is one component of their comprehension and their cooperation.

2.2 Project management assistance.

From a general point of view, project management also needs to perform operations and the assistance functions we studied in the previous subsection are also applicable to project management. However, these operations are mainly devoted to measure the work which has been done and the work which has to be done in regard with objectives and milestones. So, on one hand observation has to be done during software development and data has to be extracted at specific instants in order to control project evolution. On the other hand, as these measurements are done in respect with metrics (cost estimation, user profile management ...), the same data can be considered as historical and statistical data and can be used to build new metrics and to refine existing ones. **So, an IPSE must provide a project manager with observation capabilities and with historical and statistical data gathering functions.**

2.3 Software process modeling and validation assistance.

No assistance can be achieved if the software process which is used for software development is not modelled. The next section is devoted to the problem of software process modeling. But from now on, we can assert that software process modeling is complex enough to need assistance and that a third generation IPSE must provide tools for such an assistance: **dedicated editors and validation tools for software process modeling are fundamental components.**

3 Proposal of a meta-model for software process modeling

The previous sections emphasized the need for software process modeling to achieve software development as well as assistance in software development and management. This section suggests a meta-model for such a purpose.

The software process model should involve different kind of knowledge :

- knowledge provided by the description of the **operators** which can be used to perform activities,
- knowledge issuing from the description of the **objects** on which the operators act,

- knowledge which concerns the allowed sequence of operators,
- knowledge which describes how the system should behave in peculiar circumstances.

Particular states of the system are identified and described by means of **expressions**. A subset of these expressions is called **characteristic**. The system is in charge to ensure that these characteristics are satisfied in the related system's states.

Compulsory behavior of the system is described by means of a set of **rules**. Each rule associates an operator to an expression. The expression identifies the state of the system in which the corresponding operator may be executed.

Thus, we propose to model a software process by means of a Model for Assisted Software Process (MASP) which encompasses six components **(Om, Op, Ex, Ru, Or, Ch)** where :

- *The object model* **Om** defines the object structures by means of object properties and relations among objects. The object description model is based on the entity-relationship data model [Chen 76] extended with the subtype relationship.
- *The operator model* **Op** informs about the operators which can be executed to perform an activity or part of an activity. The description of the operators involves :
  - the condition underwhich the operator is executable **(precondition)**,
  - the condition which holds after the operator has been applied **(postcondition)**. Preconditions and postconditions are expressions (see below).
  - the object types on which the operator is allowed **(operator's domain)** and the object types produced by the operator **(operator's range)**.

  Furthermore, each operator can be described itself as a MASP.
- *Expressions* **Ex** is a set of formulas which describe particular states of the system.
- *Ordering* **Or** describes total or partial orders among operators. Ordering of operators is a mean to explicitly express policies to be enforced during the software development. **Or** enables the specification of allowed operator sequences, operators simultaneity, operators repetition, and so on...
- *Rules* **Ru** is a set of two-sided rules. The left hand side is called the condition part of the rule. It is written down as an expression of Ex. The right hand side of a rule is called the action part of the rule. It involves an operator of Op. So, **Ru** describes what to do (as indicated by the action part) when the system reaches states described by the condition part (for example when a given object does not conform to one of its expected states).
- *Characteristic* **Ch** is a subset of Ex; this subset describes safe states of the software under development. The assisting software environment is in charge to ensure that **Ch** is true at checkpoints of the system.

We want these MASP components to play *complementary roles* in assisted software development. The next section shows how we intend to make these components interact with respect to the assistance functions described before.

4 Cooperation of components in assistance

For each functionality, we try to identify which components of the MASP [Derniame 88b] are involved in and how they are. We do not refer to the underlying mechanisms which are required to ensure the functionalities because we feel that it is not under the scope of this paper.

Furthermore we classify the functionalities into a class of *working functionalities* which involves doing, observing and controlling functionalities and a class of *assistance functionalities* which involves guiding, taking initiative, explaining and learning functionalities.

4.1 Working functionalities

4.1.1 To do

The operators are the means which enable the user or the system to do something. So, after the invocation of an operator, the system has first to check wether the operator is activable or not i.e. is its precondition true and is the invocation conform to the ordering ?

Assistance at this level can be found on simple explanation on how to invoke an operator, that is, what is its syntactic profile, what is its behavior, what is its result.

4.1.2 To observe

Observation can be done by means of :

- an observation schema which describes where to store and retrieve observation data,
- a set of observation rules which allow to express *When to observe*, and *What to observe*. *When to observe* may be expressed in the condition part of the rules, while *what to observe* could be specified by the action-part of the rules.

Further, *When to observe* may concern each operation which modifies the object base or each beginning or end of a session, or each beginning and end of an operation.

What to observe can concern the evolution of the base, or the operations invoked.

