A Meta-CASE Environment for Software Process-centred CASE Environments

Flávio Oquendo[†], Jean-Daniel Zucker[‡] and Philip Griffiths*
[†]CRISS - Pierre Mendès France University
BP 47, 38040 Grenoble Cedex 9, FRANCE
[‡]GIE EMERAUDE
C/o BULL, BP 3, 78430 Louveciennes, FRANCE
*ICL Defence Systems
Eskdale Road Winnersh, Wokingham Berkshire RG11 5TT, UK

Abstract. There has been a recent explosion of interest concerning the construction of computer-aided software engineering environments assisting users during the software development process. Such environments, called process-centred, are characterised by their ability to provide some assistance or automatisation of the software process being carried out. This paper describes the ALF Meta-CASE environment developed in the framework of the ALF ESPRIT project. It consists of a formalism for modelling computer-assisted software processes and mechanisms for supporting the generation of process-centred CASE environments. These environments are able to enact formal software processes models and to assist developers during their enaction.

1 Introduction

The software crisis has been with us for some time and will continue to worsen, at least for the next few years. Two important reasons for this are that there is an ever increasing demand for software, and there are demographic problems, i.e. there are fewer and fewer people available to write this software. This latter reason will be alleviated to some extent as the third world enters the market; but this still leaves us with a severe productivity problem.

The field of software process modelling is one of several that have developed to address the software crisis. Examples of other important areas are in programming technology, design methods, and to some extent in hardware. The last of these has allowed the software community to free itself to a large degree from the shackles of hardware related constraints.

Much has been written elsewhere on the importance of software process modelling [Dowson 1991, Osterweil 1987, Stenning 1987]. But software process modelling also has the great advantage that it brings the other advances in software technology together in a way that allows them to complement each other.

We call *software process* the total set of software engineering activities needed to transform user's requirements into software. A *software process model* is a software process definition that can be instantiated for a specific project or organisation. Different software processes can be driven by the same software process model. The formal notation for describing software process models is called *the software process formalism. Software process enaction* is the action of enacting software process models. *Enacting* means executing by an agent that may be a human or a machine.

In the ALF[1] project [Benali et al. 1989, Griffiths et al. 1989], our approach to support software process modelling has been to develop a *meta* computer-aided software engineering environment: a meta-CASE environment for generating software process-centred CASE environments. A process-centred CASE environment is a knowledge based CASE environment that is able to take initiatives and assist software developers according to the knowledge it has of software process models to be carried out. An important aspect is that the meta-CASE environment will be able to generate environments to support many software life-cycle models, e.g. the "waterfall model" [Royce 1970] or "the spiral model" [Boehm 1986], and many design and development methods, e.g. HOOD or SADT. More information on how our approach compares to related work can be found in [Arbaoui and Oquendo 1991].

The main objectives of ALF generated CASE environments are:

- *to support process modelling and enaction,*
- *to provide user assistance.*

Other aims are:

- to support the whole software production process,
- to support process decomposition to the level of tools,
- to model the whole organisation to the level of roles,
- to be multi-project, multi-team, and multi-user,
- to support user communication and cooperation.

The technique for software process modelling that has been developed in ALF is generic. Furthermore the approach taken is not restricted to one paradigm; there is enough functional richness to be object oriented, to be rule based or to be constructive. Genericity is achieved by instantiation, that is where an object or operation is described in a model, the object instance or actual tool to be used does not have to be identified, i.e. instantiated, until it is needed. The "tool" might not be an actual tool but another sub-process, which, of course, can be modelled.

---

1. The ALF "Accueil de Logiciel Futur" project is partially funded by the Commission of the European Communities under the ESPRIT programme, Project Ref. No. 1520.

The following section of this paper looks at the formalism for describing software processes that has been developed in ALF. Later on the instantiation and enaction of process models are discussed. There follows a brief description of the ALF meta-environment architecture. Finally the ALF System is presented from the user's point of view, with its main contributions to increase productivity and software quality.

2 The MASP Process Modelling Formalism

An important thread of the work in the ALF project has been to design a formalism for software process modelling [Oquendo 1990]. The formalism we have developed, called *the MASP* (meta-*Model for Assisted Software Processes*), provides the means to describe rigorously computer-assisted software process models. These descriptions are *enactable* and can be used to provide better *understanding* of software processes and *communicating* software process models to people such as developers and managers involved in their enaction.

The MASP concept provides mechanisms that support the description of generic software process models which can be incrementally and repeatedly instantiated in order to produce particular software process models specific to projects or organisations. In this way software process models can be easily *reused.*

The *evolution* of software process models is supported by the MASP concept through mechanisms for interleaving instantiation and enaction. This allows changes to be made "on the fly", to software process models. These may be a consequence of changes in the organisation structure, project policies, changing deadlines, and so on; also these may be as a consequence of the feedback mechanisms provided by the underlying environment in order to tune the software process models being enacted.

The facilitating of the *management* of software processes is supported in the MASP concept by means of mechanisms for scheduling the software process activities, controlling their enaction, and monitoring their progress.

An assisted software process model is described by a MASP in terms of object types, semantic constraints to be enforced on the objects of these types, operator types defined on these object types, and control on the execution of the operators of these types.

A MASP description [Derniame et al. 1991] is composed of six models (see figure 1):

- an object model,
- an operator model,
- an expression model,
- an ordering model,
- a rule model,
- a characteristic model.

```
masp =
    MASP specification
    description
    END MASP ';'

specification =
    identifier HAS TYPE operator_type
    ':' (domain_and_range ';' )

description =
    object_model_definition ';'
    ( expression_model_definition ';' )
    operator_model_definition ';'
    ( ordering_model_definition ';' )
    ( rule_model_definition ';' )
    ( characteristic_model_definition
    ';' )

domain_and_range =
    '(' ( parameter { ';' parameter } )
    ')'

parameter =
    par_access par_list ':' par_type

par_access =
    IN | OUT | INOUT | READ |
    READWRITE

par_list =
    parameter_name { ','
    parameter_name}

par_type =
    object_type_name
    | relationship_type_name
    | attribute_type_name
```

Fig. 1. MASP specification (i.e. operator's signature) and description

The first two of the MASP components are the object and operator models, the expression model is only a means of defining conditions to be reused on the other models, the other three models (i.e. rule, ordering, and characteristic) compose the control model.

