ALEP (Advanced Language Engineering Platform): an open architecture for language engineering

Conference or Workshop Item

ALEP (Advanced Language Engineering Platform)

An Open Architecture for Language Engineering

N K Simpkins *

Cray Systems
151 rue des Muguets
L-2167 Luxembourg
neil@cray-systems.lu

ABSTRACT

This paper provides an overview of the Advanced Language Engineering Platform (ALEP) and focuses on aspects of ALEP which give the platform a high degree of openness. ALEP is an initiative of the Commission of the European Communities (CEC), to provide the natural language research and engineering community in Europe with an open, versatile and flexible general purpose development environment ([1, 3, 2]).

Natural Language Processing (NLP) related projects currently lack a widely available platform for the development of large scale, professionally designed linguistic resources and applications. As a consequence, researchers and system designers are forced to build the tools and development aids they need from scratch, before undertaking the implementation of what matters most to them; linguistic resources or applications. This situation constitutes a major bottleneck for any serious attempt to build a strong and effective European NLP industry.

ALEP is a generic formal and computational environment, which can be put at the disposal of Community and national R&D projects in relevant areas, thereby avoiding duplication of effort across research projects. In recent years the need for language engineering platforms and toolkits has been more generally recognized and several primary research centres and companies are also undertaking similar initiatives.

1 Application

The environment is intended to ease and speed up the transitions from research to laboratory prototype, and from prototype to marketable product. It is designed as a basic, low-cost, non-proprietary platform for a broad range of research and technology development activities, related to NLP. By making ALEP widely available, the CEC aims to promote cooperation between different research centres and to progress towards portability and re-use of research results.

Typical users of the ALEP environment will be either skilled researchers in computational linguistics or teams of researchers and application designers, who will be provided with a software environment enabling them to produce linguistic descriptions of different languages, for a number of NLP or MT application domains.

*Thanks to Marius Groenendijk of Cray Systems and Paul Meylemans, Iain Urquhart, Roberto Cencioni & Nino Varili of the CEC for their contributions & support of the ALEP project.

By providing tools and basic lingware in an open platform ALEP is intended to:

- be used by those with their own third party tools who want to integrate them into a wider processing application.
- act as 'starter kit' of tools and resources for those wanting to enter the field (starting from scratch is very expensive).

ALEP is an 'open' environment in the sense that it is an appropriate host for projects with differing requirements in computational linguistics and application prototyping. ALEP may be used with third party tools, linguistic formalisms and lingware, or with the instances of these which are supplied with ALEP.

The ALEP initiative also encompasses a number of user oriented actions, such as an ALEP User Group (AUG), newsletters, training courses and ALEP workshops.

2 Development of the Platform

Various aspects of the ALEP system were specified by three parallel design studies:

- The overall environment itself was designed by a software environment design study, performed by a consortium of IAI, CAP Gemini and SNI [4].
- A standard formalism was designed by SRI-CRC as part of a rule formalism and virtual machine design study [5].
- The Text Handling (TH) subsystem was the specific area for a text handling design study carried out by SEMA [6].

After the design phase, the development of the ALEP platform was entrusted to a consortium headed by BIM with partners IAI, SEMA and SRI.

Development of ALEP was planned to consist of two cycles before final release and distribution:

- First development cycle (1992-1993)
  The first development cycle culminated in a assessment by selected user sites, after which ALEP proved to be sufficient for packaging and distribution to a number of projects. The first system distributed was known as ALEP-1.
- Second development cycle (1993-1994)
  The second development cycle sought to refine the environment in view of the user assessment of ALEP-1. The platform resulting from this cycle is known as ALEP-2.

- Dissemination phase

  After the final development cycle, ALEP-2 is being distributed with technical support services available to all users and with associated activities and events, such as training, user-group under the LRE-62101 project headed by *Cray Systems* [1] with partners IAI and GSI/ERLI.

3 Architecture

The layered architecture of ALEP and the relationship between these is depicted in Figure 1. The architecture of the ALEP development system is based on the following four layers:

Figure 1: Layered Architecture of ALEP

1. **Presentation layer**

   The presentation layer consists of interfaces to the various tools. This layer is based on MOTIF graphics and Emacs, which provides editing and display functions.

2. **Control layer**

   A controlling process routes requests for actions from the presentation layer to the kernel layer and returns results and messages produced by processing back to the user.