So, the components needed for observation facilities are mainly :

- Rules for specifying what is to be observed and when the observation should take place,

- Objects where observation data are stored,
- Operators as event-generators for observation.

The results of the observation activity will serve for the control and the assistance functionalities.

4.1.3 To control

Integrity constraints can be defined on the object schema. Only primary assistance can be given upon this component, such as why a modification cannot be performed. But we assume that no deduction can be made if an operation violates these integrity constraints.

Further, characteristics are used to ensure consistency of the base at some predefined checkpoints (inactive period, choice of the user, end of a session, ...)

The operators are also concerned by the control functionality in the following way. An operator invokation succeeds only if the following conditions are satisfied :

- the operator exists in the MASP
- the invokation is syntactically correct
- the type of the operands in the invocation conforms to the types specified in the operator's domain
- there exists a binding between an operator and an operation which is the implementation of the operator
- the precondition is true
- the operator doesn't violate the ordering (that is, the operator is involved in the ordering and all the preceding operators have been successfuly invoked or the operator is not involved in the ordering).

So, for the control functionality, the following components are concerned :

- Operator definition, for syntactical and semantical verification,
- Ordering to ensure that an operator invokation does not violate an ordering,
- Characteristics to be verified or to be enforced,
- Object models if there definition involves the definition of integrity constraints.

The kind of assistance which may be provided is mentioned in the section devoted to the explanation facilities.

4.2 Assistance functionalities

4.2.1 To guide

Guidance functionality involves mainly three facilities :

- What to do next,
- How does it work,
- How to do it.

What to do next This facility requires the knowledge of the objects "owned" by the user who needs the facility. So, we assume that those objects are identifiable in the user's active-context. Thus, the available objects and the precondition of operators enable the determination of a preliminary set, say S1, of operators that can be performed. It is the set of operators whose domains involve the type of the available objects and whose preconditions are true.

The set S1 can be restricted to a set S2 which involves the only operators allowed by the ordering in the current state.

Thus, the new set S2 is the set of all the operators that can be invoked. But we are not informed about the exact operator to be invoked, i.e. which operator in S2 is the most appropriated in the current state. At this level, we may adopt different strategies, such as :

- propose only the operators that are involved in the ordering
- propose the operators the most in advance in the ordering, according to a kind of "deapth first" strategy [Nilsson 82]
- propose the operators the less in advance in the ordering, according to a kind of "breadth first" strategy.

Further, the *What to do next* facility cannot be invoked at any time, but only in inactive period (when the user has the initiative, and when all triggerable actions have been performed) for consistency reasons. Thus the rule component of the MASP may also be concerned by this facility and we assume that their action parts are invoked before answering to "what to do next".

In summary, the "what to do next" functionality appeals to the object base, to the operator definitions, to the ordering and to the rules. It may also appeal to a kind of "meta-rules" to choose the most appropriated operator within a set of candidate operators. Furthermore, we consider that the answer to "what to do next" is a one-step deduction i.e. it delivers only the next operator(s) which may be immediately applied and not a sequence of operators.

How does it work To fulfill this functionality, one may consider only the ordering. The answer could be the sub-part of the ordering which starts at the beginning of the ordering and which ends at the node(s) which involve an operator that produces an object of the target type.

Another approach to fulfill this functionality is to start from the target object-type and to try to find all the graphs that lead to it using the preconditions and the postconditions of the operators. These graphs can be restricted by the ordering.

Can the rules be concerned by this functionality since some of them may produce alternative paths to the target type or to an object-type which is involved in a path that leads to the target type ? If so, one can consider that part of the paths that lead to the target type can be "followed" automatically.

Another problem which could be addressed at this level is the one which concerns the choice of the "best" path to be followed to reach the target type. Some heuristics can be used like retaining the shortest paths or the ones which involve the greatest number of automatic operators (i.e operators which need not interaction with the user).

So, this functionality needs the ordering, the operators and perhaps the rules components of the MASP.

How to do it This functionality can be ensured as the previous one, except that the starting point is the type of the object for which the functionality is invoked.

4.2.2 To take initiative

Initiative taking is primarily based on the rules : the condition part specifies in which circumstances the initiative specified in the action part is to be taken.

The operator description and mainly the kind of the operator (automatic or not) helps in deciding whether the initiative can be taken in presence of the user or not.

Further, as "many" initiatives can be candidate at a given moment, one could order them according to a priority associated to the rules involved in the initiative or according to the kind of the operators involved in the action parts of the candidates.

Moreover, the rules can help the system in undoing on its own decision a series of actions as a consequence of the invocation of an operator. In the same way, the system can take the initiative to evaluate the consequences of an operator-invocation and inform the user of these consequences. In this case, the system makes use of the operator definition and of the links that hold between the objects which are concerned by the operator and other objects within the object base.