The first component of a MASP is the object model (see figure 2). The chosen formalism for expressing the object model is based on the PCTE+'s OMS [Oquendo et al. 1991].

The MASP object model, which is structurally object-oriented, enhances the PCTE+'s OMS data model with new mechanisms including structured and multi-valued attributes, triggers and semantic constraints [Oquendo et al. 1990]. It provides a distributed object base and includes features for managing single and composite objects and their versions [Oquendo et al. 1989a, Oquendo et al. 1989b].

A MASP object model defines an object base schema by means of a list of schema definition sets (SDS) (figure 2). Each SDS gathers a set of related object, relationship and attribute type definitions and extensions. An important feature of the SDSs is the sharing of

type definitions amongst SDSs. This is provided by the importation facility. The importation of a type definition from an SDS into another makes this definition visible in both SDSs. In this way, type definitions can be shared amongst several SDSs. The view of the object base defined by the MASP object model is given by the set of type definitions which is the union of all of the type definitions in the SDSs of the list. A new object type is always defined as a subtype of (i.e. a specialisation of) one or more other types. In addition to its applied attribute and relationship types, the new defined type inherits all of the applied attribute and relationship types of its supertypes. A predefined object type, called `object`, is the common ancestor type of all object types. The structure formed by these subtypes constitute a connected direct acyclic graph having only one root, the type object. Multiple inheritance is therefore provided.

```
object_model_definition =
    OBJECT MODEL IS
    list_of_schema_definition_sets ';'
    END OBJECT MODEL

list_of_schema_definition_sets = sds { ';' sds}

sds = sds_name | sds_definition | sds_extension

sds_definition =
    NEW SDS sds_name IS type_list ';'
    END sds_name

sds_extension =
    EXTEND SDS sds_name WITH type_list ';'
    END sds_name
```

```
type_list = type { ';' type }

type =
    object_type_definition
    | object_type_extension
    | relationship_type_definition
    | attribute_type_definition
    | type_importation

object_type_definition =
    ot_name ':' SUBTYPE OF object_type_names ( WITH
    ( ATTRIBUTE attribute_type_ d_list ';' )
    ( LINK relationship_type_d_list ';' )
    END ot_name )
```

Fig. 2. MASP object model

The second component of a MASP is a set of expressions (see figure 3). They are logical expressions which describe particular states of software processes. The chosen formalism for specifying expressions is based on first-order predicate calculus.

Expressions, like assertions in IPSE 2.5 [Warboys 1989], are only a means of defining a set of conditions which are then used in many places within a software process model, without the need to re-define them on each occasion. The basic terms in the description of expressions are events (on objects or operators) and logical expressions. In MASPs, expressions are used in pre- and post-conditions of operator types, in rules and characteristics.

The third component of a MASP is the operator model (see figure 4). It is defined by a set of operator type definitions and importations. An operator type describes a class of operators. The chosen formalism for expressing operator types is similar to, but an extension of, the ones adopted by Marvel [Kaiser and Feiler 1987] and Inscape [Perry 1987]. In these, pre- and post-conditions are only used for describing the semantics of the software process activities that are performed by software tools. We extend this approach by relaxing that restriction of applying pre- and post-conditions only to activities performed by tools, and thus allowing also to describe the semantics of complex activities described by MASPs. As in Marvel, pre- and post-conditions are the basis for backward and forward reasoning.

```
expression_model_definition =
    EXPRESSION MODEL IS
    list_of_expressions ';'
    END EXPRESSION MODEL

list_of_expressions = expression { ';'
    expression }

expression =
    event_definition
    | logical_exp_definition
    | expression_definition

expression_definition =
    ex_name ':'
    ON event_d { ';' event_d }
    EVALUATE logical_expression_d
```

```
event_d = event_name |
    event_description

event_definition =
    event_name ':' EVENT
    event_description

event_description =
    user_defined_event_situation
    | read _event_situation
    | update_event_situation
    | create_event_situation
    | delete_event_situation
    | move_event_situation
    | convert_event_situation
    | expression_event_situation
    | invoke_operator_event_situation
    | exit_operator_event_situation
    | time_event_situation
```

Fig. 3. MASP expression model

A software process model is described by a hierarchy of MASPs. Indeed, a MASP has a specification (i.e. an operator's signature) and contains a set of operator types which in turn can be described by MASPs, and so on (see figure 5). So each MASP describes a part of the software process model at an appropriate level of abstraction. The binding between an operator type and a MASP or a software tool and the MASP instantiation are done at enaction time. New MASPs can also be created.

The fourth component of a MASP is a set of orderings (see figure 6). Orderings express restrictions on the execution of operators. They are used to describe whether two specified operators must be executed sequentially, alternatively or concurrently. They are also used to describe whether an operator can be executed repeatedly, arbitrarily often, or if it can be executed simultaneously with other occurrences of itself.

```
operator_model_definition =
    OPERATOR MODEL IS
    list_of_operator_types ';'
    END OPERATOR MODEL

list_of_operator_types =
    operator_type { ';' operator_type }

operator_type =
    operator_type_definition
    | operator_type_importation

operator_type_definition =
    op_name ':' (domain_and_range)
    PRECONDITION ':'
    logical_expression_d
    POSTCONDITION ':'
    logical_expression_d
    KIND ':' '-' | INTERACTIVE | NON
    INTERACTIVE
```

```
logical_expression_d =
    log_exp_name |
    log_exp_description

logical_exp_definition =
    log_exp_name ':'
    log_exp_description

log_exp_description = ...
    "It is a logical expression built
    using the logical connectors
    AND, OR, NEGATION, and
    IMPLICATION, where variables
    may be typed and universally or
    existentially quantified..."
```