3. **Configurable layer (Tasks and Objects)**

   The kernel functions of the system are performed by Task Executors (TE) which execute tasks, expressed in the ALEP User Language AUL. A (configurable) number of tasks executors can be employed to support multitasking.

   Execution of a task can also make reference to stored data objects, such as the description of a grammar, which is to be compiled. Objects are also expressed in the AUL.

4. **Resource (Applications & Data) layer**

   The resource layer consists of data, stored in either files or databases, and applications which can be either simple executables or server applications.

[1] For details contact: *Cray Systems*, ALEP Support, 151 rue des Muguets, L-2167 Luxembourg, Email: alep-support@cray-systems.lu

The first three layers constitute the ALEP environment itself, which is distributed together with a number of standard applications and basic linguistic resources. New and additional applications and lingware can be integrated into this environment.

The configurable layer is most significant in terms of the support it offers users for structuring their work, and in terms of support for configuration and integration of new tools.

Both openness and customization are primarily supported by an environment user language, called the ALEP User Language (AUL). This is described later, but first the tools and characteristics of the platform are outlined.

4 Platform Characteristics

The design of the ALEP platform can be outlined in terms of a few high-level characteristics:

- **General purpose**

  The platform is suitable for development of a range of applications, both monolingual and multilingual. The system provides a framework for access to linguistic processing and text handling tools, resources and applications.

- **Open, Customizable, Configurable**

  The system's modular architecture allows for the extension of the toolbox and for the replacement of individual tools (Figure 2) [7]. Tools are themselves also easily configured in functionality.

Figure 2: Toolbox Extension in ALEP

- **Based on standards**

  The system builds on a number of standards (SGML, ISO 8859/1 & 8859/7 character sets, ISO Prolog draft and OSF/MOTIF).

- **Standard formalism**

  The system comes with an efficient, conservative linguistic formalism which is intended to be 'neutral' w.r.t. different linguistic schools, for example HPSG, LFG. This formalism is a 'lean' formalism for efficient execution but is sufficiently expressive for description of a wide range of language constructs ([9, 10, 11, 12]).

  The formalism has a number of layers and is itself designed to be extensible ([13, 14]).

- **Formalism Independent (environment)**

While the system comes with a standard formalism, the platform itself is formalism independent and can be used with other linguistic formalisms.

- **Basic Tools**

  The system comes with a number of basic tools for Text Handling, lingware development and debugging, and linguistic processing.

- **Graphical Interface**

  The system has a graphical customizable interface based on MOTIF and Emacs.

- **Multiuser**

  Linguistic and environment resources can be shared in public or private databases.

- **Support reuse and exchange**

  The clean 'neutral' declarative formalism allows linguistic knowledge to be imported and exported. ALEP also has import/export tools for packaging lingware. Individual applications and components of the system can also be used outside of ALEP.

- **Lingware resources**

  ALEP comes with demonstration lingware (German analysis, transfer and generation in English [15]) and ongoing projects are developing a methodology for producing large-scale grammars and resources in each of the nine community languages.

5 Tools & Applications

It is important to recognize the distinction between the ALEP environment itself, and specific formalism implementations, tools and applications for which it is a host:

- **Applications and Application Tools**

  ALEP is supplied with a variety of tools, one implementation of a typed feature formalism and demonstration lingware written in this formalism (compiler and Virtual Machine [5]). Other applications support document processing (Text Handling), grammar development and linguistic processing.

- **Environment Applications and Tools**

  ALEP is also supplied with a number of tools which provide functionalities related to use of the platform itself, such as the object editor, theory (type system) tool and query tool. These tools are neutral w.r.t. the applications and formalism employed.

The various tools and applications are briefly described in the next three sections.

5.1 Environment Tools

The system is supplied with a number of 'environment' (AUL) tools to support in using the system itself. These tools include:

- **Object Editor (OE)**

  For browsing and editing object descriptions held in private and public (shared) databases (Figure 3). The OE also supports import and export of objects together with the resources they describe.

Figure 3: GUI of Object Editor

- **User Language (Query and Batch) Tool**

  For writing, storing, compiling and executing AUL queries and launching batches of these. A query will typically be a conjunction of calls to tasks at the kernel layer.

- **Console Tool**

  A command line query interface, the ALEP Console Tool provides a flexible interactive interface to the kernel layer for querying and overall control of the configuration in use (applications).

