Unifying the Design and Implementation of User Interfaces through the Object Paradigm

Véronique Normand

Bull-IMAG Systèmes
2, rue de Vignate
38610 Gières, FRANCE
normand@imag.imag.fr

Joëlle Coutaz

Laboratoire de Génie Informatique
BP 53X
38041 Grenoble Cedex, FRANCE
joelle@imag.imag.fr

Abstract. This paper presents SIROCO, a research on user interface design and development support in an object-oriented programming environment. User interface (UI) design in SIROCO focuses on conceptual modeling; a fully object-oriented conceptual representation model and language is proposed, distinguishing the function dimension from the use dimension in an interactive system. Interaction style and presentation aspects are dealt with as generic parameters. SIROCO promotes a continuous object-oriented approach from UI design to UI implementation. Transition from conceptual design to implementation is achieved through an extended object-oriented software architecture model. Effective UI development is supported by an object factory that produces the object code needed to implement the specified user interface.

1 Introduction

Object-oriented concepts have been used in human-computer interaction for a long time now, pervading domains from user interface (UI) modeling methods to implementation tools. Decomposition and encapsulation of a behavior behind an external interface are basic object-oriented concepts that may be recognized in a number of contributions, for example: event handlers for dialogue description (e.g., [Hill 86]), or multi-agent UI system architectures (e.g., PAC [Coutaz 90]). Class inheritance is an additional concept that appears in a subset of more explicitly object-oriented proposals: MAD [Pierret-Golbreich 89] in task modeling, UIDE [Foley 88] in functional design, the GWUIMS [Sibert 86], Smalltalk MVC [Goldberg 84], programming tools such as graphical toolkits (e.g., Motif [OSF 89]) and application frameworks (e.g., MacApp [Schmucker 86]), to name a few.

The object paradigm offers a natural and powerful modeling scheme for user interfaces, which may be further exploited. This paper elaborates on one simple observation: an object may be anything from an abstract concept to a concrete implementation module. Thanks to this width in scope, the object paradigm can be viewed as a unifying concept bridging the gap between the UI design and implementation tasks.

Unifying system analysis and implementation is a concern underlying some recent works in object-oriented software design. General object-oriented analysis methods are being devised ([de Champeaux 91], particularly [Jacobson 87], ...), simplifying the transition from object-oriented analysis to object-oriented implementation; this transition cannot however be thoroughly explicated and supported in a general context, due to the

amount and the domain specificity of the factors influencing the implementation of a system. Our feeling is that user interface is a specific enough domain to allow an effective support of this transition in a given development environment.

This paper presents SIROCO, a research on UI design and development support in the GUIDE (Grenoble Universities Integrated Distributed Environment) object-oriented programming environment [Krakowiak 90]. Target GUIDE applications are data-oriented rather than processing-oriented; typical applications are office applications (e.g., an electronic agenda system), a document circulation system, etc. Target interfaces are relatively simple form- or icon-based interfaces, with an interaction style mixing keyboard entry and mouse direct designation.

User interface design in SIROCO focuses on conceptual modeling; a fully object-oriented conceptual representation model is proposed, along with a dedicated specification language. Transition from conceptual design to implementation is achieved through an extended object-oriented software architecture model. Effective UI development is supported by an object factory that produces the object code needed to implement the specified user interface.

After a rapid introduction of the notion of conceptual modeling, the SIROCO conceptual representation model and language are presented. Interaction style and user interface presentation design is then addressed. Finally, the transition from conceptual objects to implementation objects is explicated. Related work is rapidly discussed before concluding.

2 Modeling the User Interface: Focusing on the Concepts Behind the Image

The main assumption of our work is that a user interface design can be decomposed into conceptual choices, interaction style choices and presentation choices. SIROCO addresses all three aspects of UI design, focusing on conceptual modeling as the key input to a UI development tool.

2.1 User Interface Conceptual Modeling

Conceptual modeling aims at defining the designer's abstract model of the interactive system, that is the semantic model to be transmitted to the user through the interface. Just what is in this abstract model? Our approach is to distinguish two dimensions in an interactive system: the function dimension and the use dimension, emphasizing a duality underlying numerous works such as [Moran 81], [Green 85] and [Barthet 88].

The *function dimension* describes the entities and operations that constitute the functionality of the system.

The *use dimension* describes the way the functional entities and operations relate to the user tasks. It is based on a small set of specific concepts, mainly the notions of user task, operating image, and operating procedure. A *user task* defines a goal that the user will achieve with the system. Closely related to the notion of task, the concept of *operating image* denotes the user's deforming view on an entity: an entity used in the accomplishment of a task undergoes a deformation emphasizing elements relevant to the task while dismissing non relevant elements. The user accomplishes a task with a system through the execution of an *operating procedure* defined as a sequence of system commands.

2.2 Capturing Design Data

User interface design is a difficult task characterized by a large set of volatile often implicit information. Our aim is to provide a formalism helping to collect a maximum of data from the designer - data that most of the time are lost to User Interface Development System tools for they remain implicit and non formalized.

Design data can be of three types: structural, semantic and logical. Structural data describe organizational aspects of a UI, and constitute the core of the conceptual model; semantic and logical data provide complementary information attached to structure elements. For each type of data, generic aspects must be isolated from specific aspects.

