Xchart-Based Complex Dialogue Development

Fábio N. Lucena
fabio@dcc.unicamp.br

Hans K.E. Liesenberg
hans@dcc.unicamp.br

Luiz Eduardo Buzato
buzato@dcc.unicamp.br

Relatório Técnico DCC–95-07

Junho de 1995

Xchart-Based Complex Dialogue Development[*]

Fábio N. Lucena
fabio@dcc.unicamp.br

Hans K.E. Liesenberg
hans@dcc.unicamp.br

Luiz Eduardo Buzato
buzato@dcc.unicamp.br

Projeto Xchart[†]
DCC/IMECC/UNICAMP
Caixa Postal 6065
13081-970 Campinas/SP, Brazil
e-mail: xchart@dcc.unicamp.br
WWW: http://www.dcc.unicamp.br/projects/Xchart

Abstract

Xchart is a research software system that provides tools and facilities for the development of computer human interfaces. Control aspects of distributed reactive system (computer human interface) can be modelled through the use of constructs of the Xchart specification language. In this particular domain, greater attention is devoted to the subclass of multi-thread dialogues. Some of the constructs of the language have been tailored for this particular class of dialogue. By means of various tools, the environment supports different development stages ranging from diagram construction in the specification phase, to their execution. A simple example is used to illustrate Xchart's main features.

[Key-words:] distributed reactive systems, computer-human interfaces, specification/implementation of multi-thread dialogues.

1 Introduction

The development of computer-human interfaces (interfaces, for short) can be split into two different stages: the interaction design and the construction of the corresponding software (right of figure 1). The interaction design is an activity carried out by experts of different areas with no "golden rules" to assure the desired quality. It determines how an interface should look like and how it should behave. Consecutive software design of the desired interface and its implementation is also known to be a non-trivial task. Despite of steady progresses, many challenges still remain [8]. This work proposes a visual language called Xchart and a corresponding environment for the development of the control components of an interface. Xchart is well suited to describe complex multi-thread dialogues, ie simultaneously active dialogues which allow the final user to carry out several interacting, concurrent and possibly distributed tasks. This kind of dialogue are becoming more and more used and

[*]Simpósio Nipo-Brasileiro de Ciência e Tecnologia, Campos do Jordão/SP, 08/1995.
Partially funded by Fapesp, CAPES, CNPq/ProTem-CC/GEOTEC (Unicamp-UFG) and Microsoft.

[†]Project that deals with the development (specification and implementation) of user interfaces. Just as a curiosity: the 22nd letter of the greek alphabet ($\mathbf{X}$ e χ) is identified in English by the name CHI which represents the acronym of *Computer-Human Interface.*

represent one of the obstacles to interface development [6, 8]. The Xchart language represents a proposal for the specifications fo the complex dialogue and the Xchart environment supports the generation and execution of software which implements those dialogues.

The design of the Xchart language is an outcome of earlier experiments with statecharts [2] used during the development of interface dialogue control [5]. The major difficulties during this earlier stages subsidized the proposal of changes and additions to the statechart notation which produced the Xchart language. The *statechart* notation was originally conceived to describe control aspects of applications belonging to a relatively wide range category of reactive systems with static architectures [2].

The environment briefly described below intends to give support to the user of the Xchart language in tasks related to the specification and implementation of complex dialogues. The environment provides and editor, a compiler (Xchart $\rightarrow$ C++) and a module capable of executing possibly distributed specification described in Xchart. A code generation method from *statechart* specification has been proposed in [4]. This method contributed to the definition of the corresponding environment as well as other related work. An Xchart simulation, as done for statecharts [1], has not yet been considered.

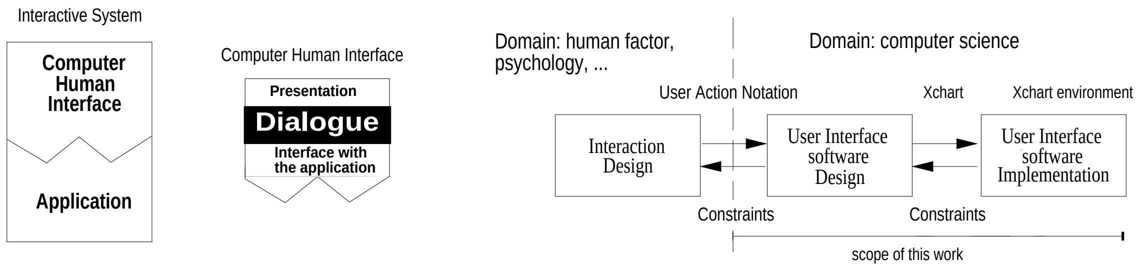


Figure 1: The interface and the development model of interactive systems.

