FrameKit and the prototyping of CASE environments

Fabrice Kordon & Jean-Luc Mounier,
Laboratoire d'Informatique de Paris 6
Université P.&M. Curie
4 place Jussieu, 75252 Paris Cedex 05, France
E-mail: Fabrice.Kordon@masi.ibp.fr, Jean-Luc.Mounier@masi.ibp.fr

1. Introduction

Software engineering methodologies rely on various and complex graphical representations as SA-RT [2], OMT [16] Class-Relation [6] etc. They are more useful when associated to CASE (Computer Aided Software Engineering) tools designed to take care of constraints that have to be respected. Such tools help engineers and facilitate the promotion of such methods.

However, implementation of such CASE tools is a complex task since they require various functions such as a graphical user interface, database facilities and, of course, the operations related to the methodology they implement (compilation, animation/simulation of specifications, code generation from specification, etc.).

CASE tools may share a common architecture and rely on a software platform [10] enabling the sharing of software components (i.e. a model compiler may produce information for both simulation and code generation) and easy tool evolution by addition of new functions.

The impact of the ECMA-NIST reference model is important in the Software engineering industry. CASE tools have now given way to CASE environments which may be adapted to a specific adaptation of a design methodology. Thus, tools architecture have changed dramatically to support the new possibilities derived from these techniques. The ECMA-NIST reference model has been derived to other application domains : for example, CORBA [5] is dedicated to the integration of various application at a the source level.

New Operating Systems, as Chorus, MACH, MacOS or Windows-95/NT may be considered as open shells, increasingly adapted to software plug-in. The evolution of Operating Systems goes with sophisticated implementation environments (and tools) composed of API's (Application Programming Interfaces). Some development environments now enable cross development over a set of target platforms. As an example, Metrowerks CodeWarrior on macintosh [15] allows cross development over a set of ***target architectures*** (hardware + OS) : Mac/MacOS, Mac/OpenStep and PC/Win32/x86, Java, Be/BeOS, embedded processor PPC821/860 and MagicCap.

In this paper, we present FrameKit, a software platform dedicated to the prototyping and quick implementation of CASE environments. FrameKit is valuable to quickly implement a CASE environment, either for evaluation purposes or as a final product, especially if graphical representations are involved. Our aim is to provide a generic CASE environment that can be filled with specific information enabling to customize it in discrete ways. FrameKit emphazizes the following aspects :

- A generic graphical user interface is able to be quickly adapted to a new graphical representation,
- A system platform manages users and proposes services offered by tools to end-users;
- The dynamic integration of software components is achieved without recompilation of the environment by means of configuration files;
- Enhanced API's for discrete languages is provided for Ada, C and Unix shell.

Section 2 shows how we parameterized a CASE environment and section 3 presents our interpretation of the five integration axis introduced in [19]. Then, section 4 presents main characteristics of FrameKit and section 5 focuses on the construction procedure of a dedicated CASE environment. We present an example and discuss advantages of this prototyping approach.

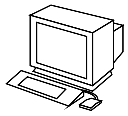
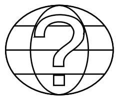
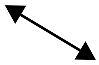
2. Parameterization of a software environment

A CASE environment helps system designers in that it offers them a set of services related to a given design methodology, based on a set of graphical or textual representations. Other useful services performing administration tasks (creation of a new user, installation, uninstallation) are also important to enable its operation for large projects.

Let us define the potential parameters of a generic CASE environment. It has to deal with :

i. manipulation of representations that are major entities for design methodologies and quite difficult to handle,
ii. tools and the services they provide (required data, outputs...),
iii. users and their characteristics (access rights...),
iv. storage of information (for representations and asso-

Formalism

shape	name	information
	**type=node** name= Computer	Host name, type, IP adress,
	**type=node** name=Hub	Type Throughput
	**type=edge** name=Cable	Throughput

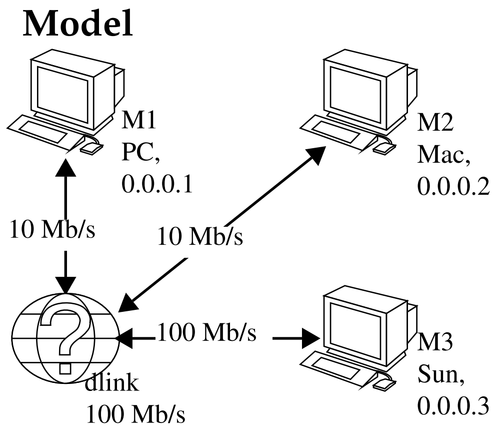


Figure 1 : *Relation between a formalism and a model.*

ciated results).

