The coSARA Design Methodology: Computer Support for Prototyping Collaborative Applications

Yadran Eterovic S.
Computer Science Department
School of Engineering
Catholic University of Chile
e-mail: yadran@ing.puc.cl

Abstract

Building collaborative applications is a difficult task. The coSARA Design Methodology is a collection of design methods, graphical modeling languages, analysis tools, and reusable models and software packages, which can be used to model and build executable prototypes of collaborative applications. It is based on an object-oriented model of collaborative applications and works in two major steps: (1) Constructing formal, graphical models of the different components of an application—the data, software structure, and software behavior of both the application's functionality and user interface--; and (2) Linking the models with reusable software packages and actual user interface objects (e.g., screen buttons, dialog boxes, and drawing windows) from the coSARA Library. The complete structure can then be executed interactively by an interpreter of the software behavior model.

1 Introduction

Building collaborative applications is a difficult task. A collaborative application is a computer application that supports multiple people working together, each on a different workstation, and concurrently sharing the application's data and functionality through the application's interactive multi-user interface. Through the work done on the coSARA System [Mujica, 1991; Wu, 1991; Eterovic, 1992; Tou et al, 1994], we have identified four major tasks that have to be accomplished in order to provide computer support for specifying, building and using collaborative applications:

1Acknowledgments: The coSARA project at the U. of California, Los Angeles, has been funded by several private, state and federal organizations. The author's work at the Catholic University of Chile has been supported by the Chilean Council for Science and Technology, under grant FONDECYT 0778.
1) Conceptualize and define a specific style of sharing, or sharing paradigm;
2) Implement a computer platform—hardware and software—that realizes this sharing paradigm;
3) Define a model of collaborative applications, identifying the applications' components and the interactions among them; and
4) Provide a set of computer tools based on the model that can be used to build collaborative applications that make use of the sharing platform and therefore behave according to the sharing paradigm.

The coSARA System for specifying, building and using collaborative applications has introduced a sharing paradigm called strong sharing, and implemented a sharing platform called object world [Tou et al., 1994]. It has also introduced an object-oriented model of collaborative applications and implemented a methodology for designing them. We describe the model and the design methodology in this paper. High level descriptions of the model and the design methodology are presented in Section 2. Sections 3 and 4 present the details of the model for the two major components of collaborative applications: the application's functionality and user interface. The steps required to transform a model into an executable prototype are described in Section 5. Section 6 provides a complete example of the models and the execution process of a simple collaborative drawing tool. Finally, the major conclusions of this work are presented in Section 7.

2 A Model of Collaborative Applications

The coSARA Design Methodology is a coherent collection of design methods, graphical modeling languages, interactive analysis and interpretation tools, and reusable models and software packages, which can be used to model and build executable prototypes of collaborative applications based on the strong sharing paradigm [Tou et al., 1994]. The coSARA Design Methodology is based on a general object-oriented model of multi-user interfaces for collaborative applications [Eterovic, 1992], which evolved from the coSARA tool model [Mujica, 1991]. We give here only a brief, high level description of this object-oriented model. More detail is introduced in the following sections as we describe the design methodology and present an example.

A collaborative application consists of two basic, interacting components: its functionality and its user interface. Each component is represented by a top-level class in our object-oriented model. These two top-level classes participate in a relation—as in entity-relationship models—representing the fact that the application's functionality and user interface communicate with each other, through different types of messages, to exchange user commands, application responses, and data. Both the application's functionality and user interface deal with data, or objects, of specific data types, or classes. These classes of objects are components of the two top-level classes and participate in several other relations of different types, such as inheritance.
relations, aggregation relations, and usage relations. These relations impose a structure on the software implementing the behavior of the classes. The software structure is specified in terms of hierarchies of communicating software modules; the software behavior is specified in terms of potentially concurrent operations, or methods, and messages, or method invocations.

The coSARA Design Methodology works in two major steps: (1) Constructing formal, graphical models of the different components of an application—the data, software structure, and software behavior of both the application's functionality and user interface—; and (2) Linking the models with reusable software packages and actual user interface objects (e.g., buttons, dialog boxes, and drawing windows) from the coSARA Library. The complete structure can then be executed interactively, on top of the object world sharing platform, by an interpreter of the software behavior model.

3 Modeling the Functionality of an Application

The first step of the coSARA Design Methodology consists of constructing formal graphical models for the data, software structure, and software behavior of the application's functionality and user interface. For this purpose, the methodology provides three different but related formal graphical modeling languages, which we briefly describe below: OREL for data models, SM for structural models, and GMB for behavioral models. Each of these languages is currently supported by a specialized graphical editor.

The class diagram of a collaborative application is a description of the collection of classes, their relationships and their manipulation methods needed to support the operation of the application's functionality and user interface. OREL, a graphical object-oriented data modeling language, which incorporates relations—as in entity-relationship models—provides a set of modeling primitives to specify simple and composite classes, class attributes, and relations among classes such as class inheritance and class usage, as shown in Fig. 1 [Mujica, 1991]. The coSARA System's OREL compiler translates a class diagram into the appropriate class definitions and the standard functions for creating class instances, assigning values to instances' attributes, adding or removing the component objects of a composite object, etc. The compiler produces CLOS (Common Lisp Object System) code that is stored in the coSARA Library.

The structural model of a collaborative application is a representation of the hierarchy of, and high level interactions among, the software components implementing the application's functionality and user interface. The language SM provides three modeling primitives to describe such hierarchies and interactions: modules represent the application's software components, usually in a hierarchical arrangement in which composite modules contain component modules; sockets represent the modules' communication ports or interfaces, allowing a module to request services from and to offer services to other modules; and connections represent explicit communication paths between the
Cl and C2 are composite classes (rounded-corner rectangles). C1, a subclass of class C0 (not shown), consists of classes C2 and RC1, and relation R1. C2 consists of classes S1, S2 and S3, and relation R2.

RC1 is a recursive composite class (rounded-corner rectangle with shadow). It is a subclass of class RC0 (not shown) and consists of classes RC1 (itself) and S4.

S1, S2 and S3 are simple classes (rectangles) without attributes, that is, representing classes already defined.

S4 is a simple class with 2 attributes: Attr1 and Attr2 (shaded rectangles).

R1 and R2 are relations (shaded diamonds). R1 relates 2 classes: C2 and RC1. R2 relates 3 classes: S1, S2 and S3.

Inheritance is represented by enclosing the name of the superclass or superclasses in parentheses.

---

**Figure 1:**
The primitives of OREL and an example of their use.

**Figure 2:**
The primitives of GMB and an example of their use.
modules' sockets [Estrin et al., 1986]. Of course, there is a close correspondence between a collaborative application's class diagram and its structural model.

The behavioral model of a collaborative application is a representation of the activities encapsulated by the application's software components, or SM modules. Each SM module has a behavioral model. The behavioral model for a composite module is the composition of the behavioral models of its component modules [Straub and Espinoza, 1994]. The behavior of a module is specified in terms of GMB, a language that models two related aspects of the high level, concurrent behavior of an application—flow of control and flow of data—and includes detailed, executable descriptions of the data flow's entities, as shown in Fig. 2 [Estrin et al., 1986]:

- A **control flow** graph models the flow of control among the events (the graph's nodes) that occur in the application, similar to a Petri net; the partial ordering of the occurrence of the events is determined by logical constraints among directed control arcs connecting the events.

- A **data flow** graph models the flow of data between program segments (processes) and data stores (datasets); the type of access of the processes over the datasets—read only, write only, or both read and write—is determined by directed data arcs connecting them. There is a one-to-one correspondence between the nodes in the control flow graph and the processes in the data flow graph.

- An **interpretation** describes the data types and values stored in the datasets, and the low level, sequential algorithms executed by the processes.

The GMB model of each functional and user interface module representing a class contains datasets for storing all the instances of the class, and processes representing the functions that can be applied to those instances. The meaning of a GMB model is formally defined by an interactive interpreter, the coSARA System's token machine, as explained in Sections 5 and 6.

It is important to mention here that class behavior and the behavior of user interfaces have been modeled before using formal, graphical notations based on extended state transition diagrams. For example, the work by Booch [1994], Interaction Objects [Jacobs, 1986], Objectcharts [Coleman et al., 1992] and Statemaster [Wellner, 1989]. Instead, we have chosen to use the SM and GMB modeling languages, because they allow us to construct test environments using the same notations [Estrin et al., 1986] and to express more naturally the concurrent aspects of collaborative applications [Eterovic, 1992], and because it is possible to automatically synthesize SM and GMB models from the semiformal statement of application's requirements [Lor and Berry, 1991].
4 Modeling the Multi-User Interface of an Application

As we said, the data, software structure, and software behavior of an application's multi-user interface is modeled using the same three graphical modeling languages described in Section 3 (OREL, SM and GMB). However, it should be noted that in our object-oriented model an interface specification is conceptually based on three types of abstractions: interactors, contacts, and dialogs [Eterovic, 1992]. Interactors represent usual interface objects that behave according to the direct manipulation, point-and-click paradigm of user interfaces, such as buttons, scroll bars, text fields, or drawing windows. Each interactor consists of one contact connected to several dialogs.

The interactor's contact models the set of user actions that an interface object responds to (e.g., user inputs such as pressing a key on the keyboard, clicking or moving the mouse, etc.), and the set of application responses that the object produces due to requests received from the application's functionality (e.g., screen outputs such as highlighting a screen region, drawing or erasing a figure, etc.).

Each interactor's dialog, connected to the contact, models the syntax of a particular type of interaction between the user and the application's functionality. It specifies which sequences of user inputs received through the contact are valid and the points in these sequences at which information is sent back to the contact for presentation to the users. Each dialog describes one valid sequence of inputs. For example, if the interactor represents a drawing window, then the contact may communicate with one dialog that handles rectangles and another that handles polylines; the rectangle dialog accepts sequences of three input events: pressing the mouse's button, dragging the mouse, and releasing the button; the polyline dialog accepts sequences of any number of mouse button clicks ended with a double-click.

Interactors, contacts and dialogs are structurally represented by SM modules, which communicate via sockets and connections, and their behavior is described using GMB models. It is important to note that, while application and interface designers can define their own interactors, most standard contacts and dialogs have already been encapsulated as SM+GMB modules and are available for use from the coSARA Library. These include click-on and switch screen buttons, text fields, interface object highlighters, box and point-list definitions, box-based and point-based data selectors, and data repositioning and reshaping definitions.

5 Transforming Graphical Models into Executable Prototypes

After building the models of the application (its data, software structure and software behavior), the second step of the coSARA Design Methodology consists of linking the models with reusable software packages and actual user interface objects, to make the models executable. This step involves:
1) Associating executable interpretation with the behavioral models;
2) Instantiating the objects of the application's functionality and user interface; and
3) Linking these objects to the models to provide them with behavior.

First, GMB models have to be completed with the interpretation associated with their data flow graphs. The interpretation software packages—i.e., the executable code defining the algorithms of the processes and the data types and values of the datasets—reside in the coSARA Library. They can be associated with the models using the GMB graphical editor. This editor requests the packages' names from the designer, looks them up in the coSARA Library, and associates them with the corresponding processes and datasets. The coSARA Library contains a collection of reusable general purpose interpretation software packages, as well as software packages generated by the OREL compiler from the class diagrams. Of course, any application specific package which is not yet part of the Library and is not generated automatically by the OREL compiler has to be coded by the designer and stored in the Library before associating it with the models.

At this point, the set of models represent a complete formal specification of the application, which can be analyzed using the coSARA System's analysis tools. The control flow analyzer can be used to analyze control flow graphs for pathologies of the form of deadlocks and potentially infinite states. The performance analyzer, which uses a queuing model, can be used to evaluate the possible performance of the application with multiple users.

Next, an executable application object can be created by instantiating the application's functionality and user interface classes, thus producing the corresponding objects, and then linking these objects to the models and therefore providing them with executable behavior.

As we said in Section 3, the code defining the classes, including their behavior attributes, and the functions to instantiate them is produced by the OREL compiler from the application's class diagram, both for the application's functionality and for its user interface. Thus, the instantiations are simply accomplished by invoking the corresponding functions; and linking the resulting objects to the behavioral models is simply a matter of assigning the models as the values of the objects' behavior attributes.

But before the application's functionality object can be actually executed by users, the application's user interface objects have to be linked to actual screen windows. These screen windows, which include only presentation aspects but no behavior, are first produced by instantiating appropriate classes stored in the coSARA Library. Then, they are made sensitive to the user actions specified in the interactors models by a process which takes into account the particular software system actually used to manage the windows. The coSARA System runs CLUE (Common Lisp User Interface Environment) on top of CLOS and CLX (the Common Lisp interface to the X window system). Now the
application object can be executed by the coSARA System's token machine, which is actually driven by users interactively operating real input devices on real screen windows.

The coSARA System's token machine is an interactive interpreter of GMB models. Each node in a GMB model's control flow graph has an input logic and an output logic and is associated with one process of the data flow graph. The input logic defines the required distribution of tokens in the node's input control arcs for the node to fire. Firing a node causes the tokens on the input control arcs to be removed and the interpretation of the process associated with the node to execute. When the interpretation finishes its execution, tokens are placed on the node's output control arcs according to its output logic.

6 Using the coSARA Design Methodology

We illustrate the coSARA Design Methodology by presenting the steps taken in the design and execution of the models of a simple collaborative drawing tool. First, we briefly describe the intended way in which the tool should be operated by a user. Then, we show the class diagram and the structural and behavioral models of part of the tool. Finally, we explain the execution by the coSARA System's token machine of a command given by a user to the tool's executable prototype, built from the models according to the instantiation and linking steps described in Section 5.

The drawing tool allows multiple users to concurrently draw and move rectangular blocks on a window. Fig. 4 shows a sample window running the tool. A block is drawn by pressing the mouse's left button at the block's upper lefthand corner, moving the mouse to the block's bottom righthand corner, and releasing the button. A block is moved by pressing the mouse's right button while pointing the mouse's cursor to a block's edge, dragging the block with the mouse to a new location, and releasing the button.

Fig. 5 shows the class diagram for the drawing tool. The tool is represented as a top-level composite class (UI), containing four classes and two relations. The tool's functionality and user interface are represented as composite classes Tool and Interactor, both participating in relations Commands and Calls together with classes Command and Call. The type of the figures handled by the tool becomes a component class of Tool (Block); and the figures' properties become attributes of this class (TopLeft and BtmRight). Relations Commands and Calls indicate that each instance of the tool's functionality uses an instance of the tool's user interface to communicate with the users through messages of types Command and Call. Classes Contact and Dialog are component of class Interactor and participate in relations Actions and Responses together with classes Action and Response; the relations indicate that an interactor's contact communicate with the interactor's dialog through messages of types Action and Response.
Figure 4: The drawing tool during operation.

Figure 5: The OREL class diagram of the drawing tool

Figure 6: The structural model of the drawing tool
The structural model of the drawing tool is shown in Fig. 6. Starting with a top-level module with no sockets (UI), the tool's software structure is hierarchically decomposed by refining modules into submodules, until the behavior of each module is precise enough to be represented by a single GMB model (e.g., I/O, DoRect and Make). Module UI contains submodules Tool and Interactor, representing the tool's functionality and user interface, in agreement with the tool's class diagram.

As examples of behavioral models, the GMB models for Make and I/O are shown in Figs. 7(b) and 7(d). Make includes the necessary behavior to create instances of blocks (process MakeFig) and to store them (dataset FigLst). I/O represents the window where users perform all the actions and observe the responses to those actions. User actions result in tokens placed on node N1's output control arcs: LDn, LUp, RDn and RUp. The responses are the result of the activation of processes DrawFig and EraseFig to draw and erase rectangles.

We describe now the activities that take place in the tool model when a user draws a rectangle on the tool's window, defining a block. To draw a rectangle on the window, a user pushes down the mouse's left button, moves the mouse, and releases the button. When the user pushes down the button and then when the button is released, process GetAction in I/O (Fig. 7(d)) sends the action's window position through data arc Pos, and causes a token to be placed on one of N1's output control arcs: LDn when the button is pushed down and LUp when it is released.

The tokens and data thus produced travel to dialog DoRect, shown in Fig. 7(e), which requires a sequence of two actions to produce a rectangle: the sequential activation of processes Start and Finish, through firing of nodes N1 and N2. After the second action is received through input control arc LUp, the positions of both actions are stored in dataset Rect and a token is placed on output control arc DataOK, thus informing module Make that a new rectangle has been defined. A token is also placed on output control arc Draw, informing I/O that the rectangle was properly defined, so that I/O can draw it on the window by the activation of process DrawFig.

In module Make, shown in Fig. 7(b), the token on arc DataOK eventually activates process MakeFig, whose interpretation, shown in Fig. 7(c), calls CLOS methods produced by the OREL compiler and stored in the coSARA Library. First, it creates a new block (i.e., a new instance of class Block); then it gets the values of the attributes of this new block, through arc FigData; and finally it stores the new block by adding it to dataset FigLst.

7 Conclusions

This work has introduced the coSARA Design Methodology for the specification, construction and execution of collaborative application prototypes. The methodology is fully supported by computer tools and is based on an object-oriented model of collaborative applications, three graphical modeling languages, an instantiation and linking process, and an interactive
Figure 7: The models of the drawing tool: (a) Sketch of SM model; (b) GMB model of Make; (c) Interpretation for processor MakeFig; (d) GMB model of I/O; (e) GMB model of DoRect.
interpreter. The object-oriented model treats uniformly the functionality and the multi-user interface of collaborative applications. The modeling languages capture the structure, the interactions and the potentially concurrent behavior of the software components of collaborative applications. The instantiation and linking process promotes and supports the reusability of models and software packages. The interactive interpreter allows designers and users of collaborative applications to more effectively evaluate requirements, designs and implementations. Using the methodology we have been able to construct the collaborative drawing tool presented in Section 6, an extended version of this tool [Tou et al., 1994], and a collaborative zoom and scroll tool that operates on graphical representations [Eterovic, 1992].

References