Some concepts and the context of the present work. The left side of the figure 1 presents a reactive system divided in two major components: the interface and the application. The application represents the component responsible for the functionality of the system and its development is beyond the scope of the present work. A significant part of what is produced in software engineering is devoted to the development of this particular component. The ultimate purpose of the interface is to give access to the end user to the application's functionality and to present results produced by the latter. An interface may be divided, as illustrated in the central part of the figure 1, into three layers: presentation, dialogue and interface with the application. The dialogue layer may be related to the syntax of the interaction between the end user and the application. One of the first notations used to describe the control flow of interface dialogues was the BNF normally used to describe grammars of programming languages. Very soon it was replaced by more intuitive STD's (*state transition diagrams*). Shortcommings of STD's were responsible for the definition of RTN's (*recursive transition networks*). Extensions made on RTN's produced ATN's (*Augmented Transition Networks*). More recently *statecharts* were proposed which combine elegantly concurrency, hierarchy and communication. A relatively great number of related projects use this particular notation or variants to specify dialogue control. Some difficulties run into while using statecharts control aspects of interfaces are described in [5]. In the present work a statechart extension, called Xchart, is being used.

Issues related to interaction design are not of a direct interest to the present work. For its scope, the development of an interface starts with a description of an interaction design produced by appropriate experts and is concluded with its implementation (software). The right side of figure 1 characterizes these two domains. The Xchart language was designed for the development stage in order to support software engineers during the construction of computer-human interfaces.

Notations (like the UAN – User Action Notation [3]) have been specifically designed for the description of interaction designs. They mostly provide minimal support for products of later development activities which have to be captured by other notations of different kinds. The difficulties of implementation, costs and required time related to the use of a variety of notations, tend no inhibit the interaction designer who tries to avoid the use of "expensive" facilities as, for example, more elaborated dialogues. The Xchart notation may be used as well to describe a part of the interaction design related to control aspects of an interface.

The implementation of complex dialogues has to take into consideration typical aspects of concurrent programming (as race conditions and communication). Concurrent programming languages have been considered to describe complex dialogues. They have originally been designed, however, for the implementation phase and they do not contribute to the reduction of the complexity at the specification phase since many implementation aspects have already to be considered very early on and programmed painstakingly in an error-prone process and hard to change. The Xchart notation provides facilities to describe control and, since it is formal, it allows to transfer the complexity related to the implementation of the control to the environment which generates and executes the appropriate code. Even specific languages designed for the development of interfaces do not in general provide more sophisticated facilities to capture control aspects. They handle equally all of the components of an interface or focus on different aspects related mostly to the presentation [7].

The major difficulty related to those languages is not related to their inherent limitations, but to the absence of a more intensive use of specification languages and to the gap between methods generally employed to produce interfaces to the necessities of the developer. Our experiences with Xchart, on the other hand, reveal that the proposal presented harmonizes with tools commonly used to develop interfaces: toolkits and C++ [9].

TEXT STRUCTURE. Section 2 describes the execution model of the Xchart language. Section 3 illustrates its use and Section 4 is dedicated to the final remarks.

2 The Xchart Model

In order to understand the structure of a system developed in Xchart it is necessary to understand the typical organization of systems which support complex dialogues or which allow a concurrent execution of an interface and its related application. The concurrency may be present solely in the application, in the interface, or both. Even relatively simple applications may have this kind of profile. Interactive systems responsible to maintain many distinct presentations of a same information or even those which allow the end user to interact while running the application concurrently are examples of this kind of systems.

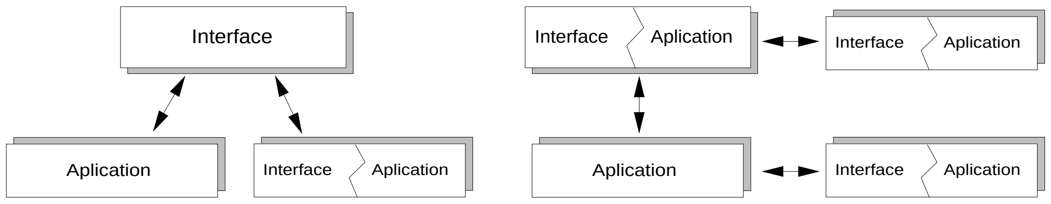


Figure 2: Typical relationship between threads.

The figure 2 illustrates the most common situations. Each box represents a different thread which may or may not be interacting with the end user. A thread is assumed to be an execution entity with a unique identifier. A process may hold many threads eventually run on different processors. Each of them may contain zero or more objects (grain of control in Xchart). These objects may be related to the dialogue of an interface, the control of the application or the communication manager between those entities. From now on, each thread which executes objects or code of the application and/or interface shall be called a **client**.

In this scenario, clients are scattered on different nodes of a network, create and destroy objects, exchange data and get synchronized by means of services provided by the runtime system (RS). The left part of figure 3 presents the runtime system described in terms of layers on the right side of the same figure. Objects and the runtime system interact asynchronously.