Fig. 4. MASP operator model

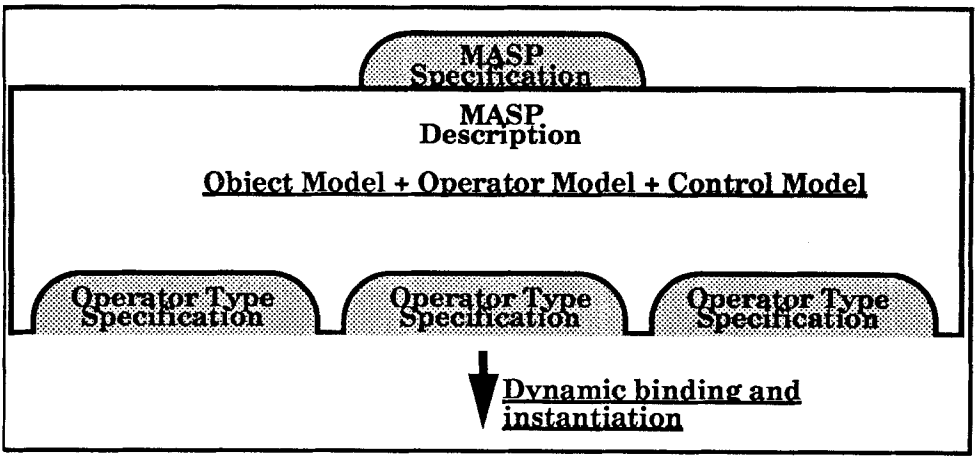

Fig. 5. Binding of operator types and MASP instantiation are done at enaction time

```
ordering_model_definition =
    ORDERING MODEL IS
    list_of_orderings ';'
    END ORDERING MODEL

list_of_orderings =
    ordering { ';' ordering }

ordering =
    ( or_name ':' ) path_expression

connection =
    FOR ALL variable ':'
    object_type_name
    ( IN ( variable | constant ) ':'
    object_type_name )
```

```
path_expression =
    ( connection DO ) operator_exp
    | ( connection DO ) br_path_exp
    | ( connection DO ) bi_path_exp

br_path_exp =
    '{' path_expression '}'
    | '(' path_expression ')'
    | '(*' path_expression '*)'
    | '(' path_expression ')'
    | '(' path_expression ')' counter

bi_path_exp =
    path_expression '||'
    path_expression
    | path_expression '|'
    path_expression
    | path_expression ';'
    path_expression
```

Fig. 6. MASP ordering model

The chosen formalism for expressing orderings is based on indexed path expressions [Campbell and Habermann 1974]. In a path expression (figure 6) ';' means sequential execution, '{' '}' simultaneous execution, '| |' concurrent execution, '|' alternative execution, '(*' '*)' optional execution, '(' ')' repeated execution, and '(' ')' grouping of operators. An ordering does not define a unique sequence of operator executions. It does define the set of all possible sequences of operator executions. In other approaches such as in Marvel and HFSP [Katayama 1989] the scheduling of operator executions is deduced implicitly from the data dependencies amongst operators.

The fifth component of a MASP is a set of rules (see figure 7). The chosen formalism for expressing rules is based on production rules. They define explicitly the possible automatic reactions to specific situations arising during the software process. In approaches such as in Marvel a user request may produce a chain of reactions which dies down eventually. In these approaches only a user request can cause this chain of reactions. This differs from our approach where this chain of reactions can be caused by a user request or by a specific situation that triggers rules (i.e. system initiative).

```
rule_model_definition =
    RULE MODEL IS
    list_of_rules ';'
    END RULE MODEL

list_of_rules =
    rule { ';' rule }

rule =
    ( ru_name ':' )
     IF expression_d
    THEN operator_name '(' (
    parameter_list ) ')'
```

```
expression_d =
    event_d
    | logical_expression_d
    | expression_name
    | expression_description

parameter_list =
    par { ',' par }

par =
    variable | constant
```

Fig. 7. MASP rule model

The sixth, and last, component of a MASP is a set of characteristics (see figure 8). The chosen formalism for expressing characteristics are MASP expressions. They describe constraints on the software processes' states that should be enforced during their enaction. In other approaches these are generally limited to constraints on the software product states. Backward and forward reasoning on the pre- and post-condition of operator types can be used in order to deduce a sequence of operator executions to react to characteristic violations. Another possible reaction is to abort the operator execution.

```
characteristic_model_definition =
    CHARACTERISTIC MODEL IS
    list_of_characteristics ';'
    END CHARACTERISTIC MODEL

list_of_characteristics =
    characteristic { ';' characteristic }
```

```
characteristic =
    ( ch_name ':' )
    logical_expression_d
    | expression_name
    |expression_definition
```

Fig. 8. MASP characteristic model

A very simple example of a MASP description is presented in figure 9. This MASP describes a process model for editing, compiling, and linking C programmes.

```
MASP c_prog_pm HAS TYPE
c_prog_develop:
    ( IN _program : c_program ;
    OUT _exec : exec_module ) ;

OBJECT MODEL IS c_prog ;
END OBJECT MODEL ;

OPERATOR MODEL IS

    edit :
    ( INOUT _m : c_module );

    compile :
    ( IN _m : c_module ; OUT _o :
    object_module )
    PRECONDITION: edited (_m,
    TRUE)
    POSTCONDITION : o (_m, _o)
    KIND : NON INTERACTIVE ;

    link :
    (IN _p : c_program ; OUT _e :
    exec_module )
```

```
    PRECONDITION : FOR ALL _m :
    c_module SUCH THAT c (_p, _m, _)
    AND compiled (_m, TRUE)
    POSTCONDITION : exec (_p, _e)
    KIND : NON INTERACTIVE ;

END OPERATOR MODEL ;

RULE MODEL IS
    module_linked : IF CREATE LINK exec
    (_p, _e) THEN set (linked, _p, TRUE) ;

END RULE MODEL ;

ORDERING MODEL
    order : DO ( link(_program, _exec) ) ;
    exit(_exec)

END ORDERING MODEL ;

CHARACTERISTIC MODEL IS
    exec(_program, _exec) ;

END CHARACTERISTIC MODEL ;

END MASP ;
```

Fig. 9. A very simple example of a MASP description

3 The Software Process Instantiation Mechanism

A MASP description defines a generic process model that can be incrementally instantiated in order to produce project-specific software process models. When a MASP has been designed there is usually not only one MASP, but a hierarchy of MASPs (figure 10). A MASP describing a complex development method or a complex software process model usually includes a lot of complex operator types which are again described by using the concept of MASP.