- **Theory Tool (ALEP-2)**

  For editing, compiling and testing a new AUL type system (for tasks & objects) which can then be used to specify a new configurable layer.

- **Help Tool**

  A graphical online browsing tool, giving access to the manuals, tutorials, lingware reports. These documents are produced using Texinfo which allows a single source document to be processed into a structured online set of 'nodes' for browsing, searching etc and a to be processed by TEX to produce a high quality printed document.

5.2 Applications

Applications within the supplied system currently include:

- **Compiler**

  The linguistic formalism and AUL are both compiled by executables called from the configurable

layer. The compiler converts formalism and user language expressions (declarations, macros, queries, rules etc) into first-order (Prolog) terms, suitable for efficient execution.

The compilers are multipurpose applications within the environment, not only compiling lingware but also supporting functionalities such as expression verification, lexicon searching by expression, etc.

- **Virtual Machine**

  The virtual machine (VM) is implemented as a server application (optionally invoked but persistently running within the system). The VM implements the standard ALEP linguistic formalism operations (TLM, analysis, transfer, synthesis etc), applying grammar and lexica to the input.

- **Debugger**

  The standard debugger is a post-mortem based server application. The debugger is invoked to browse a log of events recorded during linguistic processing (analysis, transfer etc).

- **Text Handling**

  The text handling system is realized as a suite of executables which perform document markup and which break and reassemble texts for processing by the VM in user definable chunks.

5.3 Application Tools

ALEP comes with a variety of basic tools for text handling, linguistic development and processing. These tools act as frontends to the applications and provide access to these and the resources required to perform operations, through the configurable layer:

- **Grammar tools**

  For editing (with Emacs completion, syntactic verification) and incremental compilation and loading of lingware expressions (declarations, Phrase-Structure rules, transfer rules, TLM rules, TH declarations, abbreviation tables, S&R rules etc).

- **Lexicon tools**

  For editing, searching, sorting, compiling, loading & verifying lexicons.

- **Debugger**

  A post-mortem debugging provides an interface to a stored log of events which can be produced by linguistic processing (TLM, analysis, transfer etc)

- **Graphical Feature Structure Viewer**

  A powerful tool for handling the large structures commonly produced by linguistic processing (zoom, hide, filter, co-indexing etc) [16], Figure 4.

- **Processing tools** for:

  - Analysis and generation of documents (Text Handling),
  - Linguistic operations (analysis, transfer, synthesis etc),
  - Combined text handling and linguistic operations (text2text),
  - Saving of operation events (for use with the debugger).

The standard tools distributed with ALEP have a common 'look and feel'. This is because the tools all have a common basis: they are instances of the Generic Tool [2].

Figure 4: Xmfed Structure Display

The generic tool is described in more detail later, before being related to specific aspects of openness of the platform. First the components of the ALEP User Language are outlined.

6 Configurable Layer

The configurable layer is expressed in the ALEP User Language (AUL). This language was designed as the underlying user language for ALEP to be:

- a typed declarative attribute-value language, close to the linguistic formalism and other mainstream declarative formalisms,
- familiar to (linguistic) users of declarative unification-based formalisms,
- fully compilable into Prolog terms and thus efficiently executable using modern Prolog compilers.

The system is supplied with a standard instance of the configurable layer (a set of types, objects, tasks and bindings), but the configurable layer is fully user definable and extendable. Much of the functionality of ALEP itself is specified in AUL, allowing the system to be easily and extensively modified. ALEP also has tools for supporting the user in editing, extending, and compilation of the configurable layer.

The language has four basic components; declarations, objects, tasks and bindings, which are described in the following sections.

[2] Tools are not restricted to those based on the generic tool, for example, Xmfed (Figure 4) and an interactive tracing tool are distributed with ALEP which are not based on this model.

6.1 Declarations

AUL declarations define the types (or sorts), attributes, and macros which can be used in a specific instance (objects, tasks and bindings) of the AUL. Declarations, in common with those of the standard formalism, support strict type checking and compilation to first-order (Prolog) terms. In addition, a type contains a comment string used for documentary purposes.

Figure 5: User Language Declarations

In Figure 5 a simple example of type declarations for description of source grammars is given. An object (instance) of this type (`source_grammar`) must conform to the type declarations. This is ensured by the AUL compiler. The AUL is roughly as expressive as the standard 'lean' formalism [5] and supports type hierarchies, macros and defaults. The attribute '`decl_ref`' in the example object is described as having values which are of type `tnowv`, where the default value of `type` within such values is `lg_decls` (declarations).