Objects provide a general method to express the structural aspects of our UI conceptual model. As to semantic or logical data representation, objects are of no help; we introduce the notion of *property* as an extension to the object-oriented model in order to capture logical and semantic data. A property is a valued label; the label identifies the type of information contained as the value of the property - possible types are text, number, boolean, but also function.

The next section presents the SIROCO conceptual model, highlighting the structural, semantic, and logical data representation features of the model.

3 The SIROCO Conceptual Model

The distinction between function and use dimensions in the structure of a UI is captured in a two-fold model offering on the one hand the notion of Application Concept, and on the other hand the notions of Perspective and Workspace.

The remaining part of this section presents the different elements of our model. The model is illustrated by a popular but not trivial example: an electronic mail system.

3.1 Function Dimension: Application Concepts

An Application Concept represents an entity of the application the semantics of which lie within the task domain or the interface metaphor. For example, Application Concepts of an electronic mail system may be the concepts of user, mailbox, and message. They may also include the concepts of trash bin, post-office, etc. depending on the design choices.

An Application Concept is specified as an object offering a set of data attributes and a set of operations. Data attributes express the system data model; we distinguish simple attributes (with simple data types: integer, string, date, etc.) and complex attributes (with types referencing an Application Concept definition). This simple object model is enriched with a set of properties allowing the collection of complementary information on semantic, logical and structural aspects of an Application Concept.

Example. We define four Application Concepts for the mail application: user, mailbox, message and received message. The latter concept is defined in order to express some functional constraints on the concept of message.

User. A user has a name and possesses a mailbox. It is modelled as the Application Concept User_AC in the following SIROCO specification[1]:

```
CONCEPT User_AC IS
        // properties
    _name: "user"                       // semantic property:
                                        // natural language identification

        // attributes
    ATTRIBUTE Name : STRING             // simple attribute
        _name: "name"
        _kind: KEY;                     // semantic property: key attribute
        _size: 12                       // structural property: size
    ATTRIBUTE Mbox : MailBox_AC         // complex attribute
        _name: "mailbox"
END User_AC.
```

Mailbox. A mailbox is a simple list of received messages. It is modelled as the Application Concept MailBox_AC:

```
CONCEPT MailBox_AC IS
        // properties
    _name: "mail box"                   // semantic property:
                                        // natural language identification

        // attributes
                                        // complex attribute
    ATTRIBUTE Messages : LIST OF RecMessage_AC
END MailBox_AC.
```

Message. A message has a sender, a reception time, a subject and a text; sender and reception time are two data items that cannot be overwritten. Moreover, a message may either be deleted or posted. It is modelled as Message_AC in the SIROCO description:

```
CONCEPT Message_AC IS
    _name : "message"                   // identification
        // attributes
                                        // read-only complex attribute
    R_ATTRIBUTE Sender : User_AC
        _name : "sender"
        _kind : KEY;
                                        // read-only simple attribute
    R_ATTRIBUTE Time : Time
        _name : "reception date"
                                        // read-write simple attribute
    ATTRIBUTE Subject : STRING
        _name : "subject"
        _kind : KEY;
        _size : 40
    ATTRIBUTE Text : STRING
    ...
        // operations
    OPERATION Delete
        _name : "destruction"
        _kind : DESTROY;
                                        // operation with one parameter
    OPERATION Post
        _name : "send the message"
        IN Whom : STRING                // input parameter
            _name : "receiver"
```

1. The SIROCO descriptions appearing in this paper are minimal; in particular, property lists are limited to a subset of elements.

So as to clarify the language syntax, key words as well as pre-defined property values appear in capitals. Property labels are set in lowercase letters, beginning with the '_' character.

```
        ...
END Message_AC.
```

Received message. A received message is a message the subject and text of which cannot be modified; the Post operation cannot be called on a received message. Hence the SIROCO description:

```
CONCEPT RecMessage_AC RESTRICTS Message_AC IS
      _name : "received message"
            // attributes
      R_ATTRIBUTE Sender
      R_ATTRIBUTE Time
      R_ATTRIBUTE Subject
      R_ATTRIBUTE Text
            // operations
      OPERATION Delete
END RecMessage_AC.
```

Property sets. A property set is attached to each Application Concept as well as to each element composing the Application Concept (operation, parameter, data attribute). We distinguish intrinsic properties from contextual properties. *Intrinsic properties* have values bound to the Application Concept while values of *contextual properties* may change depending on the context within which the Application Concept is being used, as will be seen later on in section 3.2.

Collecting Semantic Data. Semantic information is collected through a list of properties present in any property set. Generic semantic information is specified in `kind`, a multi-valued intrinsic property. Examples of generic semantic information are "key" for a data attribute (with the data base meaning), or "destroy" for an operation (meaning that the operation execution modifies the system state in a definitive way).

Application specific semantic information is gathered through the double mechanism of contextual and intrinsic properties. On the one hand, natural language explanations are collected in a set of contextual identification properties (textual name and description of the Application Concept). On the other hand, the intrinsic multi-valued `semantic-traits` property allows the formalization and collection of application semantics in the form of designer-defined identifiers.

