Chapter 4

Syntax: Multimedia Languages

Visual languages, which let users customize iconic sentences, can be extended to accommodate multimedia objects, letting users access media dynamically. Teleaction objects, or multimedia objects with knowledge structures, can be designed using visual languages to automatically respond to events and perform tasks like “find related books” in virtual library Bookman.

Languages that let users create custom icons and iconic sentences are receiving increased attention, as multimedia applications become more prevalent. Visual language systems let the user introduce new icons, and create iconic sentences with different meanings and the ability to exhibit dynamic behavior. With a graphical user interface, the user must generally compose commands with predefined icons, which limits the range of commands and makes dynamic composition rather difficult. It is also awkward to customize such commands without cluttering the screen. These limitations are significant in multimedia applications because the user must often access multimedia information dynamically with very few icons.

At the University of Pittsburgh and Knowledge Systems Institute, we have developed a formal framework for visual language semantics that is based on the notion of icon algebra and have designed several visual languages for the speech impaired. In Chapter 3 we described the underlying grammar for a visual language. We have since extended the framework to include the design of *multidimensional languages* — languages that capture the dynamic nature of multimedia objects through icons, earcons (sound), micons (motion icons), and vicons (video icons). The user can create a multimedia message by combining these icons and have direct access to multimedia information, including animation.

We have successfully implemented this framework in developing Bookman, an interface to a virtual library used by the students and faculty of the Knowledge Systems Institute. As part of this work, we extended the visual language concepts to develop *teleaction objects*, objects that automatically respond to some events or messages to perform certain tasks [ChangH95b]. We are continuing work on extensions to the visual interface in the context of emergency management, where the information system must react to flood warnings, fire warnings, and so on, to present multimedia information and to take actions [Khali96].

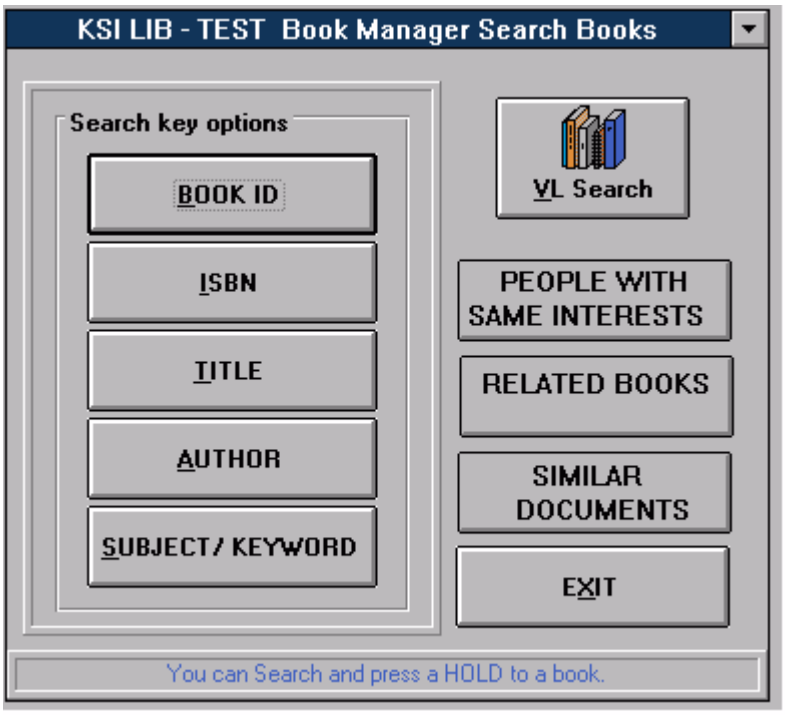

Figure 1. The BookMan interface to a virtual library lets the user select different search modes.

Figure 1 shows the search and query flexibility possible with the Bookman interface. In addition, users can perform a range of tasks, including finding related books, finding books containing documents similar to documents contained in the current book, receiving alert messages when related books or books containing similar documents have been prefetched by BookMan, finding other users with similar interests or receiving alert messages about such users (the last function requires mutual consent among the users) etc. In developing the interface, our goal was to give users the

same range of freedom they might experience in a real library. Much of this power stems from the use of TAOs.