The standard tools and taskware supplied with ALEP assume identification of an object by its `name`, `owner`, `workarea` and `version` attributes ('`NOWV`') but there are no further restrictions placed on the specification of an instance of the AUL.

6.2 Objects

Objects are structured descriptions of resources, tools, etc and are persistently stored in a clausal database (either private to the user or public).

The example object in Figure 6 illustrates an object of type `source_grammar` which conforms to the type declarations given in Figure 5.

It is not required that all parts of an object are instantiated, for example, the `tnowv` value of `decl_ref` is only partially specified. Objects are maintained using the Object Editor (Figure 3) which lists objects by their type together with their associated `NOWV`.

Objects have a privileged role within the ALEP environment in that operations are accessed and invoked by selection of an object, and according to the object's type. Under this object-action model, a user selects an object of a certain type in a tool's object list, and then selects an action to be performed. Only actions appropriate for the selected object type are made available. Selection of the object allows the appropriate information to be extracted and updated according to the requested action. Actions are carried out by tasks, which are described next.

Figure 6: User Language Object

6.3 Tasks

Tasks carry out actions in response to user queries received from the presentation layer. A task can make reference to objects and call upon other (sub-)tasks, for example:

- compile and load a grammar
- analyse a sentence and present the results
- create (start) a new tool

Tasks are stored in compiled form, and accessed by task executors (Figure 1), which execute tasks to perform the user's desires.

A task has two components:

1. A parameter part, which specifies the type of input object(s) upon which the task will act. This is commonly a partial description, where only those attributes required by the task are described, or a `NOWV` identification of an object stored in a database, which the task retrieves to act upon.
2. A 'constraint' part, which specifies the steps necessary to carry out the required action. This can make calls to other tasks, access the object databases in use and call the kernel functions of the system (prolog predicates).

An example task, of type `load_db_grammar`, is given in Figure 7. This task loads the compiled files of an object of type `source_grammar`. The task has a single parameter '`grammar`' of type `nowv` which identifies the grammar(s) to be loaded. In the first step the identifying attributes (`NOWV`) are extracted from the parameter, then a description of the `source_grammar` object

Figure 7: User Language Task

is generated (unification) as the value of a variable '`Obj`' (step 2). The third step illustrates a case of resolution against the object database (see later), which unifies the object description with an object instance stored in a database. The fourth step ensures that the grammar is compiled uptodate and, according to conventions held by the taskware for this grammar's compiler, determines the names of files that should be loaded. The last two steps load the grammar and generate messages to signal this to the user.

A task can be seen to be syntactic sugar for a Prolog clause, given that types can be compiled into first-order terms. The AUL compiler is used to produce the compiled Prolog form of tasks, ready for loading and execution by a Task Executor.

Within ALEP, tasks are executed exhaustively depth-first by a Prolog engine, so that for example, if there is more than one grammar in the databases which is identified by the given `NOWV`, then all such grammars will be loaded by the task (Figure 7).

6.4 Bindings

The specification of actions (tasks) appropriate for a specific object type is defined by one or more 'bindings'. Bindings relate tasks to objects, supporting the object-action model, and are stored in the user and public object databases.

Each binding establishes a relationship between an object type and a number of tasks. In addition a binding specifies the input parameters required to perform each task and how the output results are to be treated. Standard queries are provided for handling input and output parameters which, for example, generate input dialog boxes at the presentation layer for parameter input and assert object descriptions for results.

The example binding (Figure 8) belongs to a class of

Figure 8: User Language Binding

bindings `lg_loading:{}` (a type) and acts on (is available for) objects of type `source_grammar`. The bindings specifies two actions (`lgLoad` & `lgUnload`) which have a common input parameter (the identification `NOWV` of the object selected in the tool) and which each call a task, `lg_db_load` & `lg_db_unload` respectively. An action identifier such as `lgLoad` is translated by X resources into a string (for example '`Load`') which appears on the tool's `Actions` menu (Figure 9). When an action is selected by the user from the `Actions` menu the corresponding binding query is executed.

7 Generic Tool

The Generic Tool (GT) is an inherent part of the ALEP platform which can be used as the basic framework for specification of a tool.