Collecting Logical and Structural Data. Logical information is collected through a set of properties depending on the type of conceptual element considered.

- Data attributes have properties to specify constraints on their values: an intrinsic `size` property (for strings or sets), an `initial-value` and a `value-range` property.
- Operations have properties allowing the description of constraints on their activation: intrinsic `pre-condition`, `post-action`, `parameter-validation` condition properties.
- Properties of an operation parameter are identical to those of a data attribute, with the addition of a contextual `link` property for result parameters, so as to specify how returned values are to be used.

Relations Between Application Concepts. Relations between two Application Concepts AC1 and AC2 may thus be structural, semantic and logical.

- AC1 has a data attribute defined as an AC2; AC1 and AC2 have structural relations. In addition to this referencing relation, real composition structuring relations are under study.
- Several possibilities may coexist for semantic relationships : AC1 and AC2 may have common `semantic-traits` or common `kind` values. An IS-A relation may also be expressed between AC1 and AC2: AC1 IS-A AC2 means that AC1 is defined as a specialization of AC2 through inheritance mechanisms (only single inheritance is considered for the moment).
- Logical relations between AC1 and AC2 instances are expressed through properties such as the `initial-value` attribute property and constraint operation properties.

3.2 Use Dimension : Workspaces and Perspectives

Workspaces and Perspectives are elements of the model dedicated to the organization of the data and operation space defined by Application Concepts according to the system use dimension. Figure 1 uses a spatial metaphor to illustrate the notions of a Workspace and a Perspective relatively to an Application Concept.

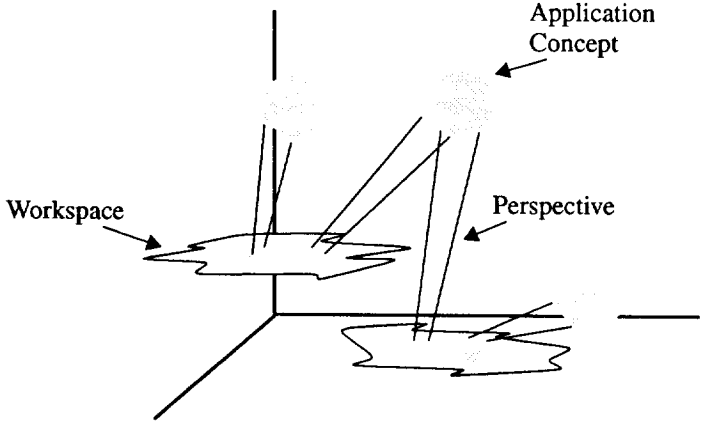

Figure 1: Organizing the Application Concept space through Workspaces and Perspectives.

A Workspace is a partitioning unit of the user interface; a Workspace represents a conceptual "place" (in the sense of the "room" concept in [Henderson 86]) where a precise set of tasks may be accomplished by the user. A Workspace gives access to a set of Application Concepts - the Application Concepts required to perform the given user tasks. This access is refined and constrained by a set of Perspectives defined on the Application Concepts. Depending on his/her goals, the user's activity may span several Workspaces; possible paths for navigating among Workspaces are defined as Workspace Operations.

A Perspective defines a point of view on an Application Concept through a filter on the attributes and operations of the concept. A Perspective defines what may be seen as an operating image on the Application Concept, highlighting the attributes and operations potentially useful in the procedures solving the tasks in a Workspace, and hiding the elements irrelevant to these tasks. Several Perspectives may be defined on a given

Application Concept; a particular one is the full Perspective (i.e., no filtering).

In terms of the object paradigm, a Workspace is defined as an object the data part of which is a set of Application Concepts, and the operation part of which is a set of navigation operations. A Perspective is defined as a restriction on the type of an Application Concept object; Perspectives are elements embodying the polymorphism of Application Concept objects.

Example. Several combinations of Workspaces may be defined for the mail application. Starting from the mail task tree (not represented here), we propose to organize the use dimension of the system into five Workspaces:

- the Connection Workspace, where the user connects to the system, giving his/her name.
- the Handling Workspace, main Workspace of the application, where the user has a global view on his/her mailbox, and may delete messages.
- the Read and Reply Workspaces, where the user reads and replies to a received message.
- the Write Workspace, where the user composes a message from scratch, and posts it.

Logical sequencing relations between these five Workspaces are illustrated in figure 2, expliciting the possible navigation paths from one Workspace to another.

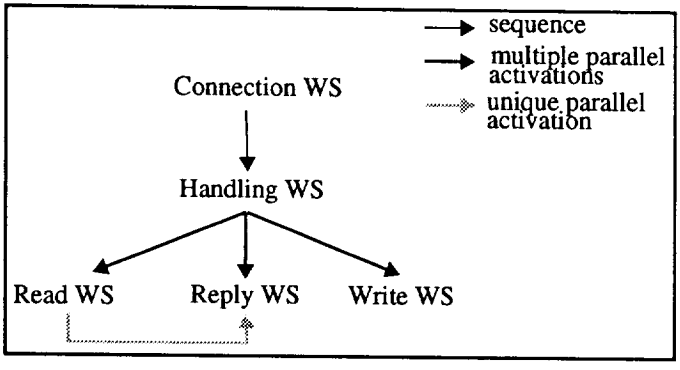

Figure 2: Workspaces of the mail system: sequencing relations.