Points (iii) and (iv) are not parameters of a given CASE environment and will be discussed in next section. On the contrary, point (i) and (ii) are basic parameters that differentiate one CASE environment from another.

2.1. Formalisms and models

To deal with representation (point i), we distinguish two notions, formalism and model :

- A ***formalism*** describes a possible representation of a knowledge domain;
- A ***model*** is an instance of formalism (Figure 1). It is one representation expressed using the related formalisms.

In FrameKit, description of formalisms is object-oriented. This allows an easy management and updating of formalisms. Each class in the formalism is either a node or an edge (interconnecting nodes) and contains a set of labels (string values, digits...) that characterizes instances of the object. Additional information (how it looks, is it a link to a «sub-level» when the formalism is hierarchic, etc.) must also be provided to fully describe the formalism.

Figure 1 shows the relation between a formalism (on the left) and a model (on the right). A simple formalism dedicated to the description of a local network is defined on the left. It has three classes : two nodes («computer» and «hub») and one edge («link»). On the right a model designed according these rules is presented. Any object is identified by means of a set of labels (attributes in the sense of OO technology).

Thus, Models are formalism instances. A «tool palette» can be easily deduced from the formalism description. Its goal is to present the set of items that can be instanciated in a model.

Formalisms may be composed when they are hierarchical. In that case, some nodes are associated to another formalism. These nodes (called «boxes») can be «opened» to display its content in a new page. Models are thus composed of pages; each page is a part of the model.

Figure 2 shows how it works. In Figure 2.a, an node in a page is associated to another page. The description of the model structure (Figure 2.b) is an oriented graph in which, nodes represent pages and, edges links between a box and a page. So, hierarchical graphical descriptions are described using a set of (flat) formalisms.

2.2. Services

To enable a flexible access to tools' functions (point ii), we introduce the notion of ***service***. A service is a tool function that can be applied on a set of data. It produces results related to the associated model. A service is associated to a set of formalism. So, a user working on a model associated to one formalism in this set, potentially reach it.

2.3. Producing a dedicated CASE environment

Producing a dedicated CASE environment using FrameKit consists in the definition of the involved formalisms and

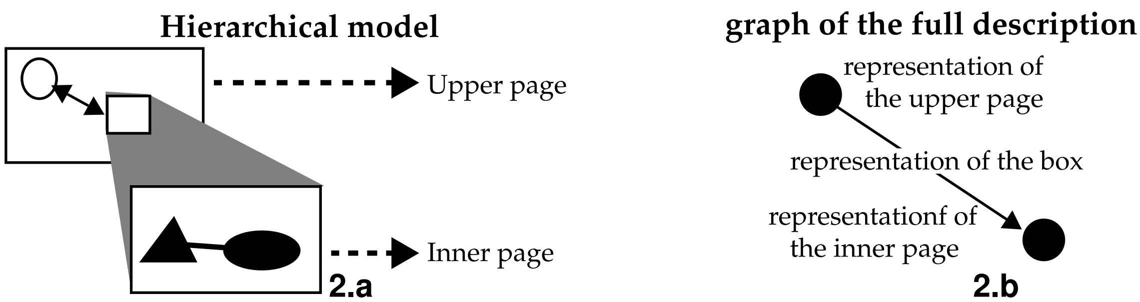


Figure 2 : *Hierarchical formalism.*

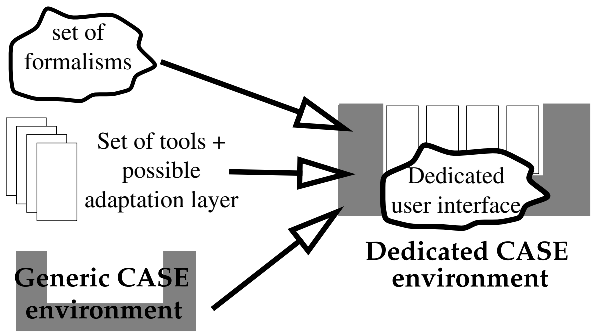


Figure 3 : *From the Generic CASE environment to the customized one.*

the implementation of associated tools. When tools are imported, they may need an adaptation layer (discussed in section 4.2.).