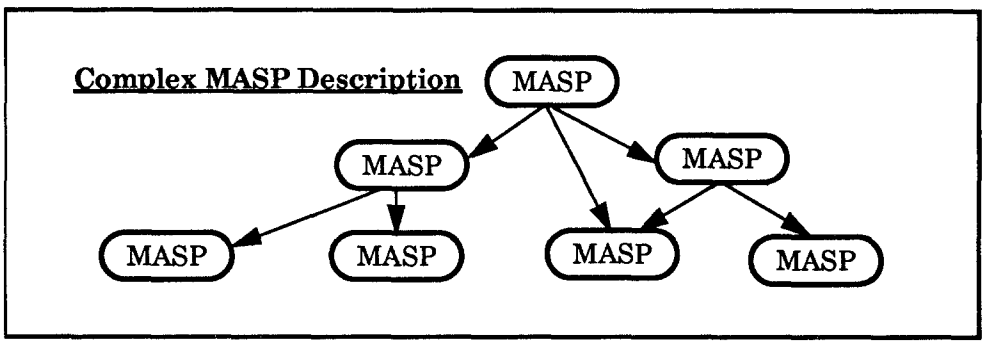

Fig. 10. A complex software process model is described by a MASP hierarchy

An approach one can think of for *instantiation of MASPs* is that the complete MASP hierarchy is instantiated and then the enaction of the resulting instantiated MASP hierarchy can begin. [Lehman 1987] points out that such a way of working (i.e. "static" instantiation) is not flexible enough. A software process may last for a long time (weeks, months, and even years) so it is not reasonable to demand that a complete instantiation must be done before the start of the enaction. Another disadvantage of static instantiation is that there is no chance to tune the behaviour of the software process to special situations; everything is fixed at the beginning and it is impossible to react to deviations of the current software process model from what was expected. Since the aim is to be able to evolve the software process model "on the fly", static instantiation is unacceptable, "dynamic" instantiation is therefore needed.

The overall approach for MASP instantiation and enaction is depicted in figure 11. Our approach is to interleave instantiation and enaction. This approach provides the possibility to consider the part of the development that has been executed before instantiating a further part. It takes into account the dynamism of process models. An *instantiated MASP* is called an *IMASP*.

The instantiation mechanism consists of instantiating the object and operator models by defining object instances and actual operators. The instantiation process is supported by a dialogue between the instantiation environment service and the MASP instantiator. There is the possibility of re-using an instantiated MASP to instantiate an operator type. The flexibility of this approach lies in the fact that a given MASP can be instantiated many times and that different instantiated MASPs can share objects.

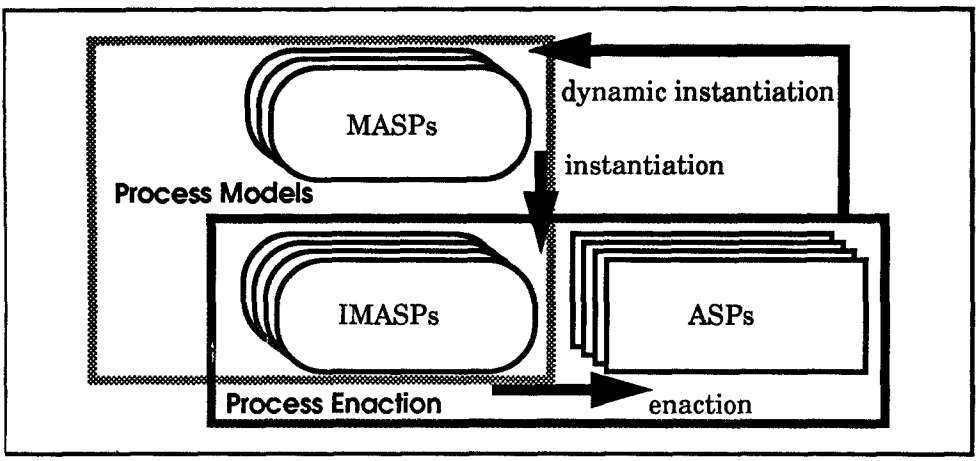

Fig. 11. MASP approach for instantiation and enaction

Associated to each IMASP there is an object set [Oquendo 1991]. An object set is a logical object base having as schema the MASP object model associated with the IMASP. It can be defined in such a way that some objects in the set are shared with some other object sets. Each software process enacting an IMASP accesses these shared objects through its MASP object model view point. A shared object then becomes the synthesis of several different software process activities, where each one contributes to build the object in function of the software process model view point it enacts. Concurrent access to shared objects is controlled by a nested transaction mechanism [Oquendo et al. 1991].

We distinguish two cases for instantiating object types:

- Object sharing. If an object exists in the object set of an existing IMASP and if its object type is compatible with the object type to be instantiated, then the object can be inserted in the object set of the MASP under instantiation. Cooperation between IMASPs is achieved by means of object sharing.
- Creation of new objects. Object sharing is not the only way of instantiating an object type. The other way is to insert new objects of that type explicitly during the instantiation dialogue.

During the instantiation of the operator model of a MASP it is necessary either to identify CASE tools (or any other kind of executable software) that can be linked to operator types or to identify MASPs that are of the corresponding types. Here we face the problem that instantiation is based on assumptions about tools and MASPs. Each available tool and each MASP is typed. A tool is typed by the person who brings that tool into the ALF meta-environment. For instance, let us take the operator type `compile` in the MASP `c_prog_pm` (figure 9). All tools that can potentially be used for instantiating that operator type are classified as of type *compile*. During instantiation the process relies on the specifications given by the typing of tools and MASPs.

