34ALF: A Framework for Building Process-Centered Software Engineering


References


8. References


1. The reader interested in more details on the ALF project and system may consult the following list of papers:
[13] exposes the ALF model and its rationale,
[15] is the reference manual for the MASP/DL,
[2][14] respectively shows how the various concepts of the model cooperate to achieve the ALF’s functionalities and role coverage,
[10][5] go in details about the system architecture,
[24] specifies part of the ISPW6 and 7 example,
[3][4] respectively shows the assessment framework and the results of the assessment of the project,
[17][18][19] contain preliminary ideas and results on the management of long-term nested transactions
nisms has a large impact on software process enactment. The COO model to
describe processes is a goal oriented model which is founded essentially on first
order logic theory: the advantages in regards to the ALF model are that the con-
cepts of the COO model are more homogeneous and more orthogonal than the
MASP concepts.

As we show in [19], the fact that modeling is goal oriented is of the highest
interest to support cooperation and one important contribution of COO is the defi-
nition of a nested long term transaction model based on a formal correctness crite-
rion which allows cooperative executions [17](we characterize a cooperative
execution as an execution in which transactions exchange intermediate results).

Another aspect that was also deepened in COO is the assistance provided to
human agents. Especially, a new planning technique has been developed which
allows the maintenance of the consistency of one plan in regards to agent initiatives
and the consistency of several interacting plans [10].

The COO language is syntactically a sub-language of the MASP language
(however, due to new concepts and/or new semantics, the expressive power of the
COO language is not inferior to the MASP language one).

7. Conclusion

This presentation has attempted to convey some of the most important design deci-
sions and results of the ALF project’s approach to software process modeling and
enactment.

The aim of the project was to build a configurable open framework for IPSEs
supporting expression of software process models and providing model-based
intelligent user assistance. We have succeeded in developing a structured and mul-
tiparadigm representation formalism of software process models, and an architec-
ture for building customized process-centered IPSEs by plugging compiled process
model descriptions into a predefined framework based on PCTE.

The feasibility of the approach was demonstrated with a 180 000 lines proto-
type and some experiments, for instance with the ISPW6 software process model.

Our current general goal is to make the benefit of formal and explicit software
process modeling available to the industrial marketplace. It requires both to deepen
some important questions about cooperation and assistance, and the conduct of
real-life experiments in various contexts. We hope the new projects in the continu-
ation of ALF will contribute to reach this goal.
In this framework, we tried to take into account the state-of-the-art in psychological experimentation with man-machine communication in the design and the implementation of the ALF system. We deliberately avoided over-automation of process-model enactment to keep the system humanly understandable. Indeed, the need and the benefits of a wide range of assistance functions have not to be proved, but the system’s initiatives should be tunable and in a certain way, they should not escape the user control. Nevertheless, today, we cannot state any confirmed result on the user acceptance of the ALF system.

6.4 Perspectives
Two main projects have been settled to evolve ALF from two different points of view: SCALE whose objective is to enhance the MASP technology and COO whose objective is to deepen some aspects of cooperation in the software process.

SCALE [27] is an ESPRIT project (No. 6334) whose objective is to develop and demonstrate:

- a set of process centered environments for the support of configuration management, system construction, change control, and system composition strategies;
- a set of product and process models for configuration management systems, system construction and change control; in particular, it will provide a process model for system composition based on large grain and fine grain reuse;

and assess:

- the benefits of the provided product and process models and process support environments in terms of fitness of purpose, user acceptability, quality (functionality, usability, reliability, performance);
- the advances necessary from a technological and human factor point of view in order to optimize the processes used by SCALE demonstrators.

SCALE is in the continuation of ALF in the sense that the starting point for process modeling support in SCALE is the MASP technology, more precisely the MASP formalism and the ALF system.

COO is a research project which has been settled in Nancy to deepen the problem of COOrdinating COOperating agents. In COO, the evolution of ALF is more fundamental than in SCALE: not only mechanisms to support software processes, but the software process description model itself has evolved and evolution of mecha-
Evaluation and Perspectives

this aim, of strong data typing and accurate support mechanisms are ways being explored.

6.2 The Layered Architecture for Process-Centered Environments
The layered architecture and the distinction between a fixed part and a variable part makes the view of a process-centered environment clearer and helps in identifying common functionalities of such environments. In the ALF project, the choice of PCTE as the underlying platform alleviated the development effort and helped in focusing on process-model related mechanisms. But, at the same time, it impacted the design of the process model and the MASP/DL. Nevertheless, we feel that the prominent industrial objective of the ALF project (provide PCTE with an initiative engine) has been reached, even though the current MINT implementation does not meet all the initial requirements. Further, the ALF experiment brought partial evolutions to PCTE. Triggering mechanisms and multi-valued attributes are among the extensions that were needed in the ALF project and that are achieved. However, from the object model perspective, the evolution toward a state-of-the-art (object-oriented flavor) object model seems costly.

Further, PCTE extension with long-term transactions needs more profound studies before being implemented. Indeed, classical rollback techniques cannot be applied to software development activities: that is, the result of an activity, such as edit, for example, cannot be removed just because something wrong happened. Software processes are long-term processes (longer than a session) that share persistent objects, and a model for the management of long-term transactions is needed. This model should allow negotiated roll-backs. Moreover, a clear strategy for treating multiple instances of events, i.e. a kind of event algebra is also needed as may be needed support for user-defined events. Finally, the semantics of the system’s initiatives is insufficiently defined from a transaction management perspective. Indeed, should a rule activation or a characteristics enforcement be considered as part of the transaction that caused the activation or the enforcement, or should it be considered as a distinct transaction? is still an open question whose answer may require the availability of mechanisms for managing nested transactions.

From another standpoint, the current prototype proves the validity of the approach toward process-centered environments. Nevertheless, the system needs severe performance improvements, as it needs appropriate tooling to support process-model design and verification.