When formalisms and tools are ready. It is necessary to declare formalisms and define access to tools' functions by means of services, according to the methodology that is being implemented.

Figure 3 illustrates the production of a dedicated CASE environment from the generic one, a set of formalisms and a set of tools working on these formalisms.

3. The FrameKit environment

FrameKit is a generic CASE environment that only offers basic services to manage users, data and graphical formalism. In order to achieve a good design, [19] considers five integration axis : presentation, data, process, control and platform.

3.1. The presentation axis

The presentation axis deals with users' interaction. To ease both learning and use of an environment, it is valuable that any tool respects a common look and feel.

To meet presentation axis needs, FrameKit provides a generic user interface : Macao [14]. This user interface fulfills two major needs :

- graphical model editing, according to the rules of the corresponding formalism,
- service display and invocation.

Macao is a polymorphic editor able to manipulate models after the corresponding formalism description. A formalism is described by a file that contains the following information :

- declaration of classes (nodes and edges),
- declaration of labels associated to nodes and edges,
- declaration of connection rules between edges and nodes.

The construction of a new formalism does not imply any recompilation of Macao. All the required information is defined in an external file that expresses possibilities of the formalism.

Macao has two modes : ***standalone*** and ***connected***. A login procedure (with user identification) to FrameKit moves Macao into the connected mode. It then becomes both a graph editor and the «front end» of the platform. It displays menus downloaded by the platform and allows any user to select a service.

Macao is able to send a model description to FrameKit on request. Each description is divided into messages that carry out syntactical information only (the model description), aesthetic information (appearance of nodes and edges) remains at the user interface level because it is useless for tools.

Macao also achieves interactivity by means of simple widget-like mechanisms that build different types of dialog boxes. These dialog boxes may be activated either by the platform itself or by tools.

3.2. The data axis

The data axis deals with both data storage in a repository and data representation. The definition of representation standards, as well as unified access mechanisms are key points to ease data exchange between tools.

The basic representation in FrameKit relies on a message based approach. Each element in the model (nodes, edges, their relations and their labels) are stored using elementary messages. This description technique is generic because it works regardless any knowledge of the corresponding formalism.

Each node or edge instance has an internal and unique identifier. The node or edge class is referenced using a string that must be recognized by tools, the platform only carries out the information without analyzing it. So, the understanding of a model relies on conventions between the person who designs a model using a formalism and the tool that works on this model. Such a storage procedure is basic but available for any type of graphical model.

The same message based techniques is used when Macao transfers a model to FrameKit or for communication between platform processes. Messages are made using ASCII strings; this solves most portability problems as well as exploitation of data by programs running on discrete target architectures without having to use XDR mechanisms. In fact, all FrameKit data are stored in ASCII.

To achieve unified access mechanism in FrameKit, we type data using tool-defined keys and behaviors. Tool-defined keys are keywords used to find out an information in the FrameKit repository. The platform uses this information but does not have any knowledge of the corresponding semantics. Three types of data behavior correspond to three persistency approaches :

- ***model-associated*** data concern all the information associated to a model. It is useful to properly handle version management : when a model changes, associated results become obsolete and should be deleted and recomputed if needed. Such data are stored with the model description in a cell stamped by its last modification date. The cell is destroyed when the model is updated;

- ***user-associated*** data concern all the information related to a user (preferences, information potentially shared by models...). This information remains reachable until the user is deleted;
- ***global data*** concern all the information related to a CASE environment. It is stored in cells that may be associated to a tool, a formalism or to the platform itself (administration data only). Data last as long as the entity (tool, formalism or platform).

Whatever its behavior is, information may also be associated to a given target architecture (a program library for example). It is then tagged as well and the corresponding link is dynamically computed considering the characteristics of the host running the application.

3.3. The process axis

The process axis deals with the scheduling of operations according to the design methodology implemented in the CASE environment. Mechanisms to enable and disable services according :

i. the user's function in the project,

ii. the current state of the work.