We distinguish three cases for instantiating operator types:

- To link an operator type to a tool. In general it is necessary to build an envelope around that tool [Gisi and Kaiser 1991].
- To share an operator type's instantiation. This is analogous to the sharing of objects. A prerequisite for sharing the instantiation of an operator type is that the operator types used in the different MASPs are compatible. This is the case if operator type descriptions are exactly the same, or if the operator type is defined in one MASP and imported by the other. To share an instantiation can either mean linking the operator type to a tool (if the operator type, whose instantiation is shared, is instantiated by a tool) or linking the operator type to an already existing IMASP (if the operator type, whose instantiation is shared, is instantiated by an IMASP).
- To link an operator type to a MASP of the corresponding type. This possibility for instantiating an operator type is the most sophisticated one. It consists of linking an operator type to a MASP of that type. If the linked MASP is not instantiated the instantiation is continued by instantiating the linked MASP.

4 Software Process Enaction and the MASP Interpreter

When a MASP is instantiated, which gives an instantiated MASP (IMASP), enough information has been gathered to *enact* software processes. The information related to the control is already present at the level of the MASP, the information about the context of execution is given by the IMASP. In the MASP approach, enaction is an *interpretation* of the MASP using the IMASP as a knowledge base in a context local to an *assisted software process* (ASP). The interpreter of a MASP is a process that can be thought of as an "expert server":

- *expert* because this process will perform some forward and backward reasoning upon the activity of other processes being under its control,
- *server* because it will interpret many assisted software processes (ASPs) that are connected to the IMASP.

The overall architecture of MASP interpretation is depicted in figure 12.

The ASPs, where the real work is done, are "clients" to these servers, i.e the MASP interpreters. Each different ASP which is a client of a particular MASP interpreter will be a different *dynamic context* of execution from the same *static context* defined by the instantiated MASP. As a client, an ASP will hence have a list of possible actions to execute. Its actions will be controlled by pre- and post-conditions of operators, orderings, rules, and characteristics. Some of its actions might also trigger system initiatives.

The MASP interpreter also coordinates the flow of dynamic information between the ASPs so that one ASP can take an action as a result of a different ASP activity. For example, if some objects are shared between different MASP interpreters, they all will be informed of a change in the status of these objects.

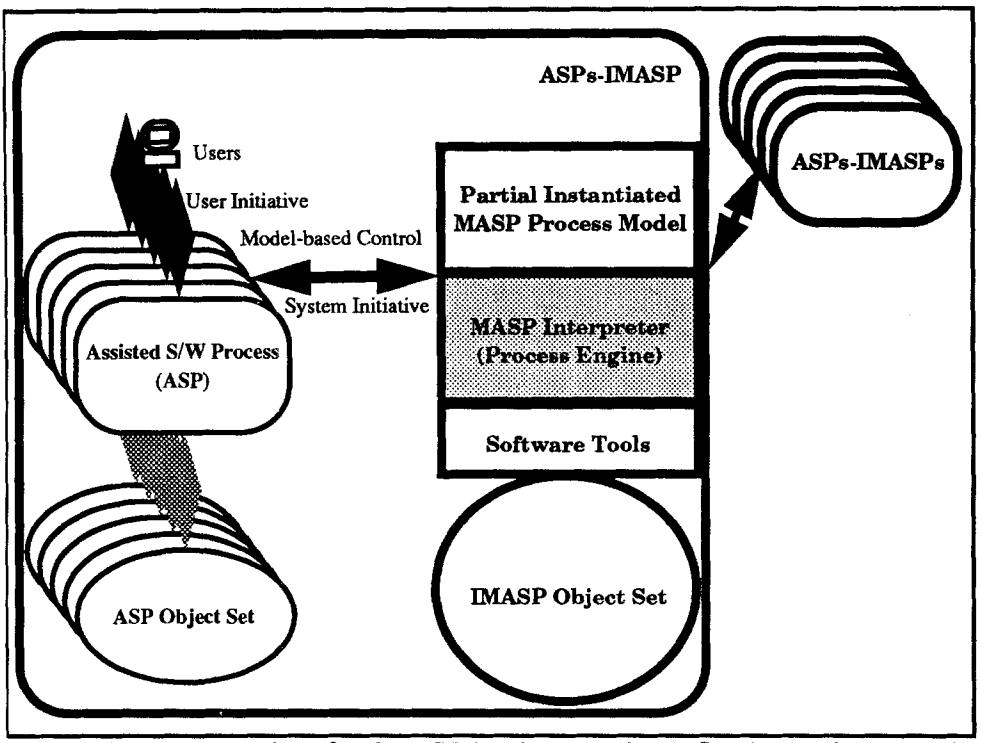

Fig. 12. Interpretation of an IMASP (static context) by ASPs (dynamic contexts)

Obviously, in the life of a MASP interpretation, the interpreter might need to instantiate an operator type by a MASP that was not needed and therefore not defined at the beginning of the interpretation. Therefore a cascade of MASP instantiations may occur.

At a given time, there are several ASPs being enacted. Each ASP carries out a software process step. Some of these are sub-processes created by expansion (i.e. binding and instantiation of an operator by a MASP) from a higher level software process (figure 13).

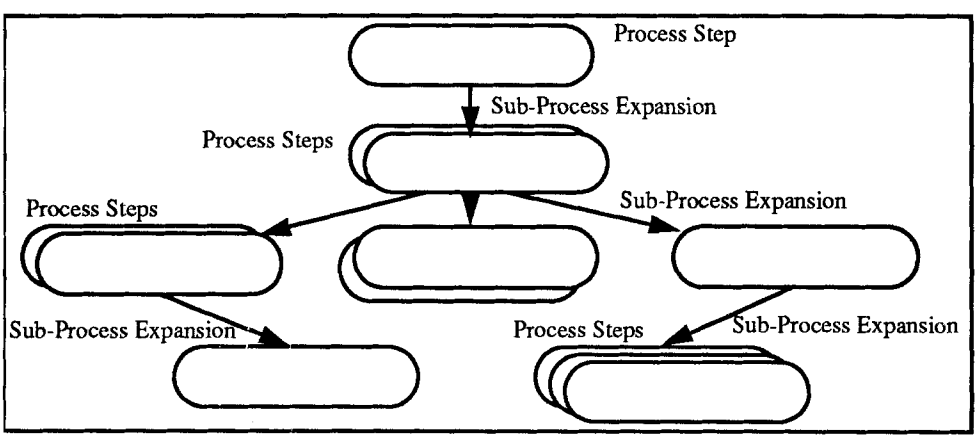

Fig. 13. ASP hierarchy of software process steps under enaction

5 The ALF Meta-Environment Architecture

In the previous sections we have described the formalism and the conceptual approach for defining, instantiating and enacting software process models in the ALF meta-CASE environment. This meta-environment, called *the ALF System*, provides the set of tools and services for generating process-centred software environments that can support the enaction of software process models in live projects. In this section the architecture of the ALF System is outlined.

