A UIMS FOR KNOWLEDGE BASED INTERFACE TEMPLATE GENERATION AND INTERACTION

Christian Martín

Fachhochschule Augsburg, FB Allgemeinwissenschaften und Informatik, Baumgartnerstr. 16
D-89000 Augsburg

A knowledge based UIMS that exploits the semi-structured nature of office objects and tasks to generate flexible, user-adapted dialogue is presented. To produce interface templates for a broad range of interactive applications, the conceptual structure of the application data and abstract dialogue objects serve as the elements of a frame-like representation of the user interface. This description is refined by a dialogue manager that evaluates rules for presentation and user preferences. A refined interface frame, which is still independent of underlying I/O-tools, is mapped to the most suitable classes of the toolkit in the preferred environment (e.g. Andrew or OSF/Motif) by a generation component. The dialogue design process is illustrated for a multimedia editor application.

1 Introduction

During the last years major advances in user interface management systems and object-oriented information systems have led to powerful and usable development environments and solutions for the office domain. The CT-UIMS approach (Conceptual Template UIMS) is primarily aimed at supporting applications like intelligent editors, electronic mail, and document retrieval systems. Those systems rely heavily on efficient and highly interactive user interfaces and provide an internal data or object representation that can directly be exploited for knowledge based automatic user interface construction. With the current implementation of the system productivity of application programmers and end-users can be considerably improved. At the same time a high degree of design flexibility can be realized.

Discussion of general models for interactive software systems, [see Balzert (1987), Lantz, Tanner (1987)], experience with prototypical knowledge based architectures [Martín, Wolhöhr (1988)], and consequences from the dynamic evolution of prototypical and commercially available user interface tools [Myers (1989)] resulted in four major design goals for CT-UIMS:

1) Providing maximum separation of application semantics and dialogue functionality.

2) Allowing for easy integration of existing and future standard user interface toolkits into the UIMS by decoupling the high-level aspects of dialogue generation from the mapping to toolkit primitives.

3) Decomposition of applications with substantial functional contributions from low-level interaction (e.g. editors) into an abstract requirements-definition part and executable class-instances.

4) Flexible and adaptable general or user-task-specific embedding of frame-based dialogue knowledge.

---

*a Formerly at Olivetti D.O.R. / TA Research Lab, Nürnberg, where this work was funded by the Commission of the European Communities as part of ESPRIT-Project 28 "MULTOS".*
The global architecture of CT-UIMS, which is now a part of the MULTOS system (Multi Media Office Server), see e.g. [Constantopoulos, Yeorgaroudakis (1986)], is shown in figure 1. The central component of the UIMS, the dialogue manager, is responsible for controlling the user interface design process by evaluating several dialogue knowledge bases according to a given strategy. The construction process uses a central dialogue design workspace for the gradual refinement of abstract user interface objects in a way directed by the results of the knowledge base consultation. The knowledge bases of the dialogue manager contain frame-like representations of the dialogue objects, dialogue profiles for different applications as well as general, user-, and application-dependent presentation rules.

An abstract programming interface, and facilities for general user guidance and inter-application communication are also provided by the UIMS.

Different user interface toolkits can optionally be integrated into CT-UIMS without affecting the higher levels of the dialogue. Each toolkit is coupled to the UIMS by a special interface generator. The generator activates the relevant toolkit classes for presentation and mediates user inputs between the active toolkit and the dialogue manager.

Automatic user interface generation by a high-level UIMS is also discussed in [Singh, Green (1989)]. Another recent approach to knowledge based dialogue design is presented in [Wiecha, Bennett (1989)].

2 Building Applications with CT-UIMS

The requirements for generating advanced user-system dialogues with CT-UIMS and the different steps of the automatic design process will be discussed in detail for one of the MULTOS applications, a conceptual structure editor, that was implemented under CT-UIMS.