FrameKit proposes two mechanisms to perform process integration :

- ***Access rights*** to services are defined for both users and groups. So, it is possible to meet point (i), hiding services to people for which access is inappropriate (i.e. do not have the proper role in a project according to the ECMA definition). Access rights are computed when a user get connected to the platform while conditions are evaluated each time a service is launched. When a user has no right on a service, it is not displayed to him;
- ***Service preconditions*** may be associated to services. So, they may be ordered according either to specific attributes (like «the service was never launched», «the services has been run OK» or «the service has been run with problems») or to session variables that may contain any string. When a new model is received by the platform, all services attributes are set to «service was never launched» and variables to an empty string. Both service attributes and variables are model-associated data; they are reset when the model is updated. When a service condition is unverified, it is displayed but dimmed.

3.4. The control axis

The control axis deals with the fine control of a tool : the platform must be able to run it properly. Standards must be defined in order to have a generic way to drive discrete components having discrete behavior and characteristics.

Like formalisms in Macao, declaration of a new service requires no recompilation of the platform. All the required information is provided in configuration files that contain :

- the external name of the service is used as an identifier for users. There is one external name per language supported in FrameKit;
- the internal name of the service is a unique name that identifies it. This name is necessary to obtain service attributes in service preconditions;
- the executable file name defines the tool that will be invoked when the menu item is selected;
- the executable file parameters are associated optional check marks in the service menu. If a check mark is enabled, the corresponding flag is set, otherwise not;
- extra information may also be provided about the tool execution like «can the service be softly interrupted».

3.5. The platform axis

The platform axis deals with the interaction of platform and tools with operating systems mechanisms (i.e. communication mechanisms). Encapsulation of basic mechanisms relying on target architecture characteristics is thus a key issue for both the environment portability and the integration of software components implemented out of FrameKit.

To achieve this point, we have parameterized some major concepts in the design of FrameKit :

- A high level communication model has been defined : several implementation are proposed (some may have restrictions). Then, any software component able to support one of these implementations should be easily integrated in FrameKit;
- A high level transmission of information by means of messages is built on top of the communication model, like the Macao widget-like mechanisms to manage interaction with users (mentioned in Section 3.1.);
- A repository (already mentioned in Section 3.2.) offers storage services. This repository hides File system related mechanisms (file naming system...).

4. Design and implementation of an application in FrameKit

Developing an application to be integrated in FrameKit is easy. Process integration axis is managed by the platform itself. So, Application should mainly care of the four other integration axis (presentation, data, control and platform).

4.1. Tool architecture and FrameKit APIs

To hide target architecture related mechanisms (and meet platform integration), all presentation, data, control axis are implemented and available for applications by means of Application Program Interfaces (API).

Three API's encapsulate required concepts and are available for discrete programming languages (currently,

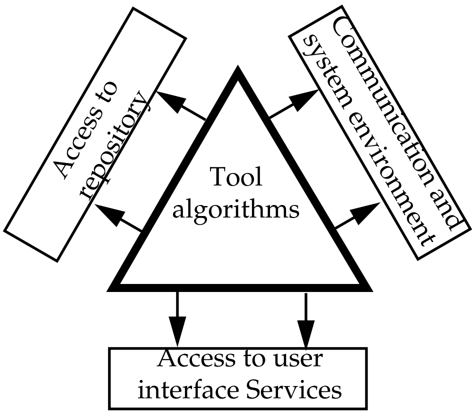


Figure 4 : *Software architecture of an executable program designed for FrameKit.*

Ada, C and Unix shell) :

- the first one encapsulates all accesses to the repository system. It deals with the dynamic construction of file links as well as the exploitation of stored models. A high level structure and associated services allows the manipulation of model pages (for hierarchic formalisms) as well as nodes and edges without having to care about the message based storage mechanisms. It is also useful to create results using the same functions. Results are stored in the repository like models (see Section 3.2.) and may thus be easily sent back to the user interface;
- the second one manages both communication mechanisms and system environment services (process creation, transmission/reception of flags). It deals with particularly delicate aspects of the execution environment. Tools that do not bypass this API should be portable to any of the supported target architectures;
- the last one deals with presentation aspects. It offers a unified (and simplified) look and feel for applications. It implements dialog boxes and all result display mechanisms offered by the Macao user interface.

The «main» program of an application implemented in the FrameKit environment is a part of the FrameKit libraries. Its goal is to properly initialize all required resources to operate the three API's and call the «tool main program».

The architecture presented in Figure 4 is strongly recommended : tool's algorithms should be expressed in a (set of) program unit(s) using functions of the provided API. The software architecture presented in Figure 4 is also the one of any platform components.