The GT defines an interface and general set of functionalities common to all the standard 'tightly' integrated ALEP tools. Most of the tools supplied with ALEP are instances of the GT. The principle of a GT, not only makes available the basic programmatic interface and infrastructure for new and third party tools, it also ensures uniformity, such that all tools have a common generic functionality and specific functionalities are easily located by the user.

The GT includes a specification and implementation of:

- **MOTIF tool window**

  A motif part of the tool which has a number of standard pulldown menus with functionalities applicable to the components of the GT itself (`File` menu, `View` menu, `Help` menu) and for control within the ALEP architecture (`Solutions` control and `Abort` query button) [3].

  The MOTIF part also provides a scrolled window for messages and a scrolled list for objects which

[3] Details of these functionalities are not important for the description here, for more details refer to [17]

can be selected for actions in the tool.

- **Emacs part**
  A number of associated Emacs buffers for editing linguistic and AUL expressions, presenting results etc. The Emacs screen is shared across different tools, each tool has a number of buffers accessed through Emacs or using the tool's `File` menu. Emacs itself is provided with an ALEP mode which performs completion, insertion etc based on the type declarations of the AUL.

Figure 9: Instance of the Generic Tool

A specific instance of the GT is defined by specification of an AUL creation task. Such a task collects functionalities belonging to specific classes which will be made available, under an object-action model, within the tool (as options of the tool's `Actions` pulldown menu, Figure 9).

Figure 10: Tool Creation Task

The outline example tool creation task (Figure 10) illustrates how a tool creation task can be specified. The task accesses the user's databases for a set of parameters held in an object of type `my_tool_params` using the task `find_object` which issues messages about the object(s) found. It then calls the task `display_tool` which collects bindings of the specified classes (`lg_loading` etc) for presentation on the tool's `Actions` menu. The `Actions` menu is restricted to those functionalities defined for the object type selected in the tool from among those collected from all bindings. The `configuration` attribute specifies a configuration of the GT with an Object Set (`os`) and an Actions Menu (`am`). The `files` attribute specifies a list of files to be loaded into Emacs (accessible via the tool's `File` menu) and `objects` lists the objects to be initially listed in the tool's object set. The `appl_class` and `app_name` relate to the X resources which will be used for the tool.

8 Openness

ALEP is designed to be an open environment which can either be used as supplied (with the tools that have been outlined) or which can be configured or extended with new resources, tools and applications. The high degree of openness leads to an expectation that ALEP will have some longevity in its appeal for development and application prototyping. The platform is able to accept new developments and to be extended with new and replacement tools which were not foreseen for the initial system.

ALEP is open to extension in a wide variety of its components [7], some of these are outlined in the following sections.

8.1 Resources

The simplest level of integration concerns the addition of new instances of lingware and tools to ALEP. This simply requires the specification of new objects for assertion into the user's database.

For example, a new grammar can be created by defining a new object of type `source_grammar` (Figure 6). This object describes the files which make up the grammar together with a number of parameters (language, compiler etc) which are required for operations on the object. In addition, under the ALEP-2 object-action model, a specific action is associated with each object type. Currently this is defined, through the configurable layer, as creation of a specific tool, for example, of a lexicon editor tool.

Simple assertion of a new object instance also immediately makes all the functionalities supported for that type of object accessible (for a lexicon, for example, search, sort, compile, load for use in analysis).

In addition to resources, objects are also used to specify parameters for processing tools. For example, ALEP is supplied with different versions of processing algorithm for analysis and means to control the application of rules by specifying sub-grammars. A processing tool 'environment' object describes these parameters. A 'new' tool with a different set of parameters (and behaviour) can be created by simple assertion of a new environment object.

8.2 Tasks

Tasks are defined in the AUL as persistent parameterized procedures. A new task can be specified using the AUL and compiled in with the existing configurable layer using the Theory Tool.

A large proportion of the functionality of ALEP is pro-

vided by AUL tasks. This extends to tool creation and message processing, so that a new tool can be defined or messages processed into a chosen language.

8.3 Queries

The notions of 'querying' and their 'resolution' are built into ALEP. A query can be thought of as a dynamic task (constraint part), which is compiled just before execution. The AUL supports complex queries, which can be stored as batches and executed when desired using the User Language tool, or through the Console Tool.

Unlike tasks, queries are not compiled in with the configurable layer but can be dynamically written by users for specific purposes. A typical batch of queries will be a specialized set of task calls, for example a set of calls to the `load_db_grammar` task (Figure 7) could be defined to load a number of the user's grammars, identified by a list of grammar identifiers (`NOWV`).