6.3 Human Factors in Process Centered Environments
In the current implementation, the scope of the guidance facilities have been limited to can it be done and how to do it. What can be done and what if facilities have been designed but are not yet implemented. And how does it work is not designed nor implemented. Furthermore, for the moment, the explanation facility has been restricted to on-line predefined manuals on the various topics related to the ALF concepts and system.
 alloys: A Framework for Building Process-Centered Software Engineering

- a batch mode that statically creates IMASPs from instantiation files before enaction.

6. Evaluation and Perspectives

Besides the limits of the current implementation that have been quoted before, this section attempts to provide a brief evaluation of the project and some lessons we learned (more details can be found in [3][4]). Then, a brief overview of the two main projects which have been established as the continuation of ALF are given.

6.1 The Process Model and the Associated Language

Process modeling requires particular knowledge and skills: software engineering expertise, database and knowledge base modeling, logical formulation. The specification of a model intensively appeals to knowledge acquisition techniques, and as such it gives rise to many questions. The most important one is the validity of the specified model. It seems clear that an ill-defined model will deliver incorrect support and an incorrect process. From a theoretical point of view, the hybrid nature of the model made it hard to determine a unique formal basis (contributions to a formal semantic definition were made in [21]). As a first consequence, we tried to give the MASP/DL an operational semantics, and as a second consequence, formal properties, like consistency and completeness of a given MASP description, cannot be proved. In the framework, considering the difficulty of achieving formal validation of a process model, we feel that simulation techniques could be used as well.

In the post-ALF context, we envisage using these techniques to validate a compiled MASP before instantiating the MASP-based environment.

From another standpoint, process modeling should be a progressive and an incremental activity. Indeed, we feel that the completeness of a process model may not be a desired quality of a model. But, merging models or dynamically changing parts of a model may be more useful mechanisms under the condition that the result of the change remains valid. In this context, we are investigating an object-oriented approach to process model evolution, expecting benefits in modularity and reuse of defined models.

Finally, the modeling of creative tasks was revealing to be the most difficult part in the specification of a process model. A trivial solution would consist in modeling such an activity as an operator type that will be instantiated by a knowledge-based tool. This means that knowledge related to the creative tasks is included into the tools that support the tasks and not in the process model. Similarly, the extension of our approach to capture fine knowledge on the objects, requires introducing additional features to our model, and the contributions, toward
MINT by enacting successively the different inference loops needed. This is done by the Communication loop, the module in charge of receiving requests messages from both users and the trigger mechanism. This loop continuously reads messages from the message queue associated to a MINT and processes each one by enacting the corresponding inference loops. The Communication loop uses in addition two tables. The first one records all created ASPs, and their state, related to the IMASP enacted by the MINT. The other one is used to record the operators currently executing. Then, the MINT periodically scans this table to determine whether an operation ended or not.

Implemented functionalities in the prototype are the following:

- precondition checking, with the expression language fully implemented except meta-predicates,
- ordering constraints, but the implementation is not persistent,
- MASP rules firing,
- characteristic checking, only in the case where the expression is built with an event part,
- Can it be Done? (precondition and ordering check) and How to do It? (plan generation) assistance facilities.

The MINTs are produced by the MASP compiler that works in four steps. The first one takes the Object Model description as input and compiles the SDSs using the Emeraude SDS compiler. The second step produces the ISK representation of the MASP being compiled. The third creates the trigger definitions associated to the MASP and, finally, the last step translates the MASP into XIA rule packets which are compiled by the XIA compiler to produce an executable code.

5.4 Miscellaneous tools
Several other tools were implemented as part of the ALF prototype. These tools are devoted to MASP definition and instantiation.

The MASP Editor is the tool devoted to MASP/DL text manipulation. The MASP Editor is a syntax-directed editor built using the xEdis tool from Syseca.

The Inconsistency Tracker For MASPs detects possible inconsistencies in MASP descriptions.

The Instantiation Tool allows the creation of IMASPs from MASPs by filling the Object and Operator sets. This tool works in two modes:

- an interactive mode generally used during enaction to dynamically create IMASPs or instantiated operator (i.e. lazy instantiation),
Guidance is a tool specifically dedicated to assistance functionalities. It is used to get plans from the system.

All these tools are implemented on the basis of the ALF User Interface Management System (ALF UIMS), a specific mechanism built for the ALF system on top of OpenWindow 2.0. The ALF UIMS allows a clear distinction between the interface definition and its use by different applications. This mechanism can also be used by production tools added in an ALF-based environment and makes very easy the maintenance, the evolution and the reuse of interface components.

5.3 MINTs implementation
The MINTs, which are produced by the MASP compiler, are implemented on the basis of the architecture detailed in 4.4. They are built on top of XIA-C, an expert system generator that uses the RETE algorithm for rule compilation.

Since XIA allows structuring rules in packets, rule sets corresponding to a MINT are structured into several rules packets, each one corresponding to a particular functionality. Each rule packet is governed by a specific inference loop and can be enacted separately from the others. A particular request is then processed by a
The ALF prototype 25 standard ones, their implementations were rewritten to add the trigger mechanism features.

5.2 User interface and Session tools
The ALF prototype provides a set of tools allowing users to work in an ALF-based IPSE. These tools are enacted each time a new session (ASP) is created. The three main session tools are shown in Fig. 1.7.

The main one is the so called ASP window tool that allows the users to take initiatives using the ALF command line. It provides also session management facilities (Quit, Disconnect, Suspend), object base navigation and scanning (Change Object, List Object) and simple report about initiatives undertaken during the session.

System Report is a tool used by the system to report about all the events occurring in the IMASP on which the session is connected. However, event types are selected by the users: only messages concerning selected events are displayed in this window.
In addition, in the case of execution requests, the message sent to the MINT may contain parameters. Due to the same requirement of persistence, each execution request is represented in the ISK by an object of type OPCALL. Parameters of this request are then represented by links starting from the OPCALL object to the effective parameter.

4.7 Architecture Overview
To briefly conclude about the architecture of ALF-based IPSEs, Fig. 1.6 summarizes the different components depicted above:

The underlying framework is PCTE, which provides object management facilities, execution support and distribution. The PCTE Object Management System is used to store all objects used by processes, and to implement the Information System. This information system ensures persistence of processes in the environment by representing their states as OMS objects.