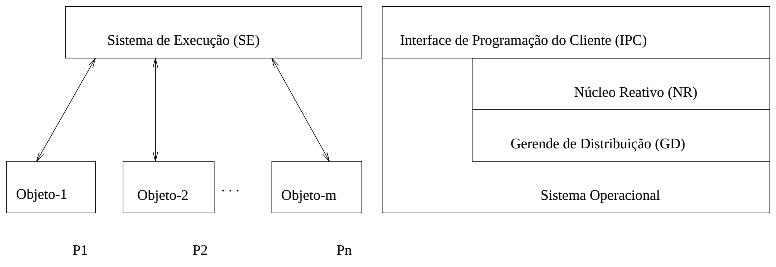


Figure 3: Relationship between objects and the RS. RS's architecture.

Figure 3 (left) presents m distributed objects executing on processors $P_1, P_2, \ldots, P_n$. In general $m \neq n$ as well as different of the total number of clients. The distribution manager (DM) is responsible for the support of the concurrent execution of those objects. On each processor with clients a reactive kernel (RK) is running to give support to the execution of Xcharts. An RK provides the behaviour of each object according to its Xchart specification passed as a parameter at the occasion of a reaction to an event which has the particular object as its target.

The communication between an RK and its clients is being illustrated on the right side of figure 4. Through the client programming interface (CPI), the objects have access to the facilities provided by the Xchart runtime system. The event queue contains events, data

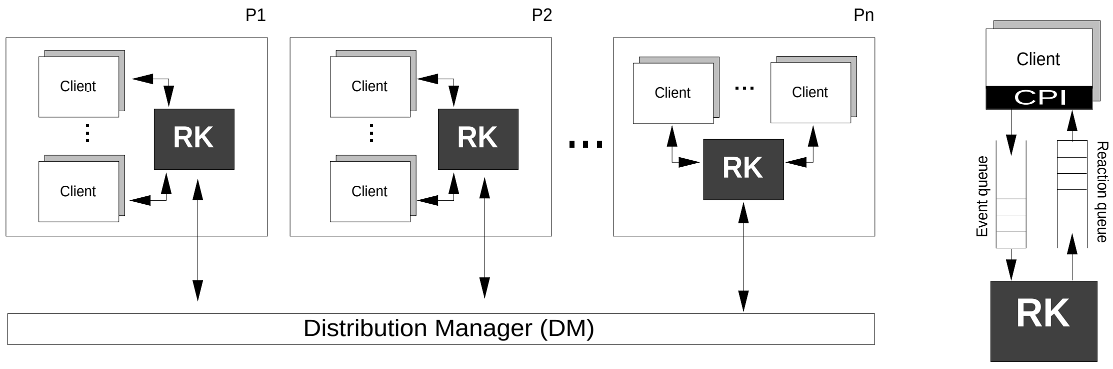


Figure 4: Xchart's distribution model and the communication between the RK and clients.

or service requests to be handled by the RK. For instance, a client may generate an event related to an action of the end user (`Help` key pressed) and signal immediately through the CPI its occurrence (followed by the related dada, if the case) to the RK. The RK, if necessary, contacts the DM and gets/supplies the information needed to perform the reaction according to the relevant Xchart specification. Whenever methods (implemented by the clients) take part in a reaction, they are invoked. The runtime system is responsible for the invocation of those methods. (NOTE: Objects and methods are used in the same sense as in C++.) Parameters required by methods may be passed as data (provided by clients) attached to events.

The data flow related to the runtime system is illustrated in figure 5. The RK interprets a given Xchart specification (Xchart) associated to an object (ObjectId), target of an event (Event, possibly carrying data) and the configuration (set of its active states) plus control variables of this object (Status). The RK may need some information locally available or execute internal routines (Internal reaction) in order to obtain distributed information. The output of the RK is put in the reaction queue of the target object. The CPI library linked to the client's code invokes the required methods according to the information being held in the reaction queue. During the execution of a specification changes of control variable values and of the object's configuration are common place. Thus, the prior Status may be altered and produces a NEW STATUS.

The Distribution Manager (figure 4) does not handle the information received from or send to the RK's, but provides solely services (like communication facilities, name server and atomic transaction mechanism). An RK request may lock/unlock a shared resource, get/update a value of a control variable as well as obtain information about Xcharts configurations running on remote RK's. In short, an RK is responsible for the interpretation of client behavioural specifications executing on a give node of a network whenever they are targets of events. Information about the state of control variables and of the configuration of a specific Xchart may be requested by a variety of active RK's. The DM takes care of the communication between RK's. The steps described bellow illustrate the typical behaviour at runtime of a system developed in Xchart:

(1) Clients create objects.

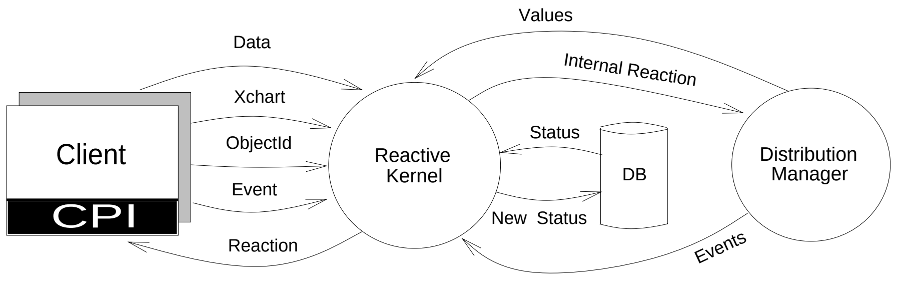


Figure 5: Data Flow.