Figure 14 sketches the architecture of the ALF System. Its main architectural component, the MASP Interpreter, will be briefly described hereafter.

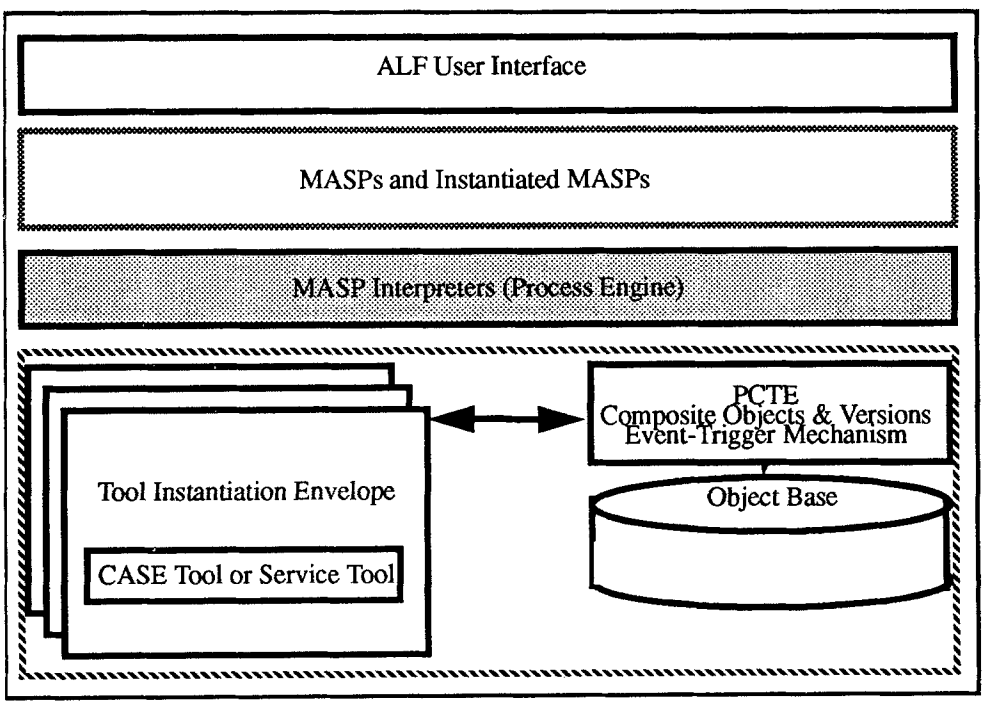

Fig. 14. The ALF System architecture

The MASP interpreter operates as a system that monitors the work done by the software developers and that takes initiatives whenever necessary. It provides assistance and guidance to software developers. The MASP Interpreter provides the most characteristic functionalities of the ALF System. These functionalities are obtained, amongst other thinks, through interpretation of software process models described by MASPs.

The MASP Interpreter has been implemented using the ALF-Rete Expert System Generator, a production system based on the Rete matching algorithm. It supports the integration of knowledge represented by rules with existing applications algorithmicly programmed, providing data sharing between the procedural and the heuristic parts of an

application. ALF-Rete works together with the PCTE Object Management System using an integration mechanism that has been designed to fulfil this function and that is based on the PCTE trigger mechanism [Oquendo et al. 1990]. ALF-Rete supports production rules in the form of *if condition then action* with forward chaining. These rules can be grouped into packets. Enhancements such as backward chaining, composite structures, quantifiers, and a mechanism for dynamic activation of rule packets with shared memory have also been introduced in order to support the MASP interpretation features. All these features together with the PCTE concurrence, synchronization and communication mechanisms have been used extensively.

The conjunction of the MASP Interpreter together with guidance and explanation, observation, history generation and feedback provides the basis for piloting the software process according to the MASP concept.

From an operational point of view, the MASP Interpreter communicates with the User Interface and the PCTE Object Management System (OMS). The MASP Interpreter receives messages from the User Interface containing user requests and sends messages to the User Interface with display requests. Both the User Interface and the MASP Interpreter are PCTE processes and, therefore, the PCTE interprocess communication facilities are used for message passing. The MASP Interpreter invokes PCTE OMS operations in order to request services from PCTE and to invoke CASE tools (that are stored as objects in the PCTE OMS' object base).

The foundation layer of the ALF System is PCTE[1] [Campbell 1988, Minot et al. 1988]. PCTE has emerged over the last few years as an Open Repository to serve as the basis for project support environments. The ALF System is based on the Emeraude implementation of PCTE 1.5 enhanced with new features such as composite objects, versions and triggers. The ALF System is currently running on a network of SUN workstations.

6 Conclusion and Future Work

In this paper, we have presented a representation formalism for describing computer-assisted software process models. A description of a generic software process model written in this formalism is called a MASP (Model for Assisted Software Process). Then we have presented the instantiation of MASPs and the enaction of assisted software processes. Finally we have sketched the architecture of the ALF meta-CASE environment that has been implemented to support the MASP concept.

In summary, a MASP (i.e. a generic process model) describes a class of IMASPs (i.e. project-specific software process models derived by instantiation) and each IMASP represents a class of run-time behaviours of software processes (ASPs). The ALF System provides a set of tools and services for editing, analysing, instantiating and enacting MASPs. The generation of process-centred CASE environments is supported by the instantiation of MASPs by MASPs or CASE tools in the ALF System.