We will only explicit the Handling and Read Workspaces.

Handling Workspace. This Workspace is defined as the main Workspace of the mail application, articulation point between the different user activities. It offers a global Perspective on the mailbox of the user; the only effective task that may be accomplished in this Workspace is the deletion of a message. The Handling Workspace offers a set of operations allowing the "access" to three other Workspaces: Read, Reply and Write Workspaces.

```
WORKSPACE Handling_WS IS
      _name : "handling the mailbox"
      _kind : MAIN;                    // semantic property: main workspace

            // data presented in the Workspace
                  // Application Concept and its associated Perspective
```

```
user : User_AC   WITH PERSPECTIVE Global_P

      // navigation operations
            // access to the Read Workspace
WS_ACCESS Read TO Read_WS
      _name : "read a message"
                  // logical sequencing property: Read_WS will run
                  // in parallel with the current Workspace
      _relation : PARALLEL;
                  // logical property: transmission of data
                  // from current to Read_WS
      _data_transmission :r_mess:= mess;
      IN mess : RecMessage_AC        // input parameter
            _name : "message to read"

            // access to the Reply Workspace
WS_ACCESS Reply TO Reply_WS
      name : "reply to a message"
      _relation : PARALLEL;
      _data_transmission :r_mess:= mess;
      IN mess : RecMessage_AC
            _name : "message to reply to"

            // access to the Write Workspace
WS_ACCESS Write TO Write_WS
      _name : "write and send a message"
      _relation : PARALLEL;
END Handling_WS.
```

Dynamics of the Handling Workspace, as expressed by the description above, imply that the user may:

- exit this Workspace, which terminates the application (as implied by the "main" value of the `kind` property).
- delete a message (operation Delete defined in perspective Global_P, as represented below).
- run a new Workspace: Read_WS, Reply_WS or Write_WS (as implied by `ws_access` operations). New Workspaces are then instanciated and run as new parallel activities of the user (as the value of the `relation` property implies).

Perspective Global_P. The definition of a Perspective recursively associates a Perspective to each of its complex data attributes (i.e., data attributes referencing an Application Concept). Perspective Global_P allows a global visualization of the list of the messages received in the mailbox of the user; an identifying header is presented for each message; the deletion operation is moreover allowed.

```
PERSPECTIVE Global_P ON User_AC IS
      R_ATTRIBUTE Name
            // recursive specification of a perspective for complex attribute Mbox
      ATTRIBUTE Mbox
            WITH PERSPECTIVE Mbox_P IS
                  // specification of a perspective for complex attribute Messages
                  ...PERSPECTIVE Header_P...
             END Mbox_P
END Global_P.

            // perspective on a received message
PERSPECTIVE Header_P ON RecMessage_AC IS
      R_ATTRIBUTE Sender WITH PERSPECTIVE Sender_P
      R_ATTRIBUTE Subject
      R_ATTRIBUTE Time
      OPERATION Delete
END Header_P.
```

```
            // perspective on the sender of a received message
PERSPECTIVE Sender_P ON User_AC IS
      R_ATTRIBUTE Name          // filtering out everything but the Name attribute
END Sender_P.
```

Read Workspace. This is the virtual "place" where the user can read a message. This Workspace offers a reading Perspective on a received message Application Concept; a navigation operation is defined as a short-cut to the Reply Workspace.

```
WORKSPACE Read_WS IS
      _name : "reading a message"
      _instances : MULTIPLE;          // logical property: this workspace may be
                                      // multiply instanciated
      _unique_idem? : TRUE;           // logical property: instances of this
                                      // workspace may not present the same data.

            // data presented in the Workspace
      IN r_mess : RecMessage_AC WITH PERSPECTIVE Read_P

            // navigation operations
                        // access to the Reply Workspace
      WS_ACCESS Reply TO Reply_WS
            _name : "reply to a message"
            _relation : PARALLEL;
            _data_transmission :r_mess:= r_mess;
END Read_WS.
```

The properties specified for the Read Workspace allow the activation of multiple different co-existing instances of this Workspace, as implied by the values defined for the `instances` and `unique_idem?` properties.

Perspective Read_P. This perspective allows a detailed view on the contents of a received message.

```
PERSPECTIVE Read_P ON RecMessage_AC IS
      R_ATTRIBUTE Sender WITH PERSPECTIVE Sender_P
      R_ATTRIBUTE Subject
      R_ATTRIBUTE Time
      R_ATTRIBUTE Text
END Read_P.
```

Property sets. In order to collect semantic and logical information, property sets are associated to each of the use dimension elements. The property sets associated to a Perspective and its sub-elements (data attributes, operations and operation parameters) have the peculiarity to allow the overloading of the contextual properties defined for the corresponding elements of the underlying Application Concept. These property sets also contain properties that are local to the Perspective.

Collecting Semantic Data. Semantic properties presented in 3.2 are also defined in the property sets of the use dimension elements. Examples of generic semantic information include:

- "main" or "exit gate" for the Workspace `kind` property value to specify the role or potential role of the Workspace within the application interface.
- "prototype" for a Perspective `kind` property value, meaning that the presented object is not an existing instance but a prototypic template.