The kernel of such an ALF-based IPSE is a set of compiled MASP s, or MINT s. MINT s provide control and assistance in a distributed manner. Each one runs in a particular work context corresponding to an IMASP, accesses to objects in a defined workspace and invokes instantiated operators. Thanks to the trigger mechanism, MINTs provide control and reasoning on persistent objects and on the real process state.

Users work in the environment through session tools that allow them to dialogue with the system. Session tools process user commands by sending execution or assistance requests to MINTs, using a persistent way of communication.

5. The ALF prototype
A prototype of the ALF system was developed at the end of the project, and demonstrated several times on the basis of the ISPW6 example. This prototype implements the major part of the language presented in and is based on the architecture depicted in section 4. Although the prototype demonstrated the feasibility of a MASP based environment, it is not actually usable for real purposes due to performance and reliability reasons.

5.1 Using PCTE
The ALF prototype was developed for the SUN 3 architecture, on top of the Emeraude V12 PCTE implementation. This implementation provides all the features conforming to the PCTE 1.5 specification. However, Emeraude V12 was modified so as to implement the trigger mechanism. Although the PCTE interfaces are the
A set of active session tools correspond to an ASP: these tools are invoked each time a new session is created or an existing suspended one is resumed. They are shut down when the session is ended or suspended.

4.6 Internal communication

The different components of an ALF-based IPSE have to communicate. Two kinds of communication are basically used. The first one is a simple call using the PCTE interfaces. This is the one used for all communication with the Object Management System, particularly by all the tools appearing in the process management layer. Another kind of communication is used to communicate with MINTs. An important requirement here is that the transferred information should persist to a MINT server shutdown. The adopted solution is to use PCTE message queues, which are similar to Unix ones except that they are persistent and distributed: a PCTE message queue is a particular OMS object accessible through navigation.

Thus, as depicted in Fig. 1.5., two message queues are defined in the ISK:

- one is linked to IMASPs and used to send messages and requests to the corresponding MINT server. This is the queue used by the session tools and by the trigger mechanism,
- the other one is linked to ASPs and used by MINTs to send back answers to their clients.

![Fig. 1.5. Persistent communication through ISK objects](image-url)
takes them into account. This functionality is realized by a communication loop that is in charge of all the external communications. This loop receives requests or change notifications, calls the appropriate inference loop and, if needed, sends back an answer to the requester.

4.5 User Tools
Users work in an ALF-based IPSE through a set of tools called Session tools that provide facilities for:

- emitting commands corresponding to tool executions,
- being reported about the state of the process and of the work context,
- asking for assistance from the system,
- scanning and accessing the object base,
- managing sessions.
Architecture of ALF-based

service allows the definition and work up of triggers. A trigger is defined by the association of an event to an action. Then, each time an event is raised, the Trigger mechanism undertakes the corresponding action. Events are defined on PCTE-OMS types. Typical events are UPDATE, CREATE or DELETE. Typical actions are to inform the concerned compiled MASPs of the event occurrence.

4.4 Compiled MASPs architecture

Let us now detail the architecture of the compiled MASPs, or MINTs. A running MINT is a server working in a given work context (IMASP), whose clients are the different ASPs related to the IMASP. A MINT provides the different services related to process enaction: control on user initiatives, assistance, reaction to events and tool invocations.

A MINT is basically implemented as a rule-based system: when compiled, a MASP/DL program is translated into a set of rules packets that are used to check preconditions, ordering constraints and characteristics, to fire MASP-rules, and to build plans. As expressions in a MASP/DL program are written from the Object Model definition, expressions in a MINT are to be evaluated against the state of the workspace. This ability to evaluate expressions and to reason about persistent objects is the first main characteristic of the MINT architecture. The second one is its server aspect: it receives requests from ASPs and sends back answers. The MINT’s architecture is illustrated in Fig. 1.4.

4.4.1 Expression evaluation and reasoning on persistent objects

As said above, a MINT is basically a rule-based system that results from MASP compilation, and the MINT runs in a given IMASP, or work context. However, firing rules requires evaluating expressions against the state of the IMASP object set.

For performance reasons, a MINT doesn’t evaluate expressions in the real workspace, but it uses an internal memory that is an exact image of the workspace. When a MINT is started, it first reads the IMASP object set to initialize the internal memory. Then, each time a running tool changes the workspace state, this change is notified to the MINT thanks to the triggering mechanism. When receiving such a notification, the MINT updates its internal memory to maintain its consistency to the workspace.

The application of rule packets is governed by inference loops to ensure that each request is correctly processed. For example, when an execution request is received, the inference loop first activates the rule packet related to precondition checking, then the rule packet related to ordering, next the rule packet related to tool invocation, and finally the rule packet related to MASP-rules.

4.4.2 A MINT as a Server

As said above, a MINT acts as a server: it receives requests from the users and processes them. It also receives change notifications from the work context and
4.3.2 **IMASP creation: instantiation**

IMASP creation is the operation that creates a work context (an IMASP) from a compiled MASP. This is performed by a tool called Instantiation Tool, working on the right hand-side of the schema depicted in Fig. 1.3.

As defined above, a work context is composed of two parts: an object set that contains all objects in the workspace, and an operator set representing the instantiation of each MASP operator type to an effective tool. Thus, the Instantiation tool:

1. creates an IMASP object in the ISK, and links it to the corresponding MASP and MINT objects,
2. builds an Object set by creating an OPSET object and a link to each object to be added,
3. builds an Operator set by linking OPERATOR_TYPE objects to PCTE static context (SCTX). An operator type may be instantiated by a PCTE tool, a Unix tool or a MASP, and the associated static context is respectively the tool itself, a script encapsulating the call to the tool or a particular tool called CALL_MASP.