8.4 Language

The AUL language (type system or sorts) can be replaced or extended. This means that for example the description of a grammar can be altered if additional attributes are required for new tasks.

8.5 Tools

The addition of new tools is supported at three different levels:

1. So called 'loose' integration [18], a task may call any (UNIX) executable. Applications such as the AUL compiler are an example of this.
2. Standard ALEP tools are created by an AUL tool creation task which invokes an instance of the Generic Tool. Such tasks collect the functionalities which are to be made available in the tool instance. The specific functionalities of a tool are defined by bindings which relate object type(s) to tasks in the configurable layer.

   A new instance of the Generic Tool can be specified either through addition of new bindings or bindings classes, or through a new tool creation task which collects a new set of bindings.
3. A new 'tightly' integrated tool can be added to the architecture. Such a tool can have a different/specialized MOTIF part. In such cases the Generic Tool serves as an example of the protocol between presentation layer and controller and provides the tool creation and Emacs interaction infrastructure required. The Graphical Query Tool [8] is an instance of such an additional tool.

The Generic Tool also provides the infrastructure required for communication with a new or replacement server application, which may be required to provide the processing functions for a new tool.

9 Future Development

ALEP-2.2 was released in December 1994. An updated version of this (ALEP-2.3 is to be released in June 1995 which has additional tools for debugging and object management. After this release the platform will ported to other platforms (Prologs and PC environment) and the architecture and standard taskware refined.

A number of extensions to the platform (formalism and tool set) have been, and are being, developed by other projects. These will be integrated in and distributed with future releases of the system. It is also intended that current and future development of the platform be driven by well specified user requirements originating from the ALEP User Group (AUG).

References

1. Meylemans P & Simpkins N, *Towards a Portable Platform for Language Research and Engineering*, Thirteenth International Conference on Artificial Intelligence, Expert Systems and Natural Language, Volume 3, pp161-170, EC2, 1993.
2. Simpkins N K, *An Open Architecture for Language Engineering*, Proceedings of the Language Engineering Convention/ Journees du Genie Linguistique, Paris 6-7th July 1994, EC2, 1994.
3. Meylemans P, *ALEP, an Advanced Language Engineering Platform*, CEC, February 1993, ELSNews, 2(1).
4. IAI, CAP Gemini & SNI, *ET6/2 Software Environment Study (Final Report)*, CEC, 1-2, 1991.
5. Alshawi H, Arnold D J, Backofen R, Carter D M, Lindop J, Netter K, Pulman S G, Tsujii J & Uszkoreit H, *Eurotra ET6/1: Rule Formalism and Virtual Machine Design Study (Final Report)*, CEC, 1991.
6. Devillers C, Burnard L D, Hockey S M & Gibeaux G, *Eurotra-6/3 Text Handling Design Study (Final Report)*, CEC, 1991.
7. Groenendijk M, *Integration into ALEP*, CEC, 1994.
8. Groenendijk M, *GQT User Guide*, Graphical Query Tool, Version 1.9, CEC, March 1994.
9. Uszkoreit, H & Erbach, G, *Linear Precedence Constraints in 'Lean Formalisms'*, CEC, December 1993.
10. Moshier A, *Preliminary Report on a Type System for ALEP*, CEC, March 1994.
11. Erbach G, Moshier A & Uszkoreit H, *Multiple Inheritence for ALEP*, CEC, 1994.
12. Simpkins N K & Groenendijk G, *Multiple Inheritence, ALEP user guide*, CEC, 1994.
13. Crouch, R, *A Prototype ALEP Constraint Solver*, SRI-CRC, February 1994.
14. Ruessink, H, *Manual to the RGR extensions for ALEP*, Preliminary Version, CEC, July 14 1994.
15. Theofilidis A, *ET9/1 Lingware Development*, Final Documentation, CEC, June 15 1994.
16. Groenendijk M, *Xmfed User Guide*, Graphical feature viewer, Version 5.3, CEC, June 1993.
17. Groenendijk M, *Environment Tools Guide*, Guide to the ALEP User Interface Tools, Version 1.0, CEC, September 1994.
18. BIM, *Distributed Architecture: Integrating User-Interface Tools*, ALEP System Documentation, Version 2.1, CEC, June 1994.