Collecting Logical Data. Logical information is collected through a set of properties depending on the type of element considered.

Workspaces have properties to specify constraints on their instanciation: an `instance` property tells whether instanciation is to be unique or multiple at execution time (i.e., whether several instances may be activated by the user); a `unique_idem` property expresses whether identical instances (i.e., presenting the same data) are proscribed or not.

Workspace operations have properties identical to those presented for Application Concepts Operations, with the addition of properties describing the relations between the Workspaces that are the source and the destination of the navigation: a `relation` property to specify sequence, parallelism, suspension, etc.; a `data_transmission` property to describe optional transmissions of data from the source Workspace to the destination Workspace.

Perspectives have logical properties identical to those defined for Application Concepts in 3.1. In particular, `pre-condition` and `post-action` properties on operations allow for the definition of part of the UI dialogue dynamics, in the context of the Workspace where the Perspective is being used.

4 Interaction Style and Presentation Design Choices

As previously explained, SIROCO considers interaction style aspects as well as presentation design independently from conceptual modeling. The SIROCO conceptual modeling formalism allows for the description of structural, semantic and logical aspects of the UI contents; so as to complete the design of the UI, interaction details must be considered, and presentation choices must be specified.

SIROCO addresses both tasks in a simplifying generic manner.

4.1 Interaction Style Design

Interaction style design is mainly concerned with the design of command activation syntax. Commands of the interactive system are the Application Concepts Operations and Workspace navigation operations that appear in the description of the Workspaces and Perspectives of the conceptual model, as seen in section 3.2.

Command activation involves the following actions: choice of the command, choice of the object on which the command is to be applied, input of values for the parameters of the command, if any.

SIROCO defines general interaction style features as common features for systems developed in the GUIDE environment; in addition to these fixed features, SIROCO defines several possible command activation styles, limiting design choices to positioning interaction style parameters.

Common Features.

- The choice of the object of a command is only required if this command is ambiguous (i.e., the command may be called on several different objects presented in the interface).
- The choice of a command, of an object to this command, or the input of a complex parameter (i.e., a reference to an Application Concept) are actions realized through direct designation of displayed interface elements.

- Input of simple parameters is realized through forms.

Interaction Style Parameters.

- Sequencing constraints on command activation actions: activation style may be pre-fixed (object choice before command choice), post-fixed, or unconstrained.

4.2 Presentation Design

Presentation design is concerned with the choice and layout of graphical representations for the contents of a UI. Presentation may be decomposed into the presentation of commands and the presentation of data.

As with interaction style choices, SIROCO defines general rules for GUIDE applications interfaces, as well as presentation style parameters.

General Rules.

- The Motif style recommendations are applied. Commands are presented in menu bars while data are presented in work areas of windows.
- A set of precise rules was defined so as to map conceptual model elements onto graphical representations. The presentation of these rules is out of the scope of this paper (cf [Normand 92]). To give the reader a flavor of the interfaces produced with these rules, figure 3 shows the SIROCO presentation of the Handling and Read Workspaces specified as examples in section 3.

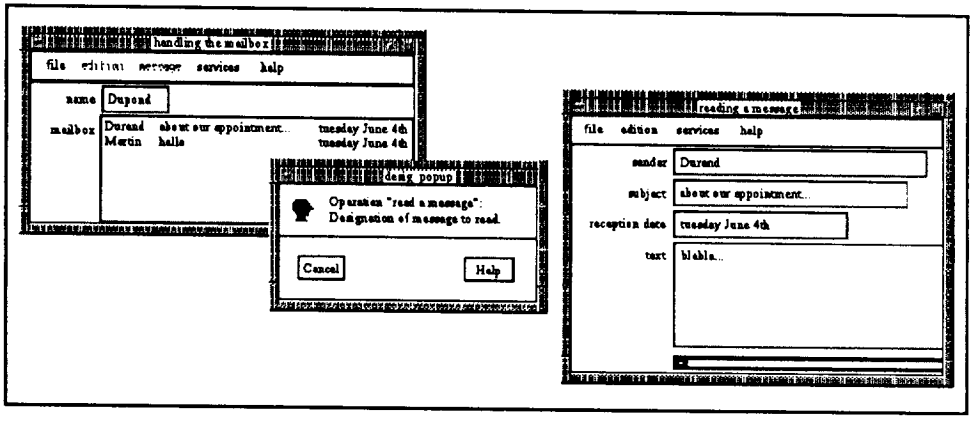

Figure 3: Presentation image for the Handling and Read Workspaces of the e-mail system.

Each Workspace fits in one independent window. The identification strings are those specified in the name properties. Command menu "services" contains the navigation operations defined on a given Workspace. Command menus "edition" and "help" are not part of the specification: these are facilities implicitely added and managed by SIROCO (resp. cut-and-paste editing facilities, and a static help system based on the natural language identification properties).