4.2. Integration of an application in FrameKit

The main classical integration procedures defined in [11, 9] are available in FrameKit :

- ***strong integration*** : if source code is available, it is possible to adapt it to fit the API described in the previous section;
- ***encapsulation*** : if only the executable file (reasonably disconnected from a graphical user interface) is available, it is possible to drive the tool by means of a specifically implemented process (Figure 5.a). If tool libraries are provided, this technique is still valid; in this case, the tool is composed of one executable file only (Figure 5.b).

Another technique, called rehosting, consists in the emulation of the imported software original execution environment [9]. This technique is yet available in FrameKit because no rehosting library has been implemented yet. However, such an implementation is technically possible (but highly time consuming and thus not planned).

4.3. FrameKit implementation

FrameKit is composed of four components (Figure 6) : a ***connection demon*** waits for connections from the ***user interface*** (Macao). When a connection is established, it forks. The father process waits for another connection. The son becomes a ***connection manager*** and manages the current connection.

A session is associated to each model opened during a session. Each session is handled by a ***session manager*** process. All FrameKit processes and tools communicates through a ***software communication bus*** [18].

Connection between services and tools, as well as the user interface customization are performed by means of configuration files. Thus, no recompilation of any FrameKit component is needed when a new service, or formalism is integrated.

This major capability allows one to quickly design a graphical representation (it automatically get the corresponding user interface) and related tools for evaluation purpose. The result can be successively improved and ex-

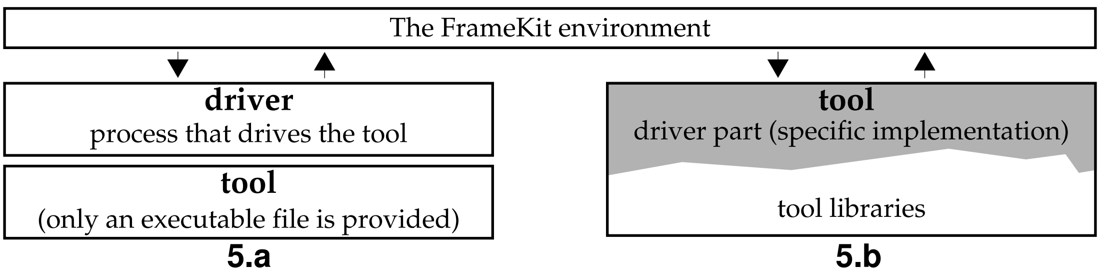


Figure 5 : *Encapsulation techniques in FrameKit.*

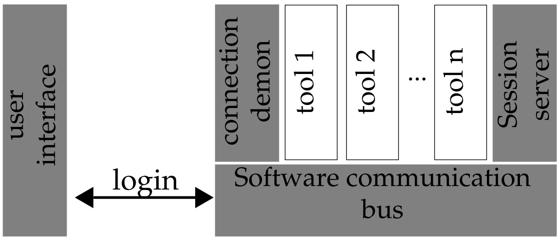


***Figure 6 :** The FrameKit software architecture.*

tended. FrameKit thus implements an evolutionary prototyping approach [3]. In the next section, we present CPN-AMI 2.0, a Petri-net oriented CASE environment built using FrameKit.

5. Building dedicated CASE environment

This section deals with the construction of a dedicated CASE environment on top of FrameKit. We first present an experimentation and then discuss the advantages of the procedure.

5.1. Building of the CPN-AMI 2.0 environment

The complexity of distributed systems is a problem when designers want to evaluate their safety. Systematic tests of the application cannot be considered as verification because of the potentially infinite number of states due to the parallelism.

Formal modeling is a key for verification of system specifications. Many formalisms such as Petri nets [13], Algebraic specifications [12] or B [1] are good candidates for the modeling and formal analysis of such systems.

CPN-AMI 2.0 is an environment dedicated to the modeling, the evaluation and code generation of distributed systems. The corresponding methodology is presented in [7] and involves three formalisms : OF-Class and H-COSTAM are dedicated to structured design of an application, and Colored Petri nets are used to evaluate interesting properties of distributed systems.

OF-Class focuses on the preliminary design and structuration of the system (coherence between software component interfaces).

H-COSTAM emphasizes the implementation aspects of the system (the mapping of conceptual solution into an operational software architecture).