(2) An event e is signalled to an object O_k executing on a processor P_i via CPI. Data is possibly associated to the particular event.

(3) The RK reacts to the occurrence of e_l. This reaction is derived from the active Xchart specifications and possibly locks control variables, updates values and invokes routines. If data is associated to the event, then the RK starts as well a transfer of the associated data to all Xcharts where e provoked a perceivable reaction. A reaction may be seen as an addition of various elements to one ore more reaction queues.

(4) The library linked to the client code calls the methods selected by a reaction.

Only after a reaction ceases, a new event may be signalled to an RK. The development process is briefly presented in figure 6. The software design of a client is related to the interface as well as to the application software design. The interaction model of an interface with a client is a provided abstraction used by the client code to signal events to Xcharts and to get hold of the resulting control tags. At the moment there are pre-defined classes which provide the access to the runtime system while methods of objects belonging to a client represent the mechanism used by the runtime system to express changes of the current state of Xcharts. The CPI library is responsible for the invocation of the client methods and it is linked to its code. The result is a module of complex behaviour, whose synchronization and communication with other clients is carried out by the runtime according to the Xchart specification of each object associated to the client.

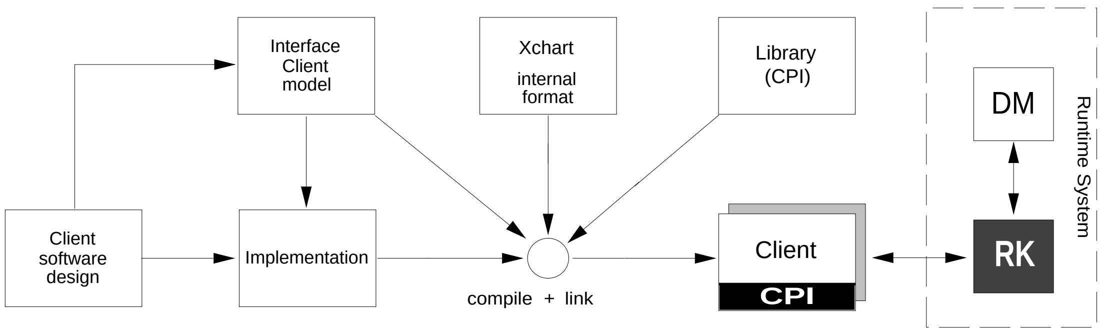
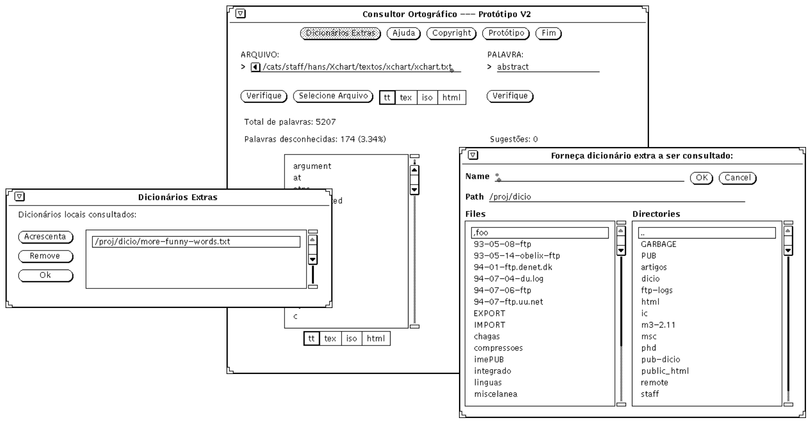


Figure 6: Xchart based development process.

3 Example: a spelling checker

A spelling checker for the Portuguese language is in use at the DCC/IMECC/UNICAMP for over two years. Its input may be provided in four possible formats. Each word is checked against words in a basic dictionary and, if not found in this dictionary and possibly other supplied dictionaries by the user, it is declared as an unknown word. For unknown words alternatives may be suggested. A variety of interface prototypes have been produced for the spelling checker. The version presented here is the latest one. The latest prototype allows the exchange of an unknown word by a suggested alternative in the submitted text. A more detailed description of the spelling checker is beyond the scope of the present paper. Figure 7 captures an interaction snapshot with the spelling checker.