---

1. European Computer Manufacturers Association (ECMA) Standard 149.

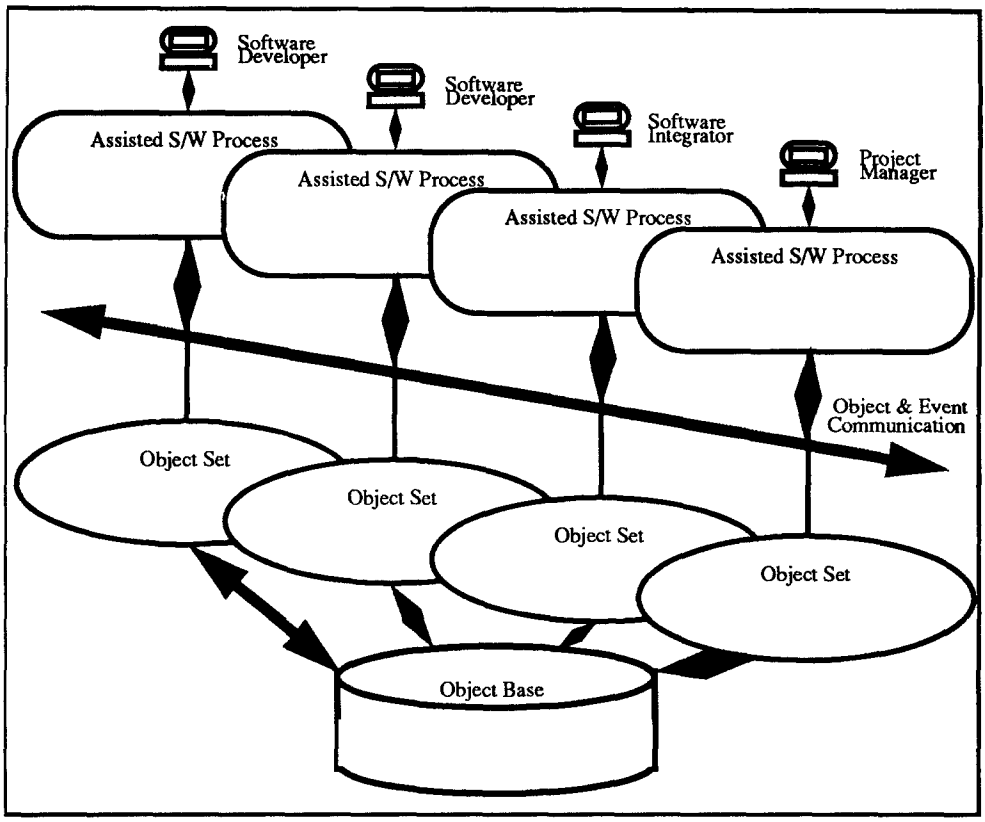

Fig. 15. The ALF System from the user's point of view

Using the ALF System, software house or large organisation's methods group can generate its specific project support environments in an Open CASE Environment Framework basis.

Figure 15 shows how does the ALF System look like to software engineers. What ALF means to software developers is a much more helpful development environment. Some of this help comes from the application of rules that constrain the developer to follow the procedures and methods of his organisation and to use the techniques chosen by that organisation for the different tasks he has to carry out. Although this may seem to some people like too much control, it will ease the developer's work by helping him to avoid forgetting the right way to do certain things and ignoring which technique (tool) to use for a particular stage of the development.

Yet more help comes from being able to get guidance on how best to proceed, explanation on what has been happening and why, and access to the ever-increasing historical information that can be used to make improved forecasts of what is likely to occur in the future. Just being able to use those tools which are appropriate to his work will be a great help to the normal software developer who is becoming snowed under by the variety of tools he can find on his workstation.

Formal descriptions of software processes as opposed to narrative or informal descriptions, have been adopted in ALF first, to improve quality and productivity through automatic interpretation of processes and data and second, to raise the level of abstraction in describing and manipulating such process in order to accommodate as many users as possible with minimum of training, particularly for new users who have to use a method or the system for the first time.

In addition, the idea of integrating such formal notations with expert system techniques whereby knowledge about the software process is stored into a knowledge base, renders the ALF System an expert advisor on a continuously increasing number of domains and on such issues as method or data applicability to a particular situation (i.e. optimising usage) and interpretation of the outcomes of such an activity; the provision of explanations and guidance on the use of a method or a tool; the use and interpretation of performance data on the ALF users, the process, or the system itself and on the reuse of components or systems already available in the system. Thus, intelligent "trainers" are embedded in the software processes themselves and can be activated by the user or at the system's initiative and as required within a particular working session.

Finally, a further contribution to productivity and increased software quality may be achieved through ALF by allowing the PCTE OMS' object base to act as a repository of models of processes or MASPs. Thus MASP designers can capture the characteristics of processes which are desirable and as these are reflected in an organisation's experience, and store them in ALF to be further used by individual users for different purposes. In other words, ALF provides the means for a possible wider acceptance of the standards, knowledge or data developed over a number of years. The benefits derived through this are that end users are the recipients of the same knowledge captured into the MASP but, also, the same users are guided by the system on the proper use/reuse and interpretation of the required knowledge or facts. Certainly, this is more advantageous to new users of a method or tool who not only need to learn how to apply it but also be aware of the consequences, of their actions.

Different aspects of existing design and development methods and life-cycle models have been modelled using MASPs, including technical as well managerial ones. Examples of models that have been described are VDM, JSD, HOOD, C Programming, and Configuration Management. Up to now the MASP concept seems to provide a powerful and flexible formalism for describing a wide range of process models.

Future work is mainly concerned with experimentation of the ALF meta-CASE environment for generating large-scale process-centred CASE environments for modelling and enacting real large-scale software processes and with experimentation of these environments in real work contexts to give feedback on how this novel technology works in practice. Psychological and sociological aspects are also important issues to be studied in order to assess the ALF System.