Transformations from one formalism to another one are controlled as described in [8].

The CPN-AMI 2.0 environment is build on top of FrameKit, by addition of the following elements (Figure 7) :

- The description of the three formalisms customizes the user interface;
- A set of tools provides services for OF-Class : a syntax checker, a transformer into Petri nets (to evaluate the correctness of the model). A simulator is planned for preliminary debug of the specification;
- A set of tools provides services for H-COSTAM : a syntax checker, a transformer into Petri nets (to compute properties that could be used for code generation). A code generator is under development;
- A set of tools provides services for Petri net models : a syntax checker (for users who directly design a Petri net), a set of tools to compute properties of the model according to the Petri net theory and a simulator for model checking.

Some of these tools have been especially designed for the FrameKit environment. They where implemented using the provided program libraries and tested first out of the environment. Then, their integration mainly consisted in the registration of the corresponding service and its association with the proper program invocation. This registration of a completed tool takes approximately ten minutes; the longest declaration took about one hour (the tool's menu offered 35 services to declare).

Some others have been adapted from other CASE tools. For example, invariant computation (place invariants, transition invariants, traps and deadlock detection) for Petri nets are modules extracted from GreatSPN [4]. We only had executable files and input/output file exchange format. Integration has been performed using the technique of Figure 5.a. The main driver was written in Unix shell script. It successively invokes a program that translates FrameKit internal representation into the one of GreatSPN, the GreatSPN module corresponding that computes the expec-

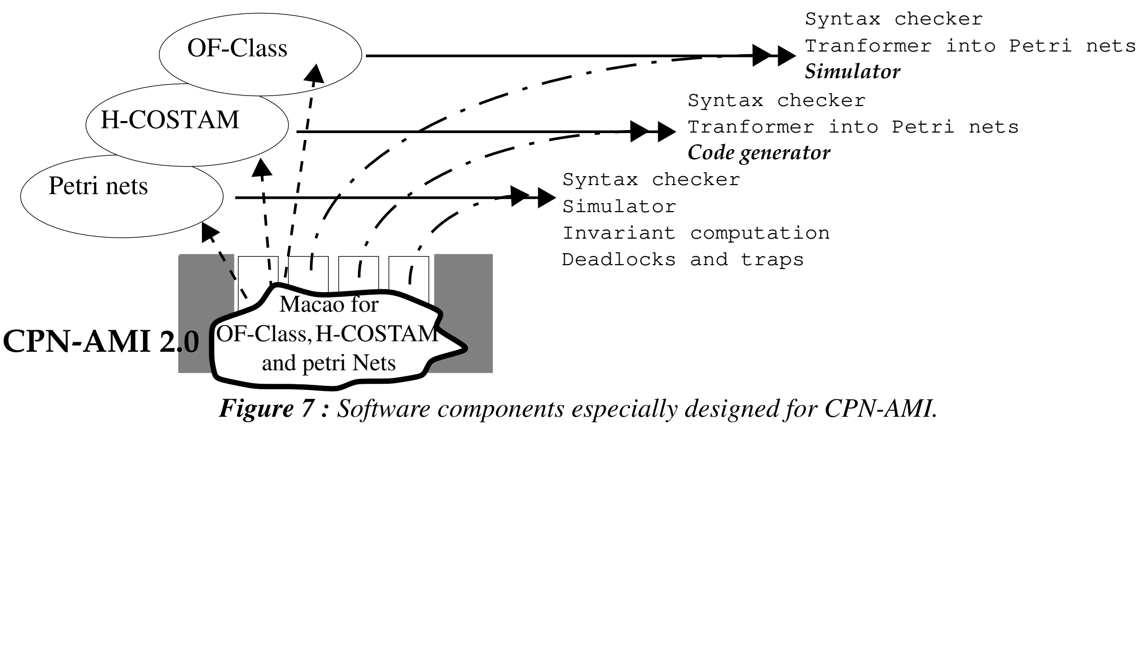


***Figure 7 :** Software components especially designed for CPN-AMI.*

ted invariant (with the appropriate parameters) and finally translates back the results to be displayed on the user interface. It took half a day to perform the whole tool adaptation and service registration.

CPN-AMI is still being improved and extended by addition of new tools in order to cover and fully implement our design methodology for distributed systems.