Presentation Style Parameters. These parameters only deal with the presentation of commands in the menu bars of the interface windows. The main question regarding menu

bars organization is whether or not the object structure defined in the conceptual model must be perceptible in the interface. Three presentation styles are defined:

- per-concept: the object structure is preserved, and menus are organized according to the data specified in Workspaces. Each Application Concept presented in a Workspace with a Perspective containing operations implies the creation of a menu the items of which present the operations as commands of the interface; complex Application Concept attributes may lead to the creation of sub-menus.
- per-perspective: the object structure is preserved, and menu organization focuses on Perspectives appearing in a Workspace.
- per-trait: menu bars organization does not respect the object structure, but focuses on the values defined for the `semantic_traits` property of the operations.

The interface presented in figure 3 corresponds to a per-concept organization of commands.

5 From Conceptual Objects to Implementation Objects

Translating conceptual objects into implementation objects is the actual task of the developer. This translation process is influenced by several general factors, mainly:

- the quality of developed code: the search for maintainability.
- the limitations of the programming and executing environment: the search for good performances.
- the implementation tools at hand.

SIROCO provides a double answer to UI development support: firstly, the SIROCO specification model has extensions into the architecture design domain; secondly, the SIROCO language can be used as input to an object factory providing full prototyping and developing support.

5.1 Using a SIROCO Specification as the Rationale for Architectural Design

The use of an architecture model in the development of an interactive system is now established; an architecture model serves as a guide simplifying the developer's task and ensuring a robust code structure allowing the management of change during the life cycle of the system.

Regarding the nature of the architecture to be used, a consensus seems to be reached in the user interface development community, as some recent workshops demonstrate in [Lisbon 90] and [Arch 91]. The basic prevailing principles stem from the Seeheim model [Pfaff 85] (see figure 4): the separation of the interactive system into a Functional Core and a user interface component; the separation of the user interface component into a Presentation Component, a Control Component, and a Functional Core Adaptor component (referred to as domain adaptor in [Arch 91]). These principles constitute a general frame of reference for user interface developers. More detailed architecture models were proposed ([Lisbon 90], PAC [Coutaz 90], ...), shedding some light on the contents of each general application component; only general theoretical large-grain elements are proposed however. Because of their generality these architecture models are often difficult to apply: for example, experience with the use of the multi-agent PAC model shows that developers often have problems identifying the agents for their system.

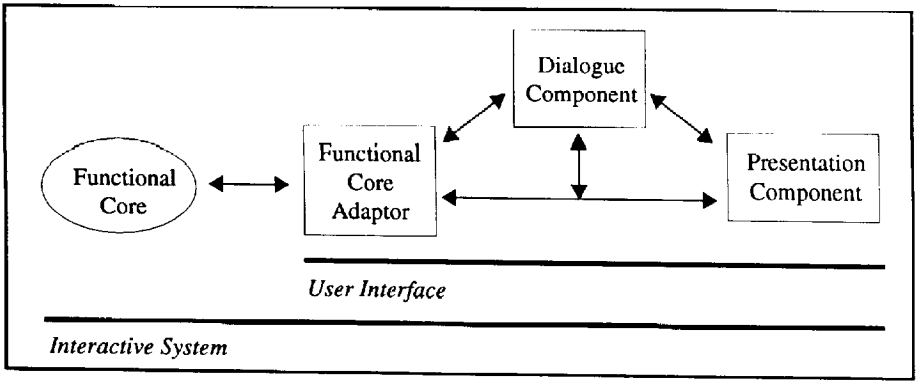

Figure 4: General principles for an interactive system architecture.

Our approach is to use SIROCO as a system description reference for a detailed fine-grain architecture model. This model is elaborated as an extension to a general model derived from PAC; using the SIROCO conceptual model as a reference, a set of objects was identified as the components of a system design. This extended architecture model is illustrated in figure 5; its main features can be thus summarized:

- The *Functional Core Adaptor* reflects the conceptual model of the UI; it is made of a set of objects representing the Application Concepts defined for the UI.

- The *Control Component* is decomposed into a set of control objects organized into a hierarchical structure. The session control object manages the whole user session, controlling the navigation among workspaces, their instanciation and deletion. The workspace control object is in charge with the management of a workspace; particularly, it coordinates data and operation control objects. A data control object manages an Application Concept presented in a Workspace: it initializes its presentation, gathers optional user input, and maintains the consistency between the internal Application Concept object and its presentation. The operation control object manages the operations available in a Workspace: it manages their presentation according to their current status (active or not), and controls the activation of "simple" operations, that is commands that do not require user input after the effective choice of the command; the operation control object delegates dialogue-thread objects to control the activation of "complex" operations. A dialogue-thread control object is in charge with the management of a direct dialogue between the system and the user in order to get the user input required to activate a command (i.e., parameter values or designation, object designation, confirmation for "dangerous" commands, etc.).

- The *Presentation Component* is decomposed into a set of objects dedicated to the management of a specific part of the interface image; these objects provide a toolkit-independent layer for use by the Control Component. Perspective objects are in charge with the display of a Perspective on an Application Concept as well as with the user input related to this Perspective. Workspace objects deal with the general presentation of a Workspace. Menu-bar objects are concerned with the presentation of the menu-bar of a Workspace, and the reception of user input related to this menu-bar. Param-form objects manage forms to get parameter input values. Dialogue-thread objects are dedicated to the management of dialogue windows such as confirmation windows, on which direct synchronous dialogue with the user is based.