1. WHAT TELEACTION OBJECTS DO

To create a TAO, we attached knowledge about events to the structure of each *multimedia object* — a complex object that comprises some combination of text, image, graphics, video, and audio objects. TAOs are extremely valuable because they greatly improve the selective access and presentation of relevant multimedia information. In BookMan, for example, each book or multimedia document is a TAO because the user can not only access the book, browse its table of contents, read its abstract, and decide whether to check it out, but also be informed about related books, or find out who has a similar interest in this subject. The user can indicate an intention by incrementally modifying the physical appearance of the book, usually with just a few clicks of the mouse.

TAOs can accommodate an almost limitless range of functions. For example, when the user clicks on a particular book, it can automatically access information about related books and create a multimedia presentation from all the books.

The drawback of TAOs is that they are complex objects and therefore the end user can not easily manipulate them with traditional define, insert, delete, modify, and update commands. Instead, TAOs require direct manipulation, which we provided through a multidimensional language.

The physical appearance of a TAO is described by a *multidimensional sentence*. The syntactic structure derived from this multidimensional sentence controls its dynamic multimedia presentation. The TAO also has a knowledge structure called the *active index* that controls its event-driven or message-driven behavior. The multidimensional sentence may be location-sensitive, time-sensitive or content-sensitive. Thus, an incremental change in the TAO's external appearance is an event that causes the active index to react. As I describe later, the active index itself can be designed using a visual-language approach.

2. MULTIDIMENSIONAL LANGUAGE

The multidimensional language consists of generalized icons and operators, and each sentence has a syntactic structure that controls the dynamics of a multimedia presentation.

2.1 Generalized icons and operators

In Chapter 3 we presented the elements of visual languages and described the icons and operators in a visual (not multidimensional) language. In a multidimensional language, we want not only icons that represent objects by images, but also icons that represent the different types of media. We call such primitives *generalized icons* and define them as $x = (x_m, x_p)$ where x_m is the meaning and x_p is the physical appearance. To represent TAOs, we replace the x_p with other expressions that depend on the media type:

- Icon: (x_m, x_i) where x_i is an image
- Earcon: (x_m, x_e) where x_e is sound
- Micon: (x_m, x_s) where x_s is a sequence of icon images (motion icon)
- Ticon: (x_m, x_t) where x_t is text (ticon can be regarded as a subtype of icon)
- Vicon: (x_m, x_v) where x_v is a video clip (video icon)

The combination of an icon and an earcon/micon/ticon/vicon is a multidimensional sentence.

For multimedia TAOs, we define operators as

- Icon operator $op = (op_m, op_i)$, such as *ver* (vertical composition), *hor* (horizontal composition), *ovl* (overlay), *con* (connect), *surround*, *edge_to_edge*, etc.
- Earcon operator $op = (op_m, op_e)$, such as *fade_in, fade_out*, etc.
- Micon operator $op = (op_m, op_s)$, such as *zoom_in, zoom_out*, etc.
- Ticon operator $op = (op_m, op_t)$, such as *text_merge, text_collate*, etc.
- Vicon operator $op = (op_m, op_v)$, such as *montage, cut*, etc.

Two classes of operators are possible in constructing a multimedia object. As described in Chapter 3, spatial operators are operators that involve spatial relations among image, text or other spatial objects. A multimedia object can also be constructed using operators that consider the passage of time. *Temporal operators*, which apply to earcons, micons, and vicons, make it possible to define the temporal relation [Allen83] among generalized icons. For example, if you want to watch a video clip and at the same time listen to the audio, you can request that the video *co_start* with the audio. Temporal operators for earcons, micons, ticons and vicons include *co_start, co_end, overlap, equal, before, meet,* and *during* and are usually treated as invisible operators because they are not visible in the multidimensional sentence.

When you use temporal operators to combine generalized icons, their types may change. For example, a micon followed in time by another icon is still a micon, but the temporal composition of micon and earcon yields a vicon. Media type changes are useful in *adaptive multimedia* so that one type of media may be replaced/combined/augmented by another type of media (or a mixture of media) for people with different sensory capabilities.