4.3.3 **Operation enactment**

Given an operator type, some parameters and a work context, operation enactment consists in querying the ISK to find the correct executable program to be started, and starts it (white parts of the schema in Fig. 1.3.). In some cases, no static context can be found because the operator type is not yet instantiated. Then, the Instantiation tool is called. In other cases, the operation to be enacted is a sub-process, i.e. the operator type is instantiated by a MASP. The called static context is then the CALLMASP tool, and it proceeds as follow:

1. choose a work context (IMASP), or create a new one, if needed, by calling the instantiation tool,
2. choose an existing, but suspended, session (ASP) or create a new one, if needed, by creating a new ASP object in the ISK (see figure) and link it to the corresponding IMASP object,
3. if needed, start a MINT,
4. start the session tools.

4.3.4 **Event detection**

As MASP/DL allows the use of events in expressions, the process management layer is in charge of detecting such events and informing the different compiled MASPs. This is done by a specific service called the Trigger Mechanism. This
Architecture of ALF-based

Defined operations falls into two categories:

1. process model work up: MASP creation, IMASP creation,
2. enaction: operation enaction, ASP creation and event detection.

4.3.1 MASP Creation

MASP creation has a MASP description as its input and produces an ISK representation and an executable program called a MINT. This is performed by the MASP compiler who (see Fig. 1.3., left hand-side):

1. creates a MASP object. This object is a very complex composite entity (not detailed at all in Fig. 1.3.).
2. compiles the MASP/DL specification into a PCTE Static context\(^1\) (SCTX) and creates a MINT Object;
3. links the MASP object to its corresponding MINT object.

---

1. A static context is an OMS object of type sys-sctx, an object type which is subtype of file. The contents of the static context are an executable or interpretable program.
4.2.2 **Link type**
A link is a unidirectional association between one origin object and one destination object. All links are instances of a link type defined by:

- a name,
- a cardinality defining if either one or many links of this type can originate at a single object,
- a category (composition, reference, system defined)
- a set of destination object types,
- a set of key attribute types (integer or strings). The values of key attributes allow to distinguish between links of the same type,
- a set of non key attribute types.

4.2.3 **Relationship type**
A pair of inverse links defines a relationship.

4.2.4 **Schema definition set**
PCTE OMS does not provide a single global schema for the whole object base. Types are organized into consistent and self-contained subsets known as Schema Definition Sets. However, PCTE provides for the sharing of common OMS data through two different SDSs. Sharing definitions among SDSs is achieved through the import mechanism. Using this primitive, a definition can be explicitly imported from one SDSs to another SDSs. Two SDSs with the same definition can see the same instances.

4.2.5 **Working Schema**
A process executes in the context of a working schema. A working schema is an ordered set of SDSs. When an ambiguity occurs, the definition which applies is the definition in the first SDS.

4.3 **The Process Management Layer**
The Process Management layer provides the set of basic operations to work up and to enact process models. As said above, these operations have persistent results: the global instantaneous state of an ALF-based IPSE is stored in an information system called ISK. The ISK is a set of predefined PCTE SDS partially depicted in Fig. 1.3. and is used as the Working Schema for each Process Management operation: these operations use that view to access, create, modify or delete objects, links and attributes in the object base.
The PCTE interfaces specify the bottom layer of the fixed part of any ALF-based IPSE and a fundamental design choice was to manage every concept or artifact as persistent PCTE objects. Especially, predefined schemas are provided to structure and store MASPs, IMASPs, ASPs and their inter-relations. These schemas and the corresponding objects constitute what is called the Information System Kernel. The management of this information is directly under the responsibility of the fixed part which allows to edit, control, compile and instantiate MASPs, to create IMASPs, and to connect and disconnect users to IMASPs.

A compiled MASP is a piece of code which, when instantiated, can respond to user’s or system’s initiatives and can ensure control, guidance, explanation and report as specified in the corresponding MASP. As a response to an initiative, an active compiled MASP can ask the Process Management services to instantiate a compiled MASP or to execute a tool. Tools are pieces of code which effectively create and modify software products.

In fact, a compiled MASP encapsulates a part of the knowledge on the process which drives the current development. The complete process model is distributed between the different compiled MASPs. The following sections detail the most important functions of the different components of an ALF-based IPSE.

4.2 Underlying framework: PCTE

The PCTE interfaces provide three main mechanisms:

- Object Management System (OMS) mechanisms,
- Execution and communication mechanisms, and
- Distribution mechanisms

We limit our description to the OMS. It is based on the binary Entity-Relationship model. Object, link and relationship are the three main concepts.

4.2.1 Object type

An object type may be characterized as:

- a set of attribute types,
- a set of emanating link types,
- a parent object type: object types are organized as a hierarchy. The root of the hierarchy is the predefined object type object. The object type file is a direct subtype of object. An object of a type file has a content.
details in the architecture and the kinds of supports that were developed for this purpose.

4. Architecture of ALF-based IPSEs

4.1 General Overview
As shown in Fig. 1.2., an ALF-based IPSE is built by plugging compiled MASPs into a predefined framework for IPSEs: an ALF-based IPSE is made of a fixed part, which contains the Process Management Services and which does not vary from an ALF-based IPSE to another, and of a variable part which contains a set of compiled MASPs and tools and which distinguishes one ALF-based IPSE from another.
The Software Process Description

- Sequences: e.g. edit(x) ; compile(x) means that the edit operator type should have been applied to an object type x before applying the operator type compile on the same object,

- Simultaneity: e.g. print(x) means that, at a given instant, several invocations can be made to the operator type print on the object x,

- Concurrency: e.g. read(x) || print(x) means that one can concurrently read and print the content of the object x,

- Grouping: e.g. (lex(x); parse(x)) means that lexical analysis and parsing of the content of the object x are performed as a compound action,

- Alternative: e.g. print(x) | prettyprint(x) : print or prettyprint can be invoked,

- Iteration: e.g. test(x) (1 TO 5) : the test operator type may be consecutively applied five times to the object x.

Moreover, a connection part of the ordering model enables the designation of the objects that are concerned by an ordering.

In the following example (inspired from [24][36]), the ordering named myordering applies to every change request.