2.1 Abstract User Interface Representation

For building application user interfaces with CT-UIMS similarities between the data representation of the applications and their potential user interface structures have to be located. An abstract representation of the data objects used within applications for interaction with the user is essential, in order to allow to clearly separate the application semantics from the user interface functionality. There exist several UIMS approaches that use a declarative approach to generate direct manipulation interfaces [e.g. Olsen (1989)]. Such UIMSS use the definitions of scalar-, vector-, array-, and record-variables of application programs for building adequate user interfaces. In addition to exploiting such standard data types in CT-UIMS the definitions of conceptual document structures serve as abstract frame representations of the user interface, especially for multimedia document processing applications. During the design process such frames are refined by the dialogue manager and translated to interactive screen representations of either instances of conceptual document types or raw templates, which simplify the creation of new documents for the user.

Templates are also the principal user interface structures of the Object Lens System [Lai, Malone (1988)], where the concept of semi-structured objects is emphasized for a broad range of interactive applications. The user interface frames used by CT-UIMS, which were inspired by the more formalized MULTOS-CSD [Barbic, Rabitti (1985)], can be seen as suitable representations for such semi-structured objects. In other terms, the same data representation scheme which serves for user interface generation in CT-UIMS, could be employed for describing office tasks, procedures, evaluation rules, etc. (N.B. Ultimately, the dialogue manager - provided with the necessary knowledge and functionality - could be used to coordinate a unified, intelligent, and multi-threaded dialogue among all users, applications, and resources of the system, including databases).

2.2 Design of a Conceptual Structure Editor

Advanced information systems distinguish between a logical, a layout, and a conceptual view on the structure of a document. The conceptual view, as it was defined in the MULTOS document model [Eirund, Kreplin (1988)], describes the semantics of a document as a tree, the already mentioned CSD (Conceptual Structure Definition). A CSD identifies and names all of the essential semantic components of a document, with their associated contents and content attributes. Components may be basic, complex or refinable objects, they may be optional or mandatory, unique or iterative. The components or nodes of a CSD specify a type.

![Image of a Document Type Hierarchy](image)
An example type hierarchy was defined as the basis for classifying any business document as an instance of a given type. Figures 2 and 3 illustrate the conceptual typing approach of the MULTOS system. The conceptual types serve as efficient structures for data base access, query formulation, content based retrieval, and filtering mechanisms for electronic mail distribution. In structured editors, conceptual types can be used to generate templates for the preparation of business documents with an a priori CSD for later retrieval purposes. CTME (CSD-Template Editor) was designed with this main goal in mind.

By conceiving CTME as a CT-UIMS application [Öenig (1989), Martín, Öenig (1989)], a user-centered editor with a maximum degree of independence of the underlying toolkit and window server could be realized, which is easily portable between different target environments.

![Diagram](image)

**Figure 3. Textual Tree Representation of the CSD-Type Mail**

In CTME, the MULTOS type hierarchy and the CSD-formalism fulfill four purposes:

- visualization of existing documents and their CSDs for structured editing,
- generation of raw templates of a specific CSD-type to facilitate the preparation of a priori CSD-structured document instances,
- automatic construction of abstract user interface frames for any new CSD,
- type refinement and interactive creation of new CSD-types.

### 2.3 The Dialogue Manager

Automatic dialogue design may be seen as a cooperative process based on contributions from several design knowledge bases which are evaluated and coordinated by the dialogue manager. A design workspace (similar to a simplified blackboard) is used for modelling the different stages of the dialogue design process, holding the dynamic user interface frames for any CT-UIMS application in their various states of completion. In contrast to the blackboard approach (Engelmore, Morgan 1988), where knowledge sources may actively access the objects located in the workspace, in CT-UIMS the dialogue manager alone controls the sequence of action by maintaining the priorities of the different knowledge bases, interpreting the abstract user interface frames, evaluating dialogue rules, and finally modifying the objects in the workspace. The dialogue control strategy, which tries to solve occurring design conflicts by allowing to restore intermediate design states, whenever necessary, is - like the rest of the dialogue manager - implemented in C, for efficiency reasons. The dialogue design process and the contents of the design knowledge bases will now be discussed for the CTME editor.