5.2. Benefits in the construction of a dedicated CASE environment

After having experimented FrameKit, it appears that using it to build a CASE environment raises two major advantages.

The first one deals with the user interface. The design of new graphical representation with is very easy, even compared with Tcl/tk based approaches [17]. Macao already implements all editing functions (cut/copy/paste, alignment functions etc.) and the design of a new formalism consists in writing a configuration file that describes possible entities according to the principles presented in section 2.1. So, it is very easy to build numerous «prerelease» of a graphical representation to check its use.

The second one is related to the definition of communication techniques between tools. Involved mechanisms are quite complex and it is easier to take benefits from an already existing standard. A proposal to the five integration axes introduced in the ECMA reference model is thus a key issue in FrameKit.

6. Conclusion

In this paper, we have presented FrameKit, a software platform dedicated to the rapid prototyping of CASE environment. FrameKit architecture is similar to the ECMA reference model [10].

It focuses on the quick definition of new graphical representations, handled by a generic user interface.

FrameKit also contains a development kit composed of program libraries in various languages (C, Ada and Unix shell). This eases both the implementation and integration of new tools. FrameKit also has good capabilities to integrate software components that were not especially designed for it.

FrameKit is operational and we have used it to build the CPN-AMI 2.0 CASE environment that implements a design methodology for distributed systems described in [8]. To achieve it, we had to describe three formalisms (one is hierarchical) and register the services of a dozen of tools.

The first version, released on Internet in the first quarter of 1997 (`<http://www-masi.ibp.fr/framekit>`) is quite satisfactory. We plan to enrich it with new mechanisms to ease the quick design of code generators as well as model simulators.

7. References

[1] R. Abrial, "The B-book", Cambridge University Press, 1995.

[2] D.Hatley & I.Pirbhai, "Strategies for real-time system specification", Donset house publishing Co, 1988

[3] R. Budde, K. Kuhlenkamp, L. Mathiassen & H. Züllighoven, "Approaches to prototyping", Springer Verlag, Berlin, 1984.

[4] G. Chiola, "A Graphical PN/Tool for Performance Analysis", International Workshop on modelling techniques and performance evaluation,AFCET, Paris, March 1987.

[5] OMG document, "CORBA 2.0", July 1995

[6] Desfray P., "Object Engineering, the Fourth Dimension", Addison-Wesley, 1994.

[7] A. Diagne & P. Estraillier, "Formal Specification and Design of Distributed Systems", In Proceedings of the First International Workshop FMOODS'96, Paris, March 1996.

[8] A.Diagne & F. Kordon, "A Multi Formalisms Prototyping Approach from Formal Description to Implementation of Distributed Systems", à paraître in proceedings of the 7th "International Workshop on Rapid System Prototyping", N.Kanopoulos Ed, IEEE comp Soc Press, Greece, June 1996.

[9] ECMA, "Portable Comon Tool Environment", Technical Report ECMA-149, European Computer Manufacturers Association, Geneva, Switzerland, December 1990.

[10] ECMA, "A Reference Model for Frameworks of Stoftware Engineerings Environments", ECMA report number TR/55 (version 3), NIST Report, April 1993.

[11] C.Fernstrom & L.Ohlsson, "The ESF Vision of a Software Factory", Proceedings of the International Conference on Software Development Environments & Factories, Berlin, May 1989.

[12] M.C. Gaudel, "Algebraic Specifications", Chapter 22 in "Software Engineer's Reference Book", John Mc Dermid ed, Butterworth, 1991.

[13] K. Jensen, "Coloured Petri Nets. Basic Concepts, Analysis Method and Practical Use (vol 1)", EATC Monographs on Theoretical Computer Science, Springer Verlag 1992.

[14] Macao Home page, ttp://www-masi.ibp.fr/macao

[15] Metrowerks home page, http://www.metrowerks.com/

[16] Rumbauh & Blaha, "Object Oriented Modeling and Design"; Prentice Hall; 1991

[17] J. Ousterhout, "Tcl and the Tk Toolkit", Addison-Wesley Publishing Company, 1995

[18] Tooltalk Home page, http://brunhilda.sensor.com/manuals/tooltalk/

[19] A.Wasserman, "Tool Integration in Software Engineering Environments", proceedings of the International Workshop on Environments, Chino, September 1989 and LNCS 467 : "Software Engineerings Environements", pp 138-150.