Example

ORDERING_MODEL_IS
myordering: FOR ALL _cr IN change_request DO
create_change(_p,_cr); change_control(_cr) ||
estimate_cost(_cr) || generate_history(_cr).
END_ORDERING_MODEL

Using the MASP/DL and the provided structuring mechanisms, one can express structural aspects by means of the object model, control aspects, by means of characteristics, preconditions, postconditions and orderings, as well as policy aspects by means of rules and orderings. It is clear that a model description is purely static. The objective of the enactment mechanisms is to exploit the static description to drive a process accordingly. The following sections go in more
3.2.3 The rule model description language
The rule model description language is influenced by production rules. Indeed, the
right-hand-side part of a rule (THEN clause) specifies an action to be performed
when the rule becomes firable, and the left-hand-side part (IF clause) specifies
when the rule is evaluated (ON clause) and under what a condition the action is to
be performed (EVALUATE clause).

Example
IF ON EXIT OPERATOR compile(_x)
   EVALUATE successful_compilation(_x)
   EVALUATED TRUE
THEN link( _x, /lib/maths.l)

The ON clause may specify various predefined types of events, like READ, CREATE,
UPDATE, DELETE, LOCK objects, attributes or links.

3.2.4 The operator model definition language
The operator model definition language is influenced by abstract data types specifi-
cations. Each operator-type is specified giving:

- a signature that includes the types of the operator type’s parameters and their
  respective passing modes (IN, OUT, INOUT),

- a precondition and a postcondition expressed as logical formulae that respec-
tively specify the condition under which the operator type may be activated
and the condition that must hold at the end of its activation.

An additional feature indicates whether the operator may be processed by the
system alone, or it needs any user’s cooperation (the KIND clause). This informa-
tion is useful for the active role of the environment i.e. the environment may decide
on its own to activate a non-interactive operator.

Example
audit : (IN _p:project; OUT _r:report)
   PRECOND: audit_authorized(_p) = TRUE
   POSTCOND: audit(_p)=TRUE AND proj_to_rep(_p,_r,NO_KEY)
   KIND : INTERACTIVE

3.2.5 The ordering model
The ordering model is described as annotated path expressions where one can
express operator-types:
Logical expressions are built using conventional comparison predicates, logical connectors and quantifiers, and a set of predicate symbols that issue from a predicative view of the object model. The later set encompasses two classes of predicate symbols: the first class contains process model predicate symbols that correspond to a logical formulation of the object model. Process model predicates are intended to be evaluated against MASP descriptions under the Closed World Assumption[29] (CWA), i.e. against information that concerns the model itself and that is kept in the Information System Kernel (see section 4.). For example, the following formulae describes the predicative view of the object type c_program and its associated link-types src and obj.

\[
\text{IS\_AN\_OBJECT\_TYPE(c\_program)} \land \\
\text{IS\_A\_SUBTYPE(c\_program, program)} \land \\
\text{OBJECT\_ATTRIBUTES(c\_program, \{tested, version\})}.
\]

\[
\text{IS\_A\_LINK\_TYPE(src)} \land \text{LINKS(obj, c\_program, c\_source)} \land \\
\text{CATEGORY(obj, COMPOSITION)}.
\]

\[
\text{IS\_A\_LINK\_TYPE(obj)} \land \text{LINKS(obj, c\_program, c\_object)} \land \\
\text{CATEGORY(obj, COMPOSITION)}.
\]

The second class of predicates encompasses predicate symbols that apply to instances of the object model, and they are evaluated against the data that describe on-going processes, under the CWA too. These data are also kept in the Information System Kernel. In the preceding example, body_unit(_b) AND body_compiled(_b, TRUE) intuitively means that b is an instance of the object-type body_unit and that body_compiled is a boolean attribute of body_unit that must evaluate to TRUE. This can be more formally stated as follows:

\[
\text{IS\_AN\_OBJECT\_TYPE(body\_unit)} \\
\land \text{OBJECT\_ATTRIBUTE(body\_unit, _x)} \\
\land \text{IS\_ELEM(body\_compiled, _x)} \land \text{IS\_AN\_OBJECT(_b)} \\
\land \text{IS\_OF\_TYPE(body\_unit, _b)} \\
\land \text{OBJECT\_ATTRIBUTE\_VALUE(body\_compiled, _b, TRUE)}.
\]

\textbf{Note:} The MASP/DL enables associating a name to an expression and then reusing this name in another context, to avoid rewriting the same expression. The same feature is offered for the orderings (see below).
Finally, relationships can exist between MASP s thanks to the MASP structuring mechanism. Indeed, an operator-type may be refined into a MASP, with its proper object model, operator model, and so on. This mechanism encourages a disciplined approach to process modeling, and a software process model may be described as a hierarchy of MASP s.

3.2 Foundations and Expressive Power of the MASP/DL

The foundations of the MASP description language (MASP/DL) are induced by the underlying influences of the model. This section shows the impact of these influences on the MASP/DL design, and discusses the expressive power of the language.

3.2.1 The object-model description language

The object-model description language is PCTE’s data definition language. It permits the specification of the structure of the object types (attribute types which describe an object type) and the relationships between the object types (link types or pair of link-types). It also permits the re-use of object type and attribute type descriptions to support type importation and type extension.

Examples:

- Attribute type definitions
  
  version# : INTEGER := 0;
  tested : BOOLEAN := FALSE;

- Object type definition
  
  c_program_dir : SUBTYPE OF sys_dir;
  c_program : SUBTYPE OF program;

- Object type extension
  
  EXTEND c_program WITH
    ATTRIBUTE tested, version#;
    LINK src : COMPOSITION LINK TO c_source;
    obj : COMPOSITION LINK TO c_object;
  END c-program

3.2.2 Characteristics

Characteristics appear as logical expressions (EVALUATE clause) associated with an optional event-part (ON clause) which indicates when the expression must be evaluated.