The dialogue **object classes** knowledge base contains an abstract frame-like description of standard dialogue resources like windows, menus, forms, text editor objects, images, bitmaps, graphics editor objects, animations, etc. In general, the user interface of each application is constructed as a spatial and temporal combination of different dialogue standard object instances.

For the CTME-application text-editor objects and menus are essentially important as dialogue objects. The descriptions of a simple text-editor object and a selection-list object are contained in the knowledge base and have the following structure:

```
((dialogue_object_class text_editor_object)
 (identification <id_number>)
 (data_object <text>)
 (display_object <input_output_window> ( <insert> <delete> ... <search> <replace> ))

((dialogue_object_class selection_list)
 (identification <id_number>)
 (data_object <array_of_string>)
 (display_object <input_output_window> ( <select_item> ))
```

The standard set of currently more than forty abstract dialogue object classes is easily extensible. Their interactive functions are specified in C source code notation and reside in the same KB. For the application programmer these standard dialogue objects form the building blocks for the design of the user interface of any specific application.
As the application programmer knows the data structure of the application, it is an easy task for her/him, to configure the abstract frames for the templates of the desired user interface by combining the appropriate dialogue objects from the knowledge base. Only a very limited vocabulary of keywords is needed for slot definition. If some aspects of the application data or functionality cannot be covered by existing dialogue objects, extensions to existing classes or new classes and their inter-active functions have to be specified within the dialogue object classes knowledge base. Such modifications of the class specifications have to be mirrored in the toolkit primitives chosen for the actual screen representation of the user interface.

The abstract user interface frames for the templates of a document or a type, i.e. the interface between applications and CT-UIMS, are organized in application profiles. These profiles provide the semantic structure of the user interface that will be generated by the UIMS. They also hold the user interface frames for initialization and subtask-selection. The frames contained in the profiles are reusable by any other applications with similar user interface requirements.

As an example, interface frames for CTME's main window and for the MULTOS-type mail are given. Note, that for any new document type with no UI-frame entry in the application profile, CTME will automatically generate a default frame by parsing the type's CSD-description, before the document's user interface is created.

```
(user_interface_frame
  (application_class document_preparation)
  (ui_frame_id CTME_1)
  (ui_frame_class application_frame)
  (node_identifier application_logo)
  (node_type NIL)
  (node_type application_logo_content)
  (node_type animation))
  (node_identifier task_selection)
  (node NIL)
  (node_type task_selection_content)
  (node_type selection_list)
  (node_type "Administration"
    "Template Generator"
    "Document Editor"
    "Type Refinement"))
```

Contrary to the CSD representation of the document type mail, only basic nodes have to be considered in the user interface frame of this type:

```
(document_preparation)
  (CTME_12)
  (conceptual_document_template)
  (mail)
  (m_sender)
  (no_scroll_text_output_object)
  (m_sender_content)
  (no_scroll_text_editor_object)
  /* semantic relation between a CSD-
   * node and its contents */
  (m_recipient)
  (no_scroll_text_output_object)
  (m_recipient_content)
  (no_scroll_text_editor_object)
  (m_date)
  (no_scroll_text_output_object)
  (m_date_content)
  (date_object))
  (sender_name)
  (no_scroll_text_output_object)
  (sender_name_content)
  (no_scroll_text_editor_object)
  (m_subject)
  (no_scroll_text_output_object)
  (m_subject_content)
  (no_scroll_text_editor_object)
  (m_cc)
  (no_scroll_text_output_object)
  (m_cc_content)
  (no_scroll_text_editor_object)
  (text)
  (no_scroll_text_output_object)
  (text_content)
  (scroll_multimedia_editor_object)))
```

The semantic information of the user interface frames is refined and completed with presentation, style, and layout information by the dialogue manager during the design process. Thus, one and the same abstract user interface frame can result in a wide spectrum of refined frames depending on the characteristics of the application environment and the current user.