Consultor Ortográfico --- Protótipo V2

ARQUIVO:
> /cats/staff/hans/Xchart/textos/xchart/xchart.txt

PALAVRA:
> abstract

Total de palavras: 5207

Palavras desconhecidas: 174 (3.34%)

Sugestões: 0

argument
at
c

Dicionários Extras

Dicionários locais consultados:

/proj/dicio/more-funny-words.txt

Forneça dicionário extra a ser consultado:

Name *

Path /proj/dicio

Files	Directories
,foo	..
93-05-08-ftp	GARBAGE
93-05-14-obelix-ftp	PUB
94-01-ftp.denet.dk	artigos
94-07-04-du.log	dicio
94-07-06-ftp	ftp-logs
94-07-ftp.uu.net	html
EXPORT	ic
IMPORT	m3-2.11
chagas	msc
compressoes	phd
imePUB	pub-dicio
integrado	public_html
linguas	remote
miscelanea	staff

Figure 7: Interaction snapshot with the spelling checker.

A set of (Unix-like) tools implement the functionality of the spelling checker. These tools are clustered together in the Application module depicted in figure 8. One process runs application code and another one code related to the interface. The communication between both processes is performed via Xchart events.

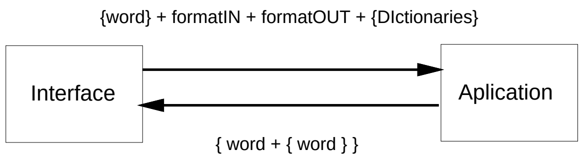


Figure 8: Data flow between interface and application.

Figures 9 and 10 illustrate the behaviour of the spelling checker related to its more common tasks: text verification, search and replacement of words. The state **Spelling Checker** on the left side of the figure is the ancestor of the state **Editing** with a small triangular tag which indicates that it does not represent a basic state. The state **Main** is activated and consequently the state NEUTRAL, since it is the first time that **Main** has been activated.

Once more it is important to point out that the control aspects described by means of an Xchart triggers actions belonging to the interface and the application. For example, whenever the state **Neutral** becomes active for the first time, the window presented in figure 7 is produced as a result of those actions. In a similar way initialization procedures may have some effect from the applications point of view.

An other example: Within the state **Select Word** there are various action specifications similar to transition labels. The difference in terms of behaviour is that if an event referred to within a particular state occurs the related action is triggered without changing the configuration of the Xchart, ie no deactivation of states followed by activations take place. For this reason the action specified at the right lower corner of the state **Editing** causes the execution of the action `save()` whenever this state is active and the event *save* occurs. This action causes a semantic action (the application must save information to the relevant file) as well as a lexical (the interface provides feedback information to the user indicating that information has been downloaded to the relevant file). Similarly to the presentation of the initial window while state **Neutral** is active, the contents of the corresponding file is presented to the user when the state **Editing** is active.

Still another peculiar Xchart features is shown in figure 9. Although the specification of the spelling checker is composed of a collection of isolated Xcharts, they operate all concurrently from the conceptual point of view. In other words, the states **Dictionaries**, **Spelling Checker** and **Check Word** may all be active at the same time. In this particular case, a sequential implementation (a sole processor) of these Xcharts is equivalent to a particular serialization of their executions. If state **Neutral** of **Editing** is active and the event *dic* is generated in accordance to the specification, then the event *show* shall be signalled to all other executing Xcharts. This means that the event *show* is inserted in the event queues of the other Xcharts. The state **Dictionaries** may be particularly be affected by this event.

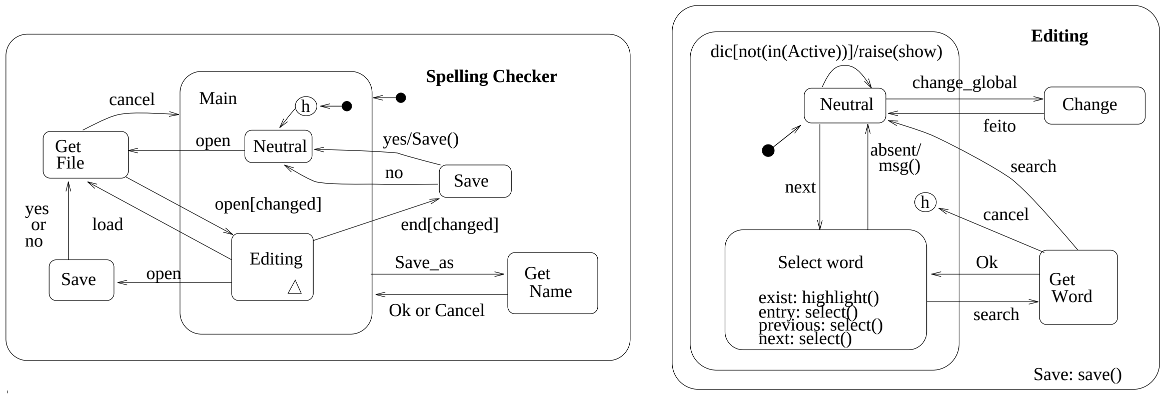


Figure 9: Basic Xcharts of the spelling checker.