Example

ON UPDATE OBJECT body_compiled OF body_unit
3. The Software Process Description Language

3.1 Structuring Mechanisms and Component Relationships

Within a MASP, the basic components are linked to each other thanks to a set of relationships and construction mechanisms. Object-types are described by a kind of aggregation mechanism where attribute-types are grouped together to describe object properties. Further, a variety of relationships among object-types are expressible (type importation, type extension, subtyping). Relationships among operator types and object types are expressed by using typed formulae as preconditions and postconditions, where a formula’s variables can range over the types in the object model. Similarly, relationships between the rule model, the object model and the operator model are de facto expressible since a rule may reference logical expressions and operator-types. Further, relationships between operator-types may be explicitly mentioned in the orderings. Indeed, ALF proposes classical control structures (sequence, iteration, ) to relate operators to each other, and, at the same time, to enable expressing constraints among them.

Fig. 1.1. A MASP-driven environment with three users
check with the inconsistency tracker if there exists some impossibilities to use M2 and NewM2 together (for instance, there is no problem if changes are only introduction of new types),

(3) instantiate NewM2 into NewIM2, declaring that NewIM2 shares all instances of IM2,

(4) change the link between IM1 and its operator IM2 to NewIM2.

2.7 Conclusion

We have defined a set of concepts to support the development of process model centered environments. Activities, projects and roles can be described using the same three basic concepts:

- Software Process Model Fragment (MASP).
- Work context (IMASP) as an instance of a model fragment.
- Tasks, i.e. processes (ASPs), cooperating in the same work context, and following the same prescriptions (rules, ordering and characteristics).

They rely on a multiparadigm approach to software process modeling and enaction allows to perform actions, to control operators applicability, to provide users with guidance and explanations, and to (partially) support roles cooperation.

The MASP/IMASP/ASP framework is interesting to incrementally build and dynamically evolve a structured set of work contexts. It provides basic mechanisms to build cooperative work:

- it is possible to create several ASPs with different parameters for the same IMASP: several users can work within the same work context;
- otherwise, several IMASPs can share all or a part of their work context: several users can cooperate through object instance sharing.

Fig. 1.1. depicts a typical state of a MASP driven environment. Three users, Jacques, Marcel, and Henri are connected to the same working context IMASP1, deriving from MASP1, through three work sessions ASP11, ASP12, and ASP13. One of the operator type of MASP1 is described by a sub-MASP MASP2. IMASP2 has been instantiated from MASP2. IMASP1 and IMASP2 share a part of their work context. One of the user, Henri, has invoked the sub-MASP MASP2, creating a session ASP21, linked to IMASP2 and son of ASP13. The next sections will propose a language and an architecture to implement these concepts.
- "information tools", which allows the system to report on its evolution to the user.

2.5 **Team working and roles cooperation**

Several agents can work together in the same work context. In this case, they can access the same objects, they can invoke the same operators and they are subject to the same prescriptions (and particularly the same objectives): they are said playing the same role. A MASP, hierarchically decomposed or not, can be viewed as the software process model of this role. In a development team, many roles are of concern, such as developers, designers, quality officers, managers, secretaries, project manager...Team working results from role cooperation. Each role may have specific objects, tools and prescriptions: at enactment time, work contexts which support these roles have to cooperate. To allow cooperation\[12\], work contexts need at least to share instances (as described in the previous point 2). This sharing is defined during instantiation. It relies on the PCTE concept of view: an IMASP A can share an instance of another B, if, and only if, the type of this instance is known in the MASPs of A and B (types can be imported).

The prescriptions imposed to such a shared instance are those defined in the two different MASPs. So, one role can prescribe constraints to another one. For instance, the MASP for the Design Reviewers' role can import the object type `design` (and an attribute `approved`) from the MASP for the Designers' role; then, a rule can be defined in Designer, to trigger an action (for instance, `modify-design`) when the attribute `approved` takes the value false.

Shared instances are submitted to the PCTE access rights and can be used in PCTE transactions. Obviously, although necessary, instance sharing with short term transactions is only the first step: to support concurrency between software processes we would need an additional service in the framework providing for long term transactions (cf 6.2)\[18]\[19].

2.6 **Process model evolution**

Basic existing mechanisms can be used to support process model evolution. Process model fragments and type importation allow a modular description. Such a description can be statically modified and/or reused, on one hand. On the other hand, instantiation parameters allow the customization of a predefined model to specific needs.

However, changes can also be introduced more dynamically. Let M1 be a model fragment, using another one M2 as operator type. They are supposed to be instantiated into IM1 and IM2 and currently active (at least one ASP is running on IM1 and another one on IM2). To modify M2, this sequence of operations could apply:

1. modify M2 and produce New M2 (or a new version of M2),
- a “command interpreter”, which controls and guides users working within the model driven work context.

In ALF, the term instantiation is restricted to the construction of the workspace and the toolset, constituting what is called an IMASP (“Instantiated MASP”). An IMASP is a purely static work context. In particular, formal parameters of MASPs are not linked to actual parameters at this stage. The production of command interpreters is considered separately in ALF, and will be explained later.

More precisely, operator types instantiation covers two different cases:

- binding actual development tools to operator types, such as a text editor, a diagram editor or an electronic mail tool; in this case operations, i.e. operator executions, are not further modeled;

- binding sub-IMASPs to operator types; it is during the instantiation phase that the effective structure of work environments is built: IMASPs are linked together by a father-child relationship that mirrors the relationship between MASPs and sub-MASPs.

Instantiation can occur either statically, before enactment, or dynamically, during enactment. Dynamic instantiation occurs when invoking an operator type that has not been instantiated. Dynamic instantiation is also called “lazy instantiation”[6]. Often, both types of instantiation are useful: at the beginning of the process, a part of the model is statically instantiated. But some operator types are not instantiated and further objects may be dynamically added afterwards. It allows taking into account what has happened during the beginning of the enactment to build incrementally user work contexts for the rest of the model.