We can add still more restrictions to create subsets of rules for icons, earcons, micons and vicons that involve *special operators*:

- For earcons, special operators include *fade_in, fade_out,*
- For micons, special operators include *zoom_in, zoom_out,*
- For ticons, special operators include *text_collate, text_merge,*
- For vicons, special operators include *montage, cut.*

These special operators support the combination of various types of generalized icons, which means the multidimensional language can fully reflect all multimedia types.

2.2 Grammar

Multidimensional languages can handle temporal as well as spatial operators. A visual language has a relational grammar, *G*, which a compiler uses to generate sentences:

$$G = (N, X, OP, s, R)$$

where N is the set of nonterminals, X is the set of terminals (icons), OP is the set of spatial relational operators, s is the start symbol, and R is the set of production rules whose right side must be an expression involving relational operators.

To describe multidimensional languages, we extended the X and OP elements of G: X is still the set of terminals but now includes earcons, micons, ticons, and vicons as well as icons, and the OP set now includes temporal as well as spatial relational operators.

2.3 Syntax

Informally, a multidimensional language is a set of multidimensional sentences, each of which is the spatial/temporal composition of generalized icons. Figure 2 without the dialog box illustrates a simple visual sentence, which describes the physical appearance of a multimedia object retrieved by BookMan. With the dialogue box, the figure becomes a multidimensional

sentence used by BookMan to generate “The children drive to school in the morning.” in synthesized speech. The multidimensional sentence has the syntactic structure

(DIALOG_BOX *co_start* SPEECH) *ver* (((CHILDREN *hor* CAR) *hor* SCHOOL_HOUSE) *hor* SUNRISE)

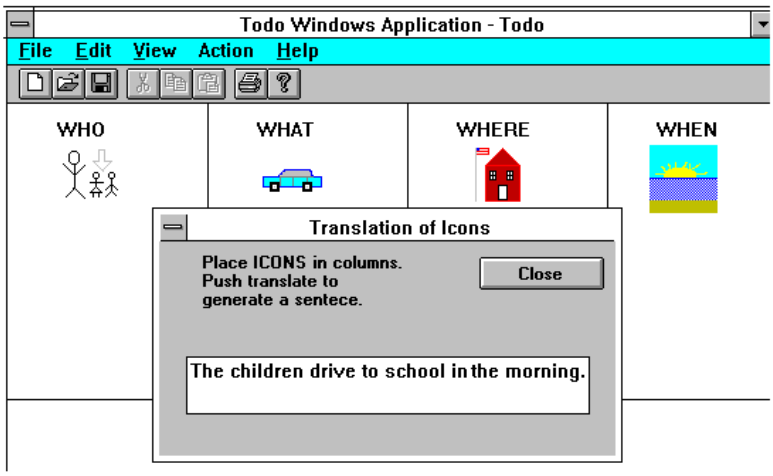

Figure 2. A multidimensional sentence whose meaning changes when the icons change their positions and is therefore a location-sensitive sentence. This sentence has the meaning “The children drive to school in the morning.”

Figure 3 is a hypergraph of the syntactic structure. The syntactic structure is essentially a tree, but it has additional temporal operators (such as *co_start*) and spatial operators (such as *hor* and *ver*) indicated by dotted lines. Some operators may have more than two operands (for example, the *co_start* of audio, image, and text), which is why the structure is called a hypergraph. The syntactic structure controls the multimedia presentation of the TAO.