The left side of figure 10 describes the behaviour of the system related to the dialogue box employed by the end user to provide a set of complementary dictionaries to be considered by the spelling checker. When state **Dictionaries** is activated, the substate **Wait** is as well activated by default. The Xchart remains in this particular state until the event *show* is generated by the end user. This event captures the intention of the user to modify the set of dictionaries in use at this particular moment. Only after the occurrence of this event the dialogue box is made visible to the end user. If a new dictionary is to be added, the event *add* deactivates the current state (**Neutral** or **Altered**) as well as the state **Activated** and then activates the state **Select**. This latter state presents a dialogue box which allows the end user to navigate through the file system and select a particular dictionary to be added to the set of dictionaries in use. The dictionary is added by means of the action `add()` and state **Altered** is activated. If the end user does not want to continue, then the event *cancel* restores the prior state by entering via history into state **Activated**.

The right side of figure 10 represents the behaviour of the dialogue box related to the checking of a particular word.

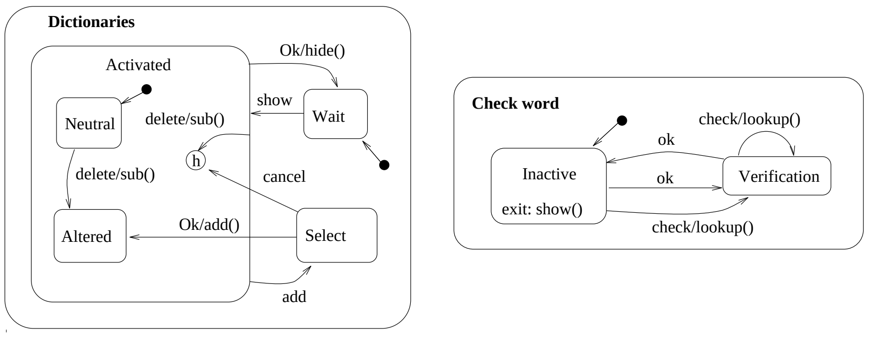


Figure 10: State **Dictionaries** and state **Check Word**.

The specification of the spelling checker sketched out above only uses a subset of the facilities provided by the notation and the environment. Nevertheless it is adequate to illustrate the basic concepts. Once a behaviour is described in terms of an Xchart its implementation has to follow. For each Xchart specification, a C++ class is generated by the environment. All generated classes are abstract classes and must be inherited by classes which implement the clients. The latter ones must define the methods which shall be invoked whenever a reaction to particular events take place in accordance to the relevant Xchart specification. For instance, on the right side of figure 10 one of the Xcharts is shown which integrate the behavioural specification of the spelling checker's interface. In figure 11 the class which implements the object of the corresponding client is shown. The class `myCheckWord` inherits from class `CheckWord` which is generated by the environment. This inheritance is sufficient to allow the CPI invoke appropriate methods, whenever necessary.

CURRENT STATE. The implementation of the example uses a version of the RK which does not yet handle all facilities provided by the Xchart language [4]. In parallel to the definition of the intricate semantics of the Xchart facilities, the corresponding implementation of the RK is evolving. An Xchart editor which handles hierarchies presented in a tree-like fashion (instead of nested nodes with rounded corners) and controlles the specialization of inherited behaviour, an Xchart parser and the services needed for the distribution are being designed and implemented. Small prototypes of distributed systems have already been built

```
class myCheckWord: public CheckWord{
  public:
     myCheckWord(int Id) : CheckWord(Id) {};
     void show(xchartE e) { theApp.DialogDic->Show(TRUE); };
     void lookup(xchartE);
};
```

Figure 11: Client Code.

to validate the presented proposal. Priority has been given to decisions concerning the typical organization of interface code. For instance, the use of objects responsible for providing control allow that widgets give support to complex behaviour without abandoning policies adopted by toolkits.

4 Final Remarks

Experiments with statecharts helped to identify improvements and additional constructs for the specification of dialogue control in distributed environments [5]. The present work presents Xchart, a product of these experiments, as well as an execution model and an environment to aid the development of complex dialogues. The environment represents an extension of an earlier work [4].

More Xchart specifications have to be built and analysed in order to validate the concepts presented in this paper. A fine tuning shall probably be carried on as a result of new experiments. An integration of the new facilities with a toolkit devoted primarily with presentation aspects of an interface into a sole environment is expected.

For a designer Xchart provides appropriate abstractions to describe control aspects of interfaces. The corresponding sophisticated code is not anymore produced by application programmers neither "vanishes," but it is produced automatically by the environment by means of objects. For graphical interface programmers, Xchart extends toolkits with facilities they do not provide in general: sophisticated control mechanisms.

References

[1] João W. L. Cangussu, Paulo Cesar Masieiro, and José Carlos Maldonado. Execução Programada de Statecharts. *Revista Brasileira de Computação*, 7(2):3–14, Jan/Jun 1994.

[2] D. Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8(3):231–274, June 1987.

[3] Deborah Hix and H. Rex Hartson. *Developing User Interfaces: Ensuring Usability Through Product & Process.* John Wiles & Sons, Inc., 1993. ISBN 0471-57813-4.