In the first refinement phase a knowledge base with presentation and style rules is exploited by the dialogue manager. The presentation rules and their associated functions are responsible for the general aspects of the layout design and the styles of the node contents. Some typical presentation rules are the following:

```
  (if presentation_mode == NIL)
    (presentation_mode = window_mode)
  (if (number(node) - 1) <= 5
    and presentation_mode == window_mode)
    (node_orientation = horizontal)
  (if (number(node) - 1) >= 6
    and (number(node) mod 3) == 0)
    (nodes_per_row = 3))
```

```
  (node_justify = TRUE)
```
increase design efficiency. Conditions of rules covering more than one type, can also be linked by or-operators.

After the user-profile evaluation the extended user interface frame for the CSD type mail has the following structure:

```
(user_interface_frame
  (application_class  document_preparation)
  (ui_frame_id  CTME_12)
  (ui_frame_class  conceptual_document_template)
  (presentation_mode  window_mode)
  (document_type  mail)
  (node_orientation  horizontal)
  (node_justify  TRUE)
  (help_exist  TRUE)
  (nodes_per_row  1)
  (node_identifier  help_info_area)
  (dobj_type  NIL)
  (node  help_info_content)
  (dobj_instance  "help/chris/standard")
  (dobj_position  right))
```

In the second design phase, the dialogue manager checks whether the current user profile contains any design rules that would modify some layout or style aspects of the user interface. These rules may cover both application-independent or application-specific user preferences. In CTME, these rules can influence the structure and the content styles of the extended user interface frame for each CSD type of some importance to a specific user.

The rules have the same structure as the general presentation rules. They can handle aspects such as the sequence of nodes, alias names of nodes, horizontal or vertical layout ordering, multimedia and other aspects.