The second step between MASP and enacting processes is the creation of work sessions. A work session corresponds to the connection of a user to a work context (IMASP); it is captured by the ASP (“Assisted Software Process”) concept. It is a model-driven work session, because user initiatives are submitted to the prescriptions defined into the MASP, the IMASP derives from. The binding of actual parameters to MASP formal parameters is made at this stage. The interface between the user and the MASP-driven environment is established through a set of customized “session tools” (they must be clearly distinguished from “development tools” such as editors, compilers, and so on):

- “action tools”, which allows the user to express his/her initiatives;

- “guidance tools”, which provides the assistance functionalities discussed in the introduction section;
At this stage, a MASP allows the modelling of all aspects of a single activity, done by a single user, inside a project, and supports it in an integrated way. This activity runs inside a work context, which is an instance of a MASP (we call it an IMASP). Operator typing and the instantiation mechanism allows to encapsulate tools in this work context. The model allows to express prescriptions. They are used for users’ control (preconditions, orderings, rules), for guidance (pre- and post-conditions with the help of an inference engine), for explanations (pre-post and/or rules), for process observation (rules) and system initiatives (rules). Structuration and role cooperation mechanisms, presented in the next paragraphs, will extend these capabilities to group of users, represented by active processes, and cooperating within the same work context or within shared work contexts.

2.3.2 Model structuring
Due to the complexity of software processes a hierarchical structuring of model fragments is an essential help for achieving understandable software process models. At the definition model level:

1. model fragments can be recursively structured: an operator type defined in a process model fragment, a MASP, can itself be a MASP.
2. object types can be imported from one fragment into another, allowing definitions reuse.

So, following 1), a complex process model is defined by a MASP hierarchy. At the enactment model level:

- 1) will induce a hierarchy between the work contexts, similar to the hierarchy between model fragments.
- 2) will allow sharing instances between work contexts, when their types are common in the corresponding model fragments.

2.4 From MASPs to Enacting Model-Driven Processes
Several steps and concepts are needed between static models of software processes, on the one hand, and enacting model driven processes, on the other hand. The first step, instantiation, is the process of transforming a MASP into an associated work context. Such a work context includes three components:

- a set of objects being instances of the object types in the model;
- a tool set, each tool implementing an operator whose type is specified in the model;
notion of role which is a set of activities potentially enactable by an agent. Playing a role, an agent can execute its activities, adopting its policies. He is helped in his work by an enactment engine, which interacts with one or several users and sometimes works alone. Actors work in work contexts. In a work context, an actor takes initiatives, applying tools on objects, following a policy, i.e. in a certain order, and following a set of rules to fulfill an objective. So, to model work contexts, we use Software Process Models fragments. In ALF, a SPM fragment is called a MASP, and is a 5-uple:

\[(O_m, O_{Pm}, R_m, O_{Rm}, C)\]

where:

- \(O_m\), is the object model, which describes all the data used in the fragment. It is a typed ERA data model (Entity Relationship Attributes): it is the PCTE model extended with multivalued attributes.

- \(O_{Pm}\) is a set of operator types which are abstractions of tools, with a profile, and pre/post conditions. When invoked to be executed, an operator type has to have been linked (statically or dynamically) to a concrete operator, which is said to be an instance of the operator type, and its prescriptions have to be verified. In particular its precondition has to be evaluated to true. So, an operator type describes a family of similar tools, a specific instance of the family being choosen in the work context. If one of its prescriptions is violated the operator cannot be immediately started and the system must react. An inference engine, using mainly pre- and post- conditions and knowledge about the current state, builds a plan to try to fulfill the violated prescription. If it succeeds, the operator is executed. If it fails, the user is requested. A postcondition is supposed to be true when the operator has been executed. Postconditions are not verified but are only used to build plans.

- \(R_m\) is a set of rules, of type event-condition-action, which express how to react to some predefined events. Rules are used to automatically start actions when a specific state of the process is reached.

- \(O_{Rm}\) is a set of ordering constraints represented by path expressions\cite{9}. They express how operator invocations can occur with respect to precedence rules, simultaneity constraints between operators and conditions on object instances.

- \(C\) is a “characteristics”, i.e. an expression which has to be true, used as an invariant and/or as an objective. We use first order logical expressions and event expressions. When a characteristics becomes false, then it must exist an instant later at which the characteristic becomes true again: the constraint violation has to be repaired. The “reparation” is done by the inference engine, alone or with the help of a human agent.
“We need to be discussing multiparadigm descriptions of the software process,... For the moment, there is little evidence that a single process description formalism will be adequate for all the range of different concerns in process description.”

Process modeling problem is very close to the external behavior specification of systems (VDM, Merise, SSADM, Statecharts, SREM, Petri nets). All these techniques which we would like to support have some common aspects:

- They use operations and sometimes abstractions of operations: we propose an operator model.
- Operations are acting on objects with or without relationships: we propose an object model.
- Actions are undertaken under the control of events or conditions: we propose a model of rules.
- Many of them use a way to constrain the operations execution (FSM, Petri-Nets, ): we propose a model for ordering.
- Some of them use logical constraints to characterize some particular states of the process and/or to express invariant properties: we propose the notion of characteristics.

The combination of these models would enable us to define a large number of existing methods. The consistency of such a description is obviously a major issue. In ALF, we propose to use an Inconsistency tracker as an help to reach consistency.

2.3 A Structured Approach

2.3.1 Definition of Model fragments

The ALF main objective being to provide process models for enacting them in a customizable SEE, we have to provide a definition model, an enactment one, and their relationships

The definition model relies on widely recognized concepts, that a process generates products by employing resources and following some policies, with the help of human or computerized agents.

A MASP (Model for Assisted Software Process) is a generic model allowing the description of complex software production processes. Instantiating such a model gives a production model specific to a project, a team or a specific organization [23]. It obviously has to be defined progressively and so to rely on model fragments composition.

The enactment model has essentially to do with instances and activities (instead of their models), and with agents’ initiatives. An agent can be abstracted by the
Models being not only useful for describing and understanding, ALF had to support process models enactment;

Projects being developed by teams, we had to provide for agents and roles definition and for their cooperation.

Process models being defined often for long term projects, we had to allow their progressive definition, and their evolution.