The precise description of each of the model implementation objects ([Normand 92])

is out of the scope of this paper.

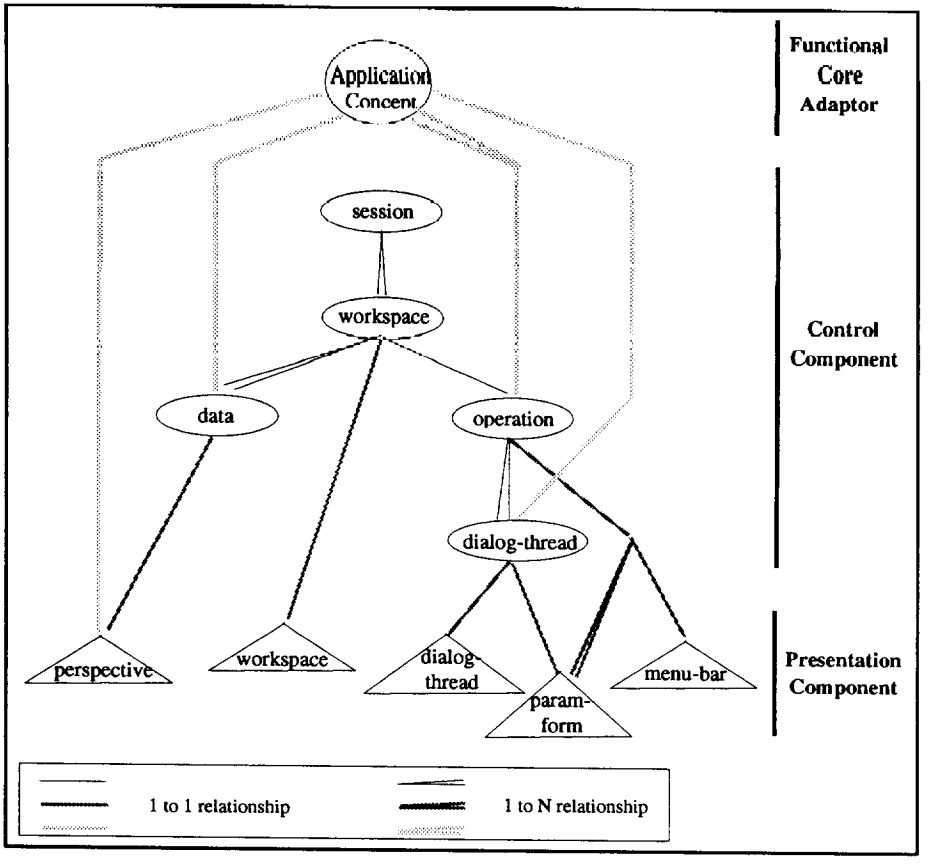

Figure 5: The SIROCO extended architecture model.

5.2 An Object Factory

A prototype of an object factory was implemented as part of SIROCO, with the GUIDE language as a target language. This code generator offers an automation of the translation process from conceptual objects to implementation objects with the following features:

- respect of the architecture model introduced above, so as to ensure modularity and maintainability of the generated code: the generated objects are "accessible" to programmers for optional tailoring and change.
- use and extension of an existing UIMS in the GUIDE environment [Normand 90] as a platform for the generated code.
- use of X Window and the Motif toolkit to implement the Presentation objects.
- maximization of object instances reuse at execution time (a simple technique to help ensure correct performances at execution time).

This object factory provides powerful prototyping facilities since the generated objects fully implement the functionalities specified with SIROCO; in particular, the generated Control Component objects manage the dialogue and UI dynamics according to the constraint properties defined on operations, the sequencing properties of Workspaces, and the interaction style parameters. These objects are not limited to prototyping however: they can be used as part of the final system code as well.

The main shortcomings of the generated code lie with the presentation objects: the automatic interface image they implement needs reviewing and improvement in order to fit in the final interface. The X resources mechanism provides a way to refine or change image details; in the case where a profound change is necessary, a whole presentation object may be replaced by another developer-defined object.

6 Related Work

Modeling the functional aspects of a UI and automatically generating code supporting this UI is nothing new, even though this approach has rarely been adopted. MIKE [Olsen 86] uses command and parameter descriptions to automatically generate a user interface. UIDE [Foley 88] has an approach closer to ours: UI specification is based on an object-oriented model; functional aspects of an interface are described in terms of object classes, a UIDE object corresponding to a SIROCO Application Concept. A UIDE object definition however features interaction style elements, e.g. the notion of current selection. Related tools have been developed in the data base application domain; for example, MacIDA UIMS [Petoud 90] generates a UI from functional specifications based on the description of entity-relationship schemes as well as atomic functions and synchronization messages (to express dynamic constraints on user actions).

Our research extends functional modeling as featured in UIDE along two directions: first, on the semantic/logical information front, allowing the collection of meta data on UI concepts; second, SIROCO addresses the use dimension of a UI, allowing the representation of Workspaces, and offering with the notion of Perspective an additional freedom degree in the expression of the conceptual elements presented in a Workspace.