[4] Fábio N. Lucena and Hans K. E. Liesenberg. A Statechart Engine to Support Implementations of Complex Behaviour. *XXI Semish*, 1994. Caxambu/MG, Brazil.

[5] Fábio N. Lucena and Hans K.E. Liesenberg. Reflections on Using Statecharts to Capture User Interface Behaviour. *Proceedings of XIV Int. Conf. of the Chilean CSS*, October 1994.

[6] Alan Morse and George Reynolds. Overcome Current Growth Limits in User Interface Development. *Communications of the ACM*, 36(4):73–81, April 1993.

[7] Brad A. Myers, editor. *Languages for Developing User Interfaces.* J&B, 1992.

[8] Brad A. Myers. Challenges of HCI Design and Implementation. *Interactions*, 1(1):73–83, 1994.

[9] Brad A. Myers and Mary Beth Rosson. Survey on User Interface Programming. In *CHI'92 Proceedings*, pages 195–202, Monterey, California, May 1992.

Relatórios Técnicos – 1992

92-01 **Applications of Finite Automata Representing Large Vocabularies,** *C. L. Lucchesi, T. Kowaltowski*

92-02 **Point Set Pattern Matching in d-Dimensions,** *P. J. de Rezende, D. T. Lee*

92-03 **On the Irrelevance of Edge Orientations on the Acyclic Directed Two Disjoint Paths Problem,** *C. L. Lucchesi, M. C. M. T. Giglio*

92-04 **A Note on Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams,** *W. Jacometti*

92-05 **An (l, u)-Transversal Theorem for Bipartite Graphs,** *C. L. Lucchesi, D. H. Younger*

92-06 **Implementing Integrity Control in Active Databases,** *C. B. Medeiros, M. J. Andrade*

92-07 **New Experimental Results For Bipartite Matching,** *J. C. Setubal*

92-08 **Maintaining Integrity Constraints across Versions in a Database,** *C. B. Medeiros, G. Jomier, W. Cellary*

92-09 **On Clique-Complete Graphs,** *C. L. Lucchesi, C. P. Mello, J. L. Szwarcfiter*

92-10 **Examples of Informal but Rigorous Correctness Proofs for Tree Traversing Algorithms,** *T. Kowaltowski*

92-11 **Debugging Aids for Statechart-Based Systems,** *V. G. S. Elias, H. Liesenberg*

92-12 **Browsing and Querying in Object-Oriented Databases,** *J. L. de Oliveira, R. de O. Anido*

Relatórios Técnicos – 1993

93-01 **Transforming Statecharts into Reactive Systems,** *Antonio G. Figueiredo Filho, Hans K. E. Liesenberg*

93-02 **The Hierarchical Ring Protocol: An Efficient Scheme for Reading Replicated Data,** *Nabor das C. Mendonça, Ricardo de O. Anido*

93-03 **Matching Algorithms for Bipartite Graphs,** *Herbert A. Baier Saip, Cláudio L. Lucchesi*

93-04 **A lexBFS Algorithm for Proper Interval Graph Recognition,** *Celina M. H. de Figueiredo, João Meidanis, Célia P. de Mello*

93-05 **Sistema Gerenciador de Processamento Cooperativo,** *Ivonne. M. Carrazana, Nelson. C. Machado, Célio. C. Guimarães*

93-06 **Implementação de um Banco de Dados Relacional Dotado de uma Interface Cooperativa,** *Nascif A. Abousalh Neto, Ariadne M. B. R. Carvalho*

93-07 **Estadogramas no Desenvolvimento de Interfaces,** *Fábio N. de Lucena, Hans K. E. Liesenberg*

93-08 **Introspection and Projection in Reasoning about Other Agents,** *Jacques Wainer*

93-09 **Codificação de Seqüências de Imagens com Quantização Vetorial,** *Carlos Antonio Reinaldo Costa, Paulo Lício de Geus*

93-10 **Minimização do Consumo de Energia em um Sistema para Aquisição de Dados Controlado por Microcomputador,** *Paulo Cesar Centoducatte, Nelson Castro Machado*

93-11 **An Implementation Structure for RM-OSI/ISO Transaction Processing Application Contexts,** *Flávio Morais de Assis Silva, Edmundo Roberto Mauro Madeira*

93-12 **Boole's conditions of possible experience and reasoning under uncertainty,** *Pierre Hansen, Brigitte Jaumard, Marcus Poggi de Aragão*

93-13 **Modelling Geographic Information Systems using an Object Oriented Framework,** *Fatima Pires, Claudia Bauzer Medeiros, Ardemiris Barros Silva*

93-14 **Managing Time in Object-Oriented Databases,** *Lincoln M. Oliveira, Claudia Bauzer Medeiros*

93-15 **Using Extended Hierarchical Quorum Consensus to Control Replicated Data: from Traditional Voting to Logical Structures,** *Nabor das Chagas Mendonça, Ricardo de Oliveira Anido*