Some typical user profile rules are listed below:

```
((if document_type = mail)
  (node_orientation = horizontal))

((if document_type = offer)
  (nodes_per_row = 4)
  (header_image = TRUE)
  (if node = sender_content)
    (dobj_type = no_scroll_raster_image)
  (if node = date_content)
    (date_format = "(ddmmyy/)"))

((if document_type = mail)
  (if (node_identifier = text) and
    (node_identifier = help_info_area)
    (font = "Swiss" "italic"))

((if document_type = offer)
  (if (node_identifier = locality)
    (alias = "Locality")
    (font = "italic"))

((if document_type = mail)
  (help_exist = TRUE)
  (if node = help_info_content)
    (dobj_instance = "help/chris/standard")

For an application like CTME the user profile entries that handle the nodes of a specific type can be grouped to a single rule in order to maintain consistency and

2.4 Interface Layout Generation

An extended user interface frame serves as the input for the interface generation component that creates a toolkit-specific interactive user interface layout with the resources of the selected toolkit. The generation component interprets the extended user interface frames and replaces abstract dialogue objects with the instances of the appropriate classes in the CT-UIMS resource set, which were built from toolkit primitives. A table that
maintains the mappings from all abstract dialogue object classes to the best fitting toolkit classes is part of the generation component.

<table>
<thead>
<tr>
<th>m_date</th>
<th>Tuesday, 12/12/1989</th>
</tr>
</thead>
<tbody>
<tr>
<td>sender_name</td>
<td>Christian Maertin</td>
</tr>
</tbody>
</table>

Dear Hans,

included you find a recent offer for future SUNLITE installations at our office. Could you please check, which of the systems can be covered by our current budget.

<table>
<thead>
<tr>
<th>Type</th>
<th>RAM</th>
<th>Speed</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUNLITE 3/3</td>
<td>4 MB</td>
<td>1.3 MPS</td>
<td>$10</td>
</tr>
<tr>
<td>SUNLITE 3/6</td>
<td>8 MB</td>
<td>2.5 MPS</td>
<td>$13</td>
</tr>
<tr>
<td>SUNLITE 3/8</td>
<td>16 MB</td>
<td>4.0 MPS</td>
<td>$18</td>
</tr>
<tr>
<td>SUNLITE 4/1</td>
<td>32 MB</td>
<td>7.0 MPS</td>
<td>$20</td>
</tr>
</tbody>
</table>

Regards,

Figure 4. CTME Mail Layout for User A

Special attention is paid to programmer contributed interactive functions for data manipulation that need to be embedded into the menus of the application user interface. In an editor application like CTME, low-level functionality, which is generally seen as a logical part of the toolkit or I/O-level, forms a major part of the application functionality. Comfortable macro functions and flexible search and replace commands, that were not part of the CT-UMS resource set, therefore had to be defined in the abstract dialogue object classes. The functions are coded in C - independent of the toolkit - as part of the application. The generation component downloads these functions to the toolkit level before runtime, in order to maximize system performance.

By interpreting the extended user interface frame, the generation component creates an interactive document editor layout. Figure 4 shows the interface for the type mail, after a document was created. Different profile entries lead to alternative template layouts (see fig. 5).

Each node of the user interface frame is translated into a toolkit class instance. Depending on the specification of the abstract dialogue objects, the nodes differ in their functional behaviour. In the mail example, node identifiers were mapped to read-only windows, whereas the node contents where mapped to several text-editor windows, a raster image window, a multimedia-editor window for the document body, and a help window.

Each window type offers a different set of functions, which are available for the end-user to interactively prepare electronic documents.

To create balanced user interface layouts, the size of the screen representation of the extended user interface frame, which is internally computed by the generator, must be within the limits of the available screen space (especially, if a tiled window manager is used) or the default size of the new window. If some of the template nodes cannot be placed within the window area to be displayed, the generation component informs the user, via the dialogue manager, that an adequate screen representation of the template is not possible. If the user wishes, the dialogue manager selects the less space-consuming default interface frame from the root user profile, adds to it the properties of the node contents of the rejected frame and again passes it to the generator.

Document Type: mail

m_sender:

m_recipient: Hans von Kleist
m_date: 12.12.1989
sender_name: Christian Maertin
m_subject: SUNLITE-Offer
m_cc: EF1

text:
Documents (user interfaces and contents) can interactively be refined and modified by the user, stored for later use, or mailed. The user has the option to enable adaptation mode, when documents are stored after an editor session. In this case, the dialogue manager will become active again and may alter part of the right-hand sides of the type-specific rules in the user profile to the values that were selected in the current document's user interface. A mail user, who receives a CTME document may either visualize the document with the layout generated by the remote CT-UIMS, or create a version of the document with layout and style features adapted to her/his personal environment by invoking the local CT-UIMS dialogue manager. If the receiving user prefers a different toolkit, the document can even be mapped to that toolkit by invoking the relevant interface generator. For this purpose all abstract dialogue objects have to be consistently represented as classes of both toolkits.

3 Implementation and Future Work

For the current CT-UIMS implementation, which has been running for several months on top of X11R3 on SUN workstations, an interface generator for the Andrew toolkit ATK [ATK (1988)] was developed first. ATK was chosen, because of the multimedia functionality inherent to its standard toolkit classes. The next step will be the implementation of an OSF/Motif [OSF (1989)] interface generator that will allow the easy migration of the complete UIMS and its applications to a broader range of UNIX machines. Future work will be concentrated on the evolution of the current architecture and will result in the design of several new components of the UIMS, such as:

- A CT-UIMS generated rule editor with support for consistency maintenance in dialogue profiles.

- An internationalization component that provides multilinguality by replacing the names of nodes and identifiers of UI frames with the correct expressions in the goal language.

- An interface generator for a voice toolkit which will allow simple mixed visual/voice dialogues.

Acknowledgements

The author’s special thanks go to Martina Oemig, who contributed many ideas and solutions to the design of the general architecture and implemented most of the prototypical CT-UIMS code as part of her diploma thesis work at TA Research.

References

Constantopoulos, P., Yeorgaroudakis, Y. et al. (1986). Office Document Retrieval in MULTOS.