Another innovative aspect of SIROCO is its incursion from UI design into UI system architecture modeling, supporting a continuous object-oriented approach to both tasks.

7 Conclusion

We have described basic aspects of SIROCO, a user interface specification model, language and code generator. SIROCO focuses on conceptual modeling, and proposes a representation model distinguishing the function dimension and the use dimension in an interactive system; interaction style and presentation aspects are dealt with as generic parameters. SIROCO fits in an object-oriented programming environment, and promotes a continuous object-oriented approach from UI design to UI implementation.

Current status. A first prototype of the object factory has been developed; although some features of the SIROCO language are not yet fully supported, the prototype allows the generation of the interfaces of systems such as the e-mail system.

Prospects. Prospects are threefold: extensions to the model, specification tools, and

evaluation support. In its present shape, the SIROCO model suits relatively simple form- or icon-based interfaces; the augmentation of the model in order to widen the target interfaces scope is one prospect - in particular, direct manipulation concepts are under study. Specification tools include editors allowing interactive specification of SIROCO descriptions, interactive adaption of the automatically generated image, etc. UI design evaluation is a topic that SIROCO does not address further than completion and connectivity checking; linking conceptual modeling to a user task description tool (e.g., MAD [Pierret-Golbreich 89] [Scapin 90]) is a particularly interesting prospect that would allow the evaluation of a conceptual design regarding the accomplishment of the user tasks.

References

[Arch 91] The Arch Model: Seeheim Revisited, User Interface Developers' Workshop, April 26, 1991.

[Barthet 88] Barthet M. F., Logiciels Interactifs et Ergonomie, Dunod, 1988.

[de Champeaux 91] de Champeaux D., Object-Oriented Analysis and Top-Down Software Development, ECOOP'91, July 1991, p. 360-376.

[Coutaz 90] Coutaz J., Interface homme-ordinateur : conception et réalisation, Dunod Publ., 1990.

[Foley 88] Foley J., Kim W. C., Kovacevic S., Murray K., The User Interface Design Environment, Report GWU-IIST-88-4, George Washington University, January 88.

[Goldberg 80] Goldberg A., Smalltalk-80: The Interactive Programming Environment, Addison-Wesley Publ., 1984.

[Green 85] Green M., The Design of Graphical User Interfaces, Ph. D. Thesis, CSRI-170-85, Computer Systems research Institute, University of Toronto, 1985.

[Henderson 86] Henderson D. A. Jr., Card S. K., Rooms : The Use of Multiple Virtual Workspaces to Reduce Space Contention in a Window Based Graphical User Interface, ACM Transactions on Graphics, (5)3, July 1986, p. 211-243.

[Hill 86] Hill R. D., Supporting Concurrency, Communication, and Synchronization in Human-Computer Interaction: the Sassafras UIMS, ACM Transactions on Graphics, (5)3, July 1986.

[Jacobson 87] Jacobson I., Object Oriented Development in an Industrial Environment, OOPSLA'87, October 87, p. 183-191.

[Krakowiak 90] Krakowiak S., Meysembourg M., Nguyen Van H., Riveill M., Roisin C., Design and implementation of an object-oriented, strongly typed language for distributed applications, Journal of Object-Oriented Programming, September 1990.

[Lisbon 90] User Interface Management and Design, Proceedings of the Workshop on User Interface Management Systems and Environments, Lisbon, Portugal, 4-6 June 1990.

[Moran 81] Moran T. P., The Command Language Grammar : a representation for the user interface of interactive computer systems, International Journal of Man-Machine Studies, (15),1981, p. 3-50.

[Normand 90] Normand V., A Practical Framework for Interactive Applications in GUIDE, an Object-Oriented Distributed System, Proceedings of TOOLS'90, July 1990, p. 657-768.

[Normand 92] Normand V., Le modèle SIROCO : de la spécification conceptuelle des interfaces utilisateur à leur réalisation, Thèse de doctorat de l'université Joseph Fourier - Grenoble I, 1992.

[OSF 89] OSF/Motif Programmer's Reference Manual, Open Software Foundation, Cambridge, MA, 1989.

[Olsen 86] Olsen D., MIKE: The Menu Interaction Kontrol Environment, ACM Transactions on Graphics, 5(4), October 1986.

[Petoud 90] Petoud I., Génération automatique de l'interface homme-machine d'une application de gestion hautement interactive, Ph.D. Thesis, Université de Lausanne, Switzerland, 1990.

[Pfaff 85] User Interface Management Systems, G. E. Pfaff ed., Eurographics Seminars, Springer-Verlag, 1985.

[Pierret-Golbreich 89] Pierret-Golbreich C., Delouis I., Scapin D., Un outil d'acquisition et de représentation des tâches orienté objet, Rapport 1063, INRIA, Rocquencourt, France, August 1989.

[Scapin 90] Scapin D., Aiding mechanisms for the design of user interfaces, Proceedings of the First International Conference on Automation Technology, July 1990..

[Schmucker 86] Schmucker K., MacApp: An Application Framework, Byte, 11(8), 1986, p. 189-193.

[Sibert 86] Sibert J. L., Hurley W. D., Bleser T. W., An Object-Oriented User Interface Management System, SIGGRAPH'86, Dallas, 1986.