93-16 $\mathcal{LL}$ – **An Object Oriented Library Language Reference Manual,** *Tomasz Kowaltowski, Evandro Bacarin*

93-17 **Metodologias para Conversão de Esquemas em Sistemas de Bancos de Dados Heterogêneos,** *Ronaldo Lopes de Oliveira, Geovane Cayres Magalhães*

93-18 **Rule Application in GIS – a Case Study,** *Claudia Bauzer Medeiros, Geovane Cayres Magalhães*

93-19 **Modelamento, Simulação e Síntese com VHDL,** *Carlos Geraldo Krüger e Mário Lúcio Côrtes*

93-20 **Reflections on Using Statecharts to Capture Human-Computer Interface Behaviour,** *Fábio Nogueira de Lucena e Hans Liesenberg*

93-21 **Applications of Finite Automata in Debugging Natural Language Vocabularies,** *Tomasz Kowaltowski, Cláudio Leonardo Lucchesi e Jorge Stolfi*

93-22 **Minimization of Binary Automata,** *Tomasz Kowaltowski, Cláudio Leonardo Lucchesi e Jorge Stolfi*

93-23 **Rethinking the** DNA **Fragment Assembly Problem,** *João Meidanis*

93-24 EGOLib — **Uma Biblioteca Orientada a Objetos Gráficos,** *Eduardo Aguiar Patrocínio, Pedro Jussieu de Rezende*

93-25 **Compreensão de Algoritmos através de Ambientes Dedicados a Animação,** *Rackel Valadares Amorim, Pedro Jussieu de Rezende*

93-26 GeoLab**: An Environment for Development of Algorithms in Computational Geometry,** *Pedro Jussieu de Rezende, Welson R. Jacometti*

93-27 **A Unified Characterization of Chordal, Interval, Indifference and Other Classes of Graphs,** *João Meidanis*

93-28 **Programming Dialogue Control of User Interfaces Using Statecharts,** *Fábio Nogueira de Lucena e Hans Liesenberg*

93-29 EGOLib – **Manual de Referência,** *Eduardo Aguiar Patrocínio e Pedro Jussieu de Rezende*

Relatórios Técnicos – 1994

94-01 **A Statechart Engine to Support Implementations of Complex Behaviour,** *Fábio Nogueira de Lucena, Hans K. E. Liesenberg*

94-02 **Incorporação do Tempo em um** SGBD **Orientado a Objetos,** *Ângelo Roncalli Alencar Brayner, Claudia Bauzer Medeiros*

94-03 **O Algoritmo KMP através de Autômatos,** *Marcus Vinícius A. Andrade e Cláudio L. Lucchesi*

94-04 **On Edge-Colouring Indifference Graphs,** *Celina M. H. de Figueiredo, João Meidanis, Célia Picinin de Mello*

94-05 **Using Versions in** GIS, *Claudia Bauzer Medeiros and Geneviève Jomier*

94-06 **Times Assíncronos: Uma Nova Técnica para o** Flow Shop Problem, *Hélvio Pereira Peixoto e Pedro Sérgio de Souza*

94-07 **Interfaces Homem-Computador: Uma Primeira Introdução,** *Fábio Nogueira de Lucena e Hans K. E. Liesenberg*

94-08 **Reasoning about another agent through empathy,** *Jacques Wainer*

94-09 **A Prolog morphological analyser for Portuguese,** *Jacques Wainer, Alexandre Farcic*

94-10 **Introdução aos Estadogramas,** *Fábio N. de Lucena, Hans K. E. Liesenberg*

94-11 **Matching Covered Graphs and Subdivisions of K_4 and $\overline{C_6}$,** *Marcelo H. de Carvalho and Cláudio L. Lucchesi*

94-12 **Uma Metodologia de Especificação de Times Assíncronos,** *Hélvio Pereira Peixoto, Pedro Sérgio de Souza*

Relatórios Técnicos – 1995

95-01 **Paradigmas de algoritmos na solução de problemas de busca multidimensional,** *Pedro J. de Rezende, Renato Fileto*

95-02 **Adaptive enumeration of implicit surfaces with affine arithmetic,** *Luiz Henrique de Figueiredo, Jorge Stolfi*

95-03 **W3 no Ensino de Graduação?,** *Hans Liesenberg*

95-04 **A greedy method for edge-colouring odd maximum degree doubly chordal graphs,** *Celina M. H. de Figueiredo, João Meidanis, Célia Picinin de Mello*

95-05 **Protocols for Maintaining Consistency of Replicated Data,** *Ricardo Anido, N. C. Mendonça*

95-06 **Guaranteeing Full Fault Coverage for UIO-Based Methods,** *Ricardo Anido and Ana Cavalli*

Departamento de Ciência da Computação — IMECC
Caixa Postal 6065
Universidade Estadual de Campinas
13081-970 – Campinas – SP
BRASIL

`reltec@dcc.unicamp.br`