4.2.3 To explain

Explanations can be provided in different situations :

1. When the invocation of an operator is rejected : this situation arises when the profile or the precondition of the operator are not obeyed or when the invocation is not allowed by the ordering (it is too soon or too late to do it) or when the binding to an operation cannot be done. So, such a situation makes use of the operator descriptions and the ordering.

2. When the system takes an initiative : In this case, the system should explain why it decided to take the initiative. Explanations could be generated from the rules that determine the initiative. The interpretation of the condition part provides the reason of the initiative while the action part provides the aim of the initiative. A little bit more explanation can be provided when ordering is used to restrict the set of candidate rules to the ones whose action-parts do not contradict the ordering or when any strategy has been followed to order the initiatives which candidates in the given situation.

3. When the system cannot avoid the violation of a characteristic : When the system cannot establish a characteristic (i.e. it cannot find any means to make the characteristic true) the interpretation of the expression which defines the characteristic is the basis for explanation generation. More precisely, since a characteristic may be a compound expression and not only an atomic expression, one should find which sub-expression of the characteristic is false and try to interpret it.

4.2.4 To learn

This functionality can be regarded from two standpoints. The first one concerns user training while the second one concerns machine learning.

User training may concern each category of users of an ALF system. By training we essentially mean "education" of the users. So it may concern education on MASP design as well as utilisation of a MASP.

Machine learning is classically meant as the ability of a machine to aquire knowledge from what is already known and from past activities. For the moment, we feel that this aspect is a quite difficult one since it is still a fundamental research field in artificial intelligence.

5 ALF functions vs IPSEs function.

ALF is not intended to be an IPSE, but it wants to be an environment kernel based on PCTE [Gallo 86] and designed to develop IPSEs provided with software development assistance features. In other terms, ALF does not propose one model of assisted software process but a meta-model for assisted software process modeling. It does not propose an algorithm for each assistance function, but basic mechanisms to program these functions at low cost. ALF offers choices for assistance design: choice of concepts for a particular software process modeling and, in dependence with this first choice, choice of algorithm for each assistance function. But, on one hand, software process modeling and validation functions must be (and will be) built-in functions of the ALF system, and on the other hand, ALF will be provided with observation capabilities and with historical and statistical data gathering functions.

References

[ALF 87] ALF. Advanced Software Engineering Logistics Framework. *Technical Annex*, 1987.

[Ambras 88] J. Ambras and V. O'Day. Microscope : a knowledge-based programming environment. *IEEE Software*, May 1988.

[Benali 89] K. Benali, N. Boudjlida, F. Charoy, J-C. Derniame, C.Godart, Ph.Giffiths, V.Gruhn, Ph.Jamart, A. Legait, D.E. Oldfield, and F. Oquendo. Presentation of the **ALF** project. In *Proceedings of the International Conference on System Environment Development and Factories*, Berlin, Germany, May 1989. To be published.

[Chen 76] P.P. Chen. The Entity-Relationship model: Toward an Unified View of Data. In *ACM-TODS 1,1*, pages 9–36, Mars 1976.

[Derniame 88a] J.C. Derniame, B. Ayoub, K. Benali, N. Boudjlida, C. Godart, and V. Gruhn. Toward assisted software process. In *Proceedings CASE88*, Boston - MA, July 1988.

[Derniame 88b] J.C. Derniame, K. Benali, N. Boudjlida, J.J. Chabrier, R. Champagne, J. Garbarossa, P. Griffiths, and V. Gruhn. *The components of the MASP*. ALF/CIG-RCH/WG6.1, october 1988.

[Gallo 86] F. Gallo, R. Minot, and I. Thomas. The object management system of PCTE as a software engineering database management system. *ACM SIGPLAN Notices*, 22(1), 1986.

[Kaiser 87] G.E. Kaiser and P.H. Feller. An architecture for Intelligent Assistance in Software Development. In *Proceedings of the 9th Int'l Conf. on Soft. Eng.*, Monterey - CA, March 1987.

[Kaiser 88] G.E. Kaiser, P.H. Feller, and S.S. Popovitch. Intelligent Assistance for software development and maintenance. In *IEEE - Software*, 1988.

[Lehman 87] M.M. Lehman and W.M. Turski. Essential properties of IPSEs. *ACM SIGSOFT, Software Engineering Notes*, 12(1), January 1987.

[Nilsson 82] N.J. Nilsson. *Principle of artificial intelligence*. Springer Verlag, 1982.

[Osterweil 87] L. Osterweil. Software Processes are Software Too. In *Proceedings of the 9th Int'l Conf. on Soft. Eng.*, pages 2–13, Monterey, CA, March 1987.

[Perry 88] D.E. Perry and G.E. Kaiser. Models of software development environments. In *Proceedings of the 10th Int'l Conf. on Soft. Eng.*, Singapoor, March 1988.