2.1 A Configurable Open framework

To be able to support existing methods (such as VDM, SSADM, HOOD) as well as methods customized per project and per user defined policies, the focus in defining a Software Engineering environment shifts from a "mechanist" point of view to another one more oriented towards users activities and goals. In classical environments, often modeled by a couple (objects, operators), in which objects are coarse grained (such as files) and operators are commands, the knowledge about the processes is either built-in in the tools (see most of the CASE tools), in the underlying mechanisms, or come from the users. In new environments, this knowledge about processes has to be made explicit.

Referring to the ECMA-NIST reference model terminology [35], most of existing frameworks [8][16], provide integration mechanisms for the platform level, for control, for presentation and structuration, and for data. However, current proposals are poor to support policies and rule expression. Likewise, to support the requirements expressed by R. Balzer in [1]: "involve People in Automated Process for social activities support, (see also [25]), measure Process and Process progress, predict process evolution, improve process, support process change, manage inconsistency and incompleteness", we need an additional layer.

Classical underlying services have to be completed by a language able to describe the progressive building of SEEs components and their customization until the per project level. Main interests of using a language instead of a set of services are the structuration facilities for programming in the large (modularity) and capabilities for process model components reuse. Obviously, it is more economic to rely on an existing framework providing the classical mechanisms. Due to its acceptance as an Ecma standard (and soon as an ISO one), and to the existence of implementations, ALF is defined on top of PCTE. However, its concepts are mainly independent of PCTE.

2.2 A Multiparadigm approach

The range of software process entities is very large: "Those things which models must be able to represent, either as primitives or as constructs, are generally agreed to include at least some of the following: actions, activities, agendas, agents, configurations, deliverables, events, messages, methods, obligations, permissions, pre-post conditions, rules, roles, triggers, types, versions, views."[33]. Therefore, we concluded as [32]:
The ALF Model

- Explain what is happening, and especially when the control function forbids the performance of a user’s initiative,

- Learn from the past through observations to provide feedbacks to the system and also help the users to learn how to use the system.

Initially viewed as a process-centered environment based on PCTE-Emeraude[31], the ALF project resulted in a platform that enables the definition and the instantiation of process-centered Software Engineering Environments [11][22][30][34]. Indeed, the ALF system encompasses support for describing process models and for instantiating an environment that conforms to a given process model. It also provides support for enacting process models in instantiated environments.

This presentation offers a general view of the project and critically evaluates the results. These results include:

- a model to define enactable software process models, called the ALF model, and an associated language, called the MASP/DL (Model for Assisted Software Process Description Language); the model and the language are described in sections 2. and 3., respectively,

- an architecture for process-centered environments, called the ALF architecture and described in section 4.,

- a prototype of the ALF system that conforms to the defined architecture and that implements the major parts of the MASP/DL: it is described in section 5.,

- and a mock-up of a process-centered environment built using the ALF system and interpreting the ISPW6/7 software process model. Experimenting the mock-up permitted an evaluation of the proposal: section 6. reports on the evaluation and discusses projects following up ALF.

2. The ALF Model

The ALF concepts for process modeling have been defined following six main considerations:

1. ALF had to provide a configurable open framework for SEE’s supporting expression of software process models and providing assistance to its users;

2. To be as generic as possible, we had to adopt a multiparadigm approach;

3. Process models being complex objects, we had to propose a structured model;
2ALF: A Framework for Building Process-Centered Software Engineering

The ALF project wanted to improve the 1987-1988 state-of-the-art strategies for building SEEs by introducing a higher layer in a SEE, which provides at least two functionalities. The first one concerns intelligent user assistance: as far as possible, provide the best of assistance to any type of user or any type of role. The second functionality is to provide facilities that permit the customization of environments.

ALF’s approach is to provide means to describe the process activities and to integrate mechanisms to exploit these descriptions into the hosting environment. The integration of a Software Process Model into an environment includes informing the environment about the activities to be carried out, declaring people who will carry out these activities, and defining policies that govern the activities (like conditions for activities to start or to end). These features require the provision of Process Modeling Concepts and Process Model Description Formalisms, together with accurate mechanisms to manage process model descriptions. They also require the definition and the implementation of Process Enaction Mechanisms to soundly drive software processes in accordance with described process models, i.e. mechanisms to interpret or to execute an existing process model. The following goals were defined:

(1) Control the user’s initiatives and forbid them if they do not conform to some prescription of the underlying defined process model.

(2) Make the system play an active role, i.e. enable it to take:
   - Constructive initiatives to perform activities on behalf of the user,
   - Corrective initiatives to try to enforce process consistency when some of the prescriptions are violated,

(3) Guide the environment’s users, providing the following facilities:
   - What to do next to inform the user what activities can be performed considering the current state of the on-going processes,
   - How to do “something”, that is provide a kind of path that leads to a goal, like producing an object of a given type,
   - How does “something” work to help in performing an activity,
   - Analyze the impact of an action, i.e. evaluate the consequences of an action, especially when the action would change something in the model or in the current results of on-going processes,
   - Report on what has been done and how the current state of the processes has been reached,
CHAPTER 1

ALF : A Framework for Building Process-Centered Software Engineering Environments

Gérome Canals, Nacer Boudjlida, Jean-Claude Derriame, Claude Godart, Jacques Lonchamp

CRIN Campus Scientifique Bat. LORIA BP 239 - F54 506 Vandoeuvre-lès-Nancy cedex

1. Introduction

The ALF Project (1987--1992) falls in the category of research projects which aim at providing computerized facilities to support software development activities. It is also intended to make an environment play an active role during software development. ALF applies Knowledge Based Systems and advanced Information Systems techniques to Software Engineering Environments (SEE). Its objectives were to progress on the ways of building customizable Software Engineering Environments and to move from a mechanist point of view of SEE to another one which is more oriented toward users’ activities and goals.

1. The project was partially sponsored by the Commission of the European Communities under the ESPRIT programs (Project 1520). Its consortium comprised industrial partners from France (GIE Emeraude: BULL, Syseca, Eurosoft), Spain (GMV), Great Britain (ICL), Greece (CTC), Belgium (CSC NV/SA) and Academic Partners from France (Nancy University and CRIDijon-Burgundy University), Germany (Dortmund-Informatik X University) and Belgium (Louvain-La-Neuve Catholics University).