Acknowledgements

The authors acknowledge the contribution to the work presented in this paper from all of the members of the ALF project. The partners involved in the ALF project are: GIE Emeraude (Bull, Eurosoft, Syseca) (France), CSC NV/SA (Belgium), Computer Technologies Co.-CTC (Greece), Grupo de Mecánica del Vuelo, S.A. (Spain), International Computers Limited (United Kingdom), University of Nancy-CRIN (France), University of Dortmund-Informatik X (Germany), Cerilor (France), Catholic University of Louvain (Belgium) and University of Dijon-CRID (France). This work is partially sponsored by the Commission of the European Communities under the ESPRIT programme (Project Ref. No. 1520).

References

[Arbaoui and Oquendo 1991] Arbaoui, S. and Oquendo, F., "Où en est la modélisation du processus de production du logiciel?", *Proceedings of the 4th International Conference on Software Engineering and its Applications*, Toulouse, France, December 1991 (in French).

[Benali et al. 1989] Benali, K., Boudjlida, N., Charoy, F., Derniame, J-C., Godart, C., Griffiths, Ph., Gruhn, V., Jamart, Ph., Legait, A., Oldfield, D. E., Oquendo, F., "Presentation of the ALF Project", *Proceedings of the International Conference on System Development Environments and Factories*, Berlin, May 1989.

[Boehm 1986] Boehm, B., "A Spiral Model of Software Development and Enhancement", *ACM Software Engineering Notes*, Vol. 11, August 1986.

[Campbell 1988] Campbell, I.,"Portable Common Tool Environment", *Computer Standards and Interfaces*, No. 8, North-Holland, 1988.

[Campbell and Habermann 1974] Campbell, R. and Habermann, A., "The Specification of Process Synchronisation by Path Expressions", *Lecture Notes in Computer Science*, Springer, Vol. 16, 1974.

[Derniame et al. 1991] Derniame, J-C. et al., "Reference Manual for the MASP Definition Language", *Technical Report*, ALF Project, CEC ESPRIT Research Programme, July 1991.

[Dowson 1991] Dowson, M., "Why is Process Important?", *Proceedings of the 1st International Conference on the Software Process*, Redondo Beach, USA, October 1991.

[Gisi and Kaiser 1991] Gisi, M. A. and Kaiser, G. E., "Extending a Tool Integration Language", *Proceedings of the 1st International Conference on the Software Process*, Redondo Beach, USA, October 1991.

[Griffiths et al. 1989] Griffiths, Ph., Legait, A., Menes, M., Oldfield, D., Oquendo, F., "ALF: Its Process Model and its Implementation on PCTE", *Proceedings of the International Software Engineering Environment Conference*, Durham, England, April 1989. Published as *Software Engineering Environments -- Research and Practice*, K. H. Bennett (ed.), Ellis Horwood Books in Information Technology, England, 1989.

[Kaiser and Feiler 1987] Kaiser, G. E. and Feiler, P. H., "An Architecture for Intelligent Assistance in Software Development", *Proceedings of 9th International Conference on Software Engineering*, Monterey, April 1987.

[Katayama 1989] Katayama, T., "A Hierarchical and Functional Software Process Description and its Enaction", *Proceedings of the 11th International Conference on Software Engineering*, Pittsburgh, May 1989.

[Lehman 1987] Lehman, M. M., "Process Models, Process Programs, Programming Support", *Proceedings of 9th International Conference on Software Engineering*, Monterey, April 1987.

[Minot et al. 1988] Minot, R., Gallo, F., Boudier, G., Oquendo, F., Thomas, I., "The Object Management System of PCTE and PCTE+", *Proceedings of the IEE Colloquium on Standard Interfaces for Software Tools*, London, September 1988.

[Oquendo et al. 1989a] Oquendo, F., Gallo, F., Minot, R., Thomas, I., "Modeling Composite Objects in a Software Engineering Object Management System", *Proceedings of the 3rd International Workshop on Computer-Aided Software Engineering*, London, July 1989.

[Oquendo et al. 1989b] Oquendo, F., Berrada, K., Gallo, F., Minot, R., Thomas, I., "Version Management in the PACT Integrated Software Engineering Environment", *Proceedings of the 2nd European Software Engineering Conference*, Warwick, September 1989.

[Oquendo 1990] Oquendo, F., "Building Object and Process-centered Software Environments on the PCTE Public Tool Interface", *Proceedings of the 6th International Software Process Workshop*, Hokkaido, Japan, October 1990.

[Oquendo et al. 1990] Oquendo, F., Tassart, G., Zucker, J-D., "Support for Software Tool Integration and Process-centered Software Engineering Environments", *Proceedings of the 3rd International Conference on Software Engineering and its Applications*, Toulouse, France, December 1990.

[Oquendo 1991] Oquendo, F., "Supporting Software Process Communication and Cooperation Through Object Sets", *Proceedings of the 7th International Software Process Workshop*, Yountville, USA, October 1991.

[Oquendo et al. 1991] Oquendo, F., Boudier, G., Gallo, F., Minot, R., Thomas, I., "The PCTE+'s OMS: A Distributed Software Engineering Database System for supporting Large-Scale Software Development Environments", *Proceedings of the 2nd International Symposium on Database Systems for Advanced Applications*, Tokyo, Japan, April 1991.

[Osterweil 1987] Osterweil, L., "Software Processes are Software Too", *Proceedings of the 9th International Conference on Software Engineering*, Monterey, March 1987.

[Perry 1987] Perry, D. E., "Software Interconnection Models", *Proceedings of the 9th International Conference on Software Engineering*, Monterey, March 1987.

[Royce 1970] Royce, W., "Managing the Development of Large Software Systems", *Proceedings of IEEE WESCON*, August 1970.

[Stenning 1987] Stenning, V., "On the Role of an Environment", *Proceedings of the 9th International Conference on Software Engineering*, Monterey, March 1987.

[Warboys 1989] Warboys, B., "The IPSE 2.5 Project: Process Modelling as the basis for a Support Environment", *Proceedings of the International Conference on System Development Environments and Factories*, Berlin, May 1989.