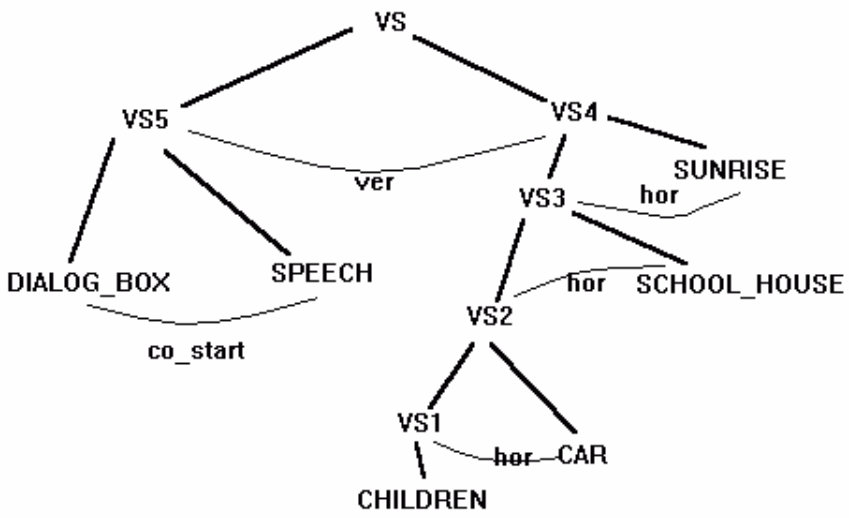

Figure 3. The syntactic structure of the multidimensional sentence shown in Figure 2. This structure is a hypergraph because some relational operators may correspond to lines with more than two end points.

Multidimensional languages must also account for multimedia dynamics because many media types vary with time. This means that a dynamic multidimensional sentence changes over time.

We defined rules for spatial and temporal operators that let us transform the hypergraph in Figure 3 to a Petri net that controls the multimedia presentation. Figure 4 represents the Petri net of the sentence in Figure 2. As such, it also a representation of the dynamics of the multidimensional sentence in Figure 2. The multimedia presentation manager can execute this Petri net dynamically to create a multimedia presentation [Lin96]. For example, the presentation manager will produce the visual sentence in Figure 2 as well as the synthesized speech.

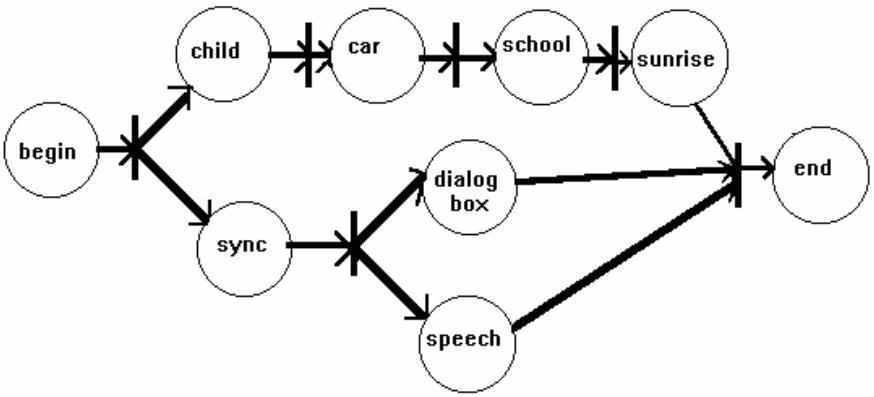

Figure 4. A time-sensitive visual sentence for the Petri net controlling the presentation of the multidimensional sentence shown in Figure 2.

3. KNOWLEDGE STRUCTURE

An *index cell* is the fundamental building block of the active index, which is the key element of a TAO [Chang95a]. Without the active index, a TAO would not be able to react to events or messages, and the dynamic visual language would lose its power.

3.1 Cell communication

An index cell accepts input messages, performs some action, and posts an output message to a group of output index cells. Depending on its internal state and the input messages, the index cell can post different messages to different groups of output index cells. Therefore the connection between an index cell and its output cells is dynamic. For example, if a Bookman user wants to know about new books on nuclear winter, he modifies the visual sentence, causing TAO to send a message to activate a new index cell that will collect information on nuclear winter.

An index cell can be either live or dead, depending on its internal state. The cell is live if the internal state is anything but the dead state. If the internal state is the dead state, the cell is dead. The entire collection of index

cells, either live or dead, forms the *index cell base*. The set of live cells in the index cell base forms the active index.

Each cell has a built-in timer that tells it to wait a certain time before deactivating (dead internal state). The timer is re-initialized each time the cell receives a new message and once again becomes active (live). When an index cell posts an output message to a group of output index cells, the output index cells become active. If an output index cell is in a dead state, the posting of the message will change it to the initial state, making it a live cell, and will initialize its timer. On the other hand, if the output index cell is already a live cell, the posting of the message will not affect its current state but will only re-initialize its timer.

Active output index cells may or may not accept the posted message. The first output index cell that accepts the output message will remove this message from the output list of the current cell. (In a race, the outcome is nondeterministic.) If no output index cell accepts the posted output message, the message will stay indefinitely in the output list of the current cell. For example, if no index cells can provide the BookMan user with information about nuclear winter, the requesting message from the nuclear winter index cell will still be with this cell indefinitely.

After its computation, the index cell may remain active (live) or deactivate (die). An index cell may also die if no other index cells (including itself) post messages to it. Thus the nuclear winter index cell in BookMan will die if not used for a long time, but will be re-initialized if someone actually wants such information and sends a message to it.

Occasionally many index cells may be similar. For example, a user may want to attach an index cell to a document that upon detecting a certain feature sends a message to another index cell to prefetch other documents. If there are 10,000 such documents, there can be ten thousand similar index cells. The user can group these cells into an *index cell type*, with the individual cells as instances of that type. Therefore, although many index cells may be created, only a few index cell types need to be designed for a given application, thus simplifying the application designer's task.

3.2 Cell construction

To aid multimedia application designers in constructing index cells, we developed a visual-language-based tool, IC Builder, and used it to construct the index cells for the BookMan interface. Figure 5 shows a prefetch index cell being built. Prefetch is used with two other index cell types to retrieve documents [Chang95a]. If the user selects the prefetch mode of the BookMan interface, the active index will activate the links to access information about related books. Prefetch is responsible for scheduling

prefetching, initiating (issuing) a prefetching process to prefetch multimedia objects, and killing the prefetching process when necessary.

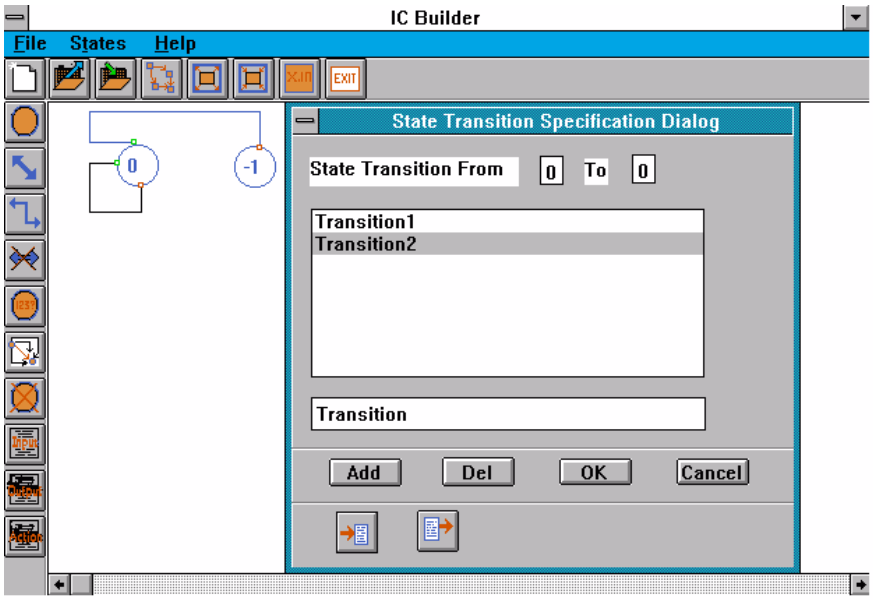

Figure 5. The visual specification of the state transitions for an active index cell of the virtual library's user interface BookMan.

Figure 5 shows the construction of the state-transition diagram. The prefetch index cell has two states: state 0, the initial and live state, and state -1, the dead state. The designer draws the state-transition diagram by clicking on the appropriate icons. In this example, the designer has clicked on the fourth vertical icon (zigzag line) to draw a transition from state 0 to state 0. Although the figure shows only two transition lines, the designer can specify as many transitions as necessary from state 0 to state 0. Each transition could generate a different output message and invoke different actions. For example, the designer can represent different prefetching priority levels in BookMan by drawing different transitions.

The designer wants to specify details about transition2 and so has highlighted it. Figure 6 shows the result of clicking on the input message icon (top icon to the right of the State Transition Specification Dialog box.) IC Builder brings up the Input Message Specification Dialog box so that the designer can specify the input messages. The designer specifies message 1 (start_prefetch) input message. The designer could also specify a predicate, and the input message is accepted only if this predicate is evaluated true. Here there is no predicate, so the input message is always accepted.

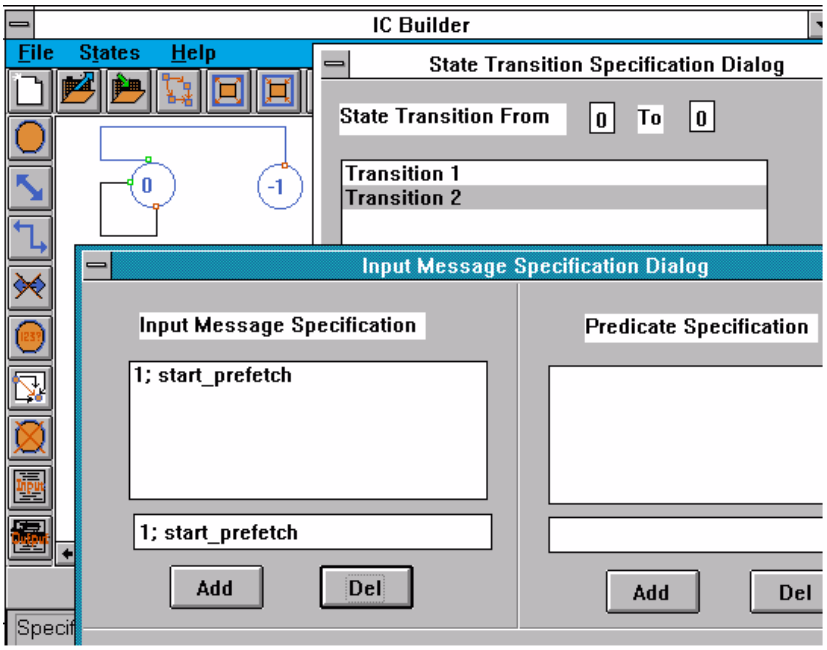

Figure 6. The visual specification of the input message for an active index cell of the virtual library's user interface BookMan.

Figure 7 shows what happens if the designer clicks on the output message icon in Figure 5 (bottom icon to the right of the State Transition Specification Dialog box). IC Builder brings up the Output Message Specification Dialog box so that the designer can specify actions, output messages, and output index cells. In this example, the designer has specified three actions: compute_schedule (determine the priority of prefetching information), issue_prefetch_proc (initiate a prefetch process), and store_pid (Once a prefetch process is issued, its process id or pid is saved so that the process can be killed later if necessary).

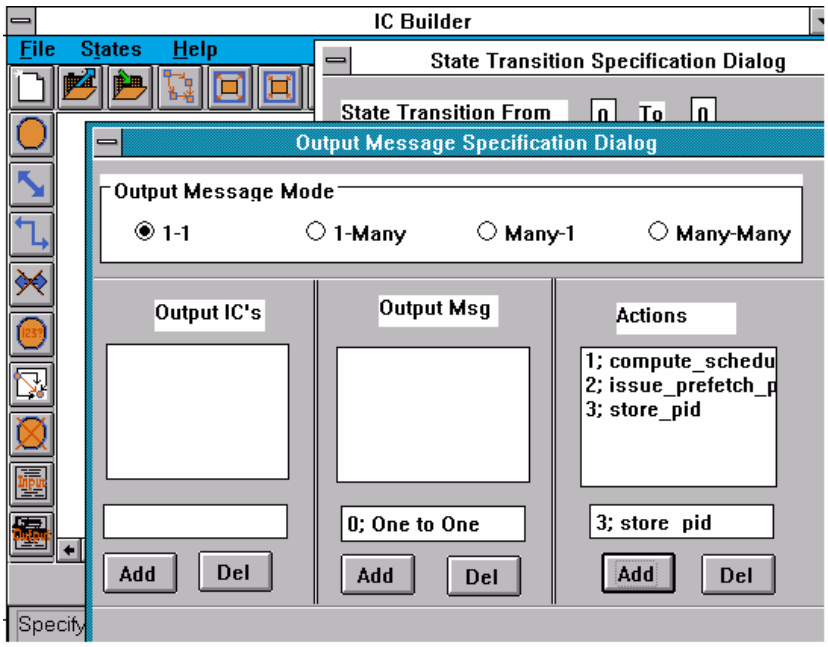

Figure 7. The visual specification of the output message and actions for an active index cell of the virtual library's user interface BookMan.

In the figure, there is no output message, but both input and output messages can have parameters. The index cell derives the output parameters from the input parameters.

4. DYNAMIC VISUAL LANGUAGE FOR QUERYING

When the user makes incremental changes to a multidimensional sentence, certain events occur and messages are sent to the active index. For example, suppose the user clicks on a book TAO to change the color attribute of the book. This is a *select* event, and the message *select* is sent to the active index. If the user creates a new *related_info* operation icon, this is a *related_info* event, and a message *prefetch_related_info* is sent to the

active index. The incremental changes to a multidimensional sentence can be either:

- *Location-sensitive*. The location attribute of a generalized icon is changed.
- *Time-sensitive*. The time attribute of a generalized icon is changed.
- *Content-sensitive*. An attribute of a generalized icon other than a location or time attribute is changed or a generalized icon is added or deleted, or an operator is added or deleted.

A visual sentence or multidimensional sentence can also be location-sensitive, time-sensitive, or content-sensitive. Chapter 3 gives examples of different types of visual sentences. The resulting language is a dynamic visual language or dynamic multidimensional language.

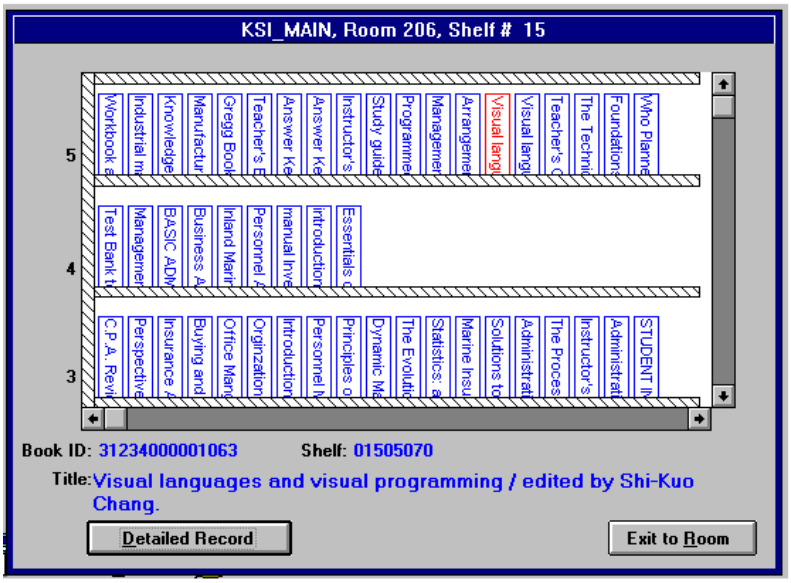

Figure 8. The BookMan interface to a virtual library lets the user browse the virtual library and select desired book for further inspection.

A dynamic visual language for virtual reality serves as a new paradigm in a querying system with multiple paradigms (form-based queries, diagram-based queries and so on) because it lets the user freely switch paradigms [Chang94a]. When the user initially browses the virtual library, the VR query may be more natural; but when the user wants to find out more details,

the form-based query may be more suitable. This freedom to switch back and forth among query paradigms gives the user the best of all worlds, and dynamic querying can be accomplished with greater flexibility.

From the viewpoint of dynamic languages, a VR query is a location-sensitive multidimensional sentence. As Figure 8 shows, BookMan indicates the physical locations of books by marked icons in a graphical presentation of the books stacks of the library. What users see is the same (with some simplification) as what they would experience in a real library. That is, the user selects a book by picking it from the shelf, inspects its contents and browses adjacent books on the shelf.

In Figure 1, initially the user is given the choice of query paradigms: search by title, author, ISBN, or keyword(s). If the user selects the virtual library search, he can then navigate in the virtual library, and as shown in Figure 8, the result is a marked object. If the user switches to a form-based representation by clicking the "DetailedRecord" button, the result is an item in the form of Figure 9. The user can now use the form to find books of interest, and switch back to the VR query paradigm by clicking the "VL location" button in Figure 9.

Essentially, the figure illustrates how the user can switch between a *VR paradigm* (such as the virtual library) and a *logical paradigm* (such as the form).

There are certain admissibility conditions for this switch. For a query in the logical paradigm to be admissible to the VR paradigm, the retrieval target object should also be an object in VR. For example, the virtual reality in the Bookman library is stacks of books, and an admissible query would be a query about books, because the result of that query can be indicated by marked book icons in the virtual library.

Conversely, for a query in the VR paradigm to be admissible to the logical paradigm, there should be a single marked VR object that is also a database object, and the marking is achieved by an operation icon such as *similar_to* (find objects similar to this object), *near* (find objects near this object), *above* (find objects above this object), *below* (find objects below this object), and other spatial operators. For example, in the VR for the virtual library, a book marked by the operation icon *similar_to* is admissible and can be translated into the logical query "find all books similar to this book."

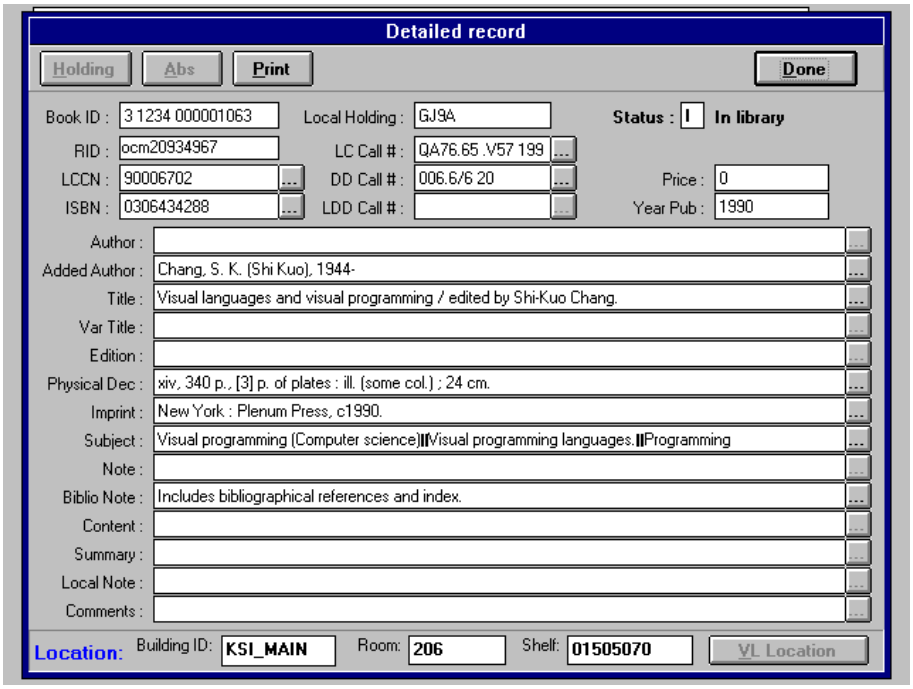

Figure 9. The BookMan interface to a virtual library also lets the user switch to a traditional form-based query mode.

5. DISCUSSION

Visual query systems for multimedia databases, like Bookman, are under active investigation at many universities as well as industrial laboratories. These systems are extremely flexible. For example, a user can easily and quickly ask for any engineering drawing that contains a part that looks like the part in another drawing and that has a signature in the lower right corner that looks like John Doe's signature. In fact, in our work on Bookman, we plan to build a mechanism that will let users create similarity retrieval requests that prompt Bookman to look for books similar to the book being selected.

We have implemented the active index for BookMan to support optional modes like prefetching. We can also extend BookMan to perform searches on the World Wide Web using a Web browser enhanced with an active index [Chang95a].

We have also used our multidimensional language framework to design user interfaces for personal digital assistants. Chapter 3 described TimeