ALBERT: A Formal Language and Its Supporting Tools for Requirements Engineering

Eric Dubois

Facultés Universitaires Notre-Dame de la Paix, Institut d' Informatique, Rue Grandgagnage 21, B-5000 Namur, Belgium. Email: edu@info.fundp.ac.be.

1 Introduction

The use of formal specification languages has been proven useful in the development of large and complex software such as safety-critical software. Specification languages like Z, Larch and Statemate propose useful artefacts for modelling and reasoning on the behaviour of the desired *software*. For Requirements Engineering (hereafter RE), similar artefacts are needed but with the objective of capturing customers' descriptions related to a desired *system*.

Albert is (yet another) formal specification language proposed to support RE activities. The design of the language started around 1992 within the framework of the Esprit II project *Icarus*. The validation of the language constructs has been achieved through the handling of non trivial case studies (ranging from CIM to advanced telecommunication systems) performed by the members of the team who developed the language [1]. The language has been the subject of several technology transfer initiatives. In particular, it has been used by two industrial partners in the context of the development and the evolution of two large, distributed, software-intensive, heterogeneous systems (a video-on-demand application and a satellite-based telecommunication system) [2].

Besides the development of the language, we have also started the development of tools in 1995. This development is done within the framework of a 5 years project (25 man/year) where we have adopted an incremental strategy resulting in the delivery of a first version of the tool in December 1997 and and the delivery of second version of the tool in December 1999.

2 The Albert language

Albert organizes an RE specification around the agents identified in the usage environment, where an **agent** is an active entity that can perform or suffer *actions* that change or maintain the *state* of knowledge about the external world and/or the states of other agents. Actions are performed by agents to discharge contractual obligations expressed in terms of *local constraints*, applicable to the agents itself, and *cooperation constraints*, that apply to the interactions between agents. Hereafter, we illustrate the application of the language on the specification of a fragment of a Video on Demand (Vod) system, i.e. a system consisting of a set top box located in the home of a customer, connected to the television set of the customer and which offers roughly the functionality of a video player.

A specification in Albert is made up of (i) a graphical specification component in which the vocabulary of the specification is *declared* and of (ii) a textual specification component in which the admissible behaviors of agents are *constrained* through logical formulas.

Figure 1 contains part of the graphical declaration of the VoD system according to the Albert conventions. Each agent is represented by an oval and multiplicity is indicated by shadowing an oval. Figure 1 also declares the internal structure of the VoD agent. It declares the state structure and the actions that may happen during the lifetime of an agent and which may change the state of the agent. State components are represented by rectangles and actions are represented by ovals. State components are typed and actions can have typed arguments. Types may vary from simple data types to complex data types (recursively built using the usual data type constructors like e.g. set, sequence, table, etc.). From graphical conventions used in fig. 1, we know that *Movies* and *Display* are tables respectively indexed on MOVIE and ENDUSER (the type associated the identity of the End-User agent) while *List-cat* corresponds to a set of CATEGORY and is derived (see below) from the *Movies* component.

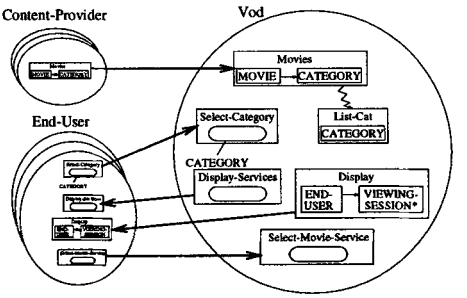

Fig. 1. The graphical declaration of the VoD system.

Besides graphical declarations, textual constraints are used for pruning the (usually) infinite set of possible lives associated with the agents of a system. As explained above, and also because we need this for *naturalness*, Albert supports two styles of specification. Those are reflected in 12 different templates of constraints which are classified into **Local Constraints** describing the internal behavior of an agent and **Cooperation Constraints** (describing the interaction of agents within a society).

State Behaviour templates express restrictions on the possible values that can be taken by the objects and data forming the state of an agent. These restrictions can be static (i.e. invariants which hold at any time) or dynamic (i.e. depending on time). An example is:

```
The display of the categories list (lc) to a given End-User (eu) does
not last for more than 1'.
```

$[\ \neg\ \mathsf{Lasts}_{1'}\ \ \mathsf{Display[eu]} = \mathsf{lc}\ ]$

Action composition constraints define how actions may be refined in terms of finely grained actions. Example is the following statement:

```
The Select-Movie-service action brought by an End-User has to be
followed by a Select-Category action made by this End-User or the
Display-Services-List.
```

Category-Selection
$\longleftrightarrow$ eu.Select-Movie-Service <(> eu.Select-Category(c) $\oplus$ Display-Services-List)

The interested reader can find the complete specification of the VoD system in [2]. The semantics of an Albert specification is given by mapping it to a real-time temporal action-based logic called Albert-Kernel.

3 Development of tools for ALBERT

At the moment, we have developed a first set of tools that are available and accompanied by a set of documents including a reference and a user manuals. Basically, the supporting environment has been developed according to an internet-based client/server architecture where The client part is called the "editor". It is running on a Windows 95/NT platform and offers a set of basic facilities which allow local work performed by the analyst. From a practical point of view, this is this part of the environement that we distribute. Facilities include:

- Editing the specification: both a graphical component as well as a textual component are available for editing an Albert specification. In particular, the two kinds of representation are managed in a consistent way (e.g., modifications brought at the graphical level are impacted at the textual level).
- Parsing the specification: at any moment, the analyst can decide to parse the specification in order to check its conformance with the syntax. The scope of the parsing has to be decided by the analyst. It can be concerned with the parsing of one specific Albert statement or with the whole (or a fragment) of the specification.
- Exporting the specification: the specification (both the graphical and the textual parts) can be cut and paste from the editor to any kinds of OLE compliant documents. For example, the graphical part can be put in a Word document. This is also possible to generate the specification in an Ascii format or a LaTeX format.
- Managing traceability links. Facilities are offered by the tools for managing the links existing between fragments of the formal specification and various semi-formal or informal sources of information coming from customers. This is possible, for example, to relate formal Albert statements to informal interviews, fragments of ERA diagrams or even videos capturing real-world scenes.
- Downloading the specification: at any moment, the analyst can decide to download his/her specification on the server through internet. This is when the analyst wants more complex checks (see below) be performed.

At the level of the server, the specification is stored in a deductive object-oriented repository called ConceptBase. On the specification are performed a set of more complex checks including a partial type checking as well as number of heuristics applied for discovering basic incompletenesses and inconsistencies. All results of these checks are presented to the client in a transparent way. The server part is running on top of a Unix platform. From a practical point of view (maintenance and future development), this server is located at the University of Namur.

We plan to achieve a second version of the tool in December 1999. In this version, we hope to integrate more advanced facilities at the validation and verification levels.

At the validation level, we work on:

- a so-called 'paraphrazer' supporting the analyst in a semi-automatic generation process of informal statements from the formal statements written with Albert templates;
- a distributed animation tool allowing the exploration of the different possible lives associated with a specification.

At the verification level, progresses are towards:

- a tool supporting a 'framework' for high-level reasoning. This is to support the analyst in a rigorous (conceptual) reasoning which is at the basis of the skeleton of a proof which can be passed to,
- the PVS theorem prover which will support a complete formal reasoning. To this end, a syntactic embedding of the Albert semantics is done as well a semantic one. The idea is to support both theorem proving and model-checking techniques.

The actual architecture of tools will evolve in order to reach a clean distributed objects architecture. According to the characteristics of the client platform, specific parts of the tool will be downloaded on-demand. With this respect we are investigating Corba, Java and JavaBeans.

Acknowledgements: Thanks are due to all the members of the ALBERT team: P. Du Bois, F. Chabot, L. Claes, P. Heymans, B. Jungen, M. Petit, J.M. Zeippen. This work is supported by the Walloon Region (DGTRE) Project CAT, contract nr. 2791.

References

1. P. du Bois, E. Dubois, and J-M. Zeippen. On the use of a formal requirements engineering language: The generalized railroad crossing problem. In *Third IEEE International Symposium on Requirements Engineering*. IEEE ICS Press, January 1997.
2. R. Wieringa and E. Dubois. Integrating semi-formal and formal software specification techniques. In *Information System Journal*. (to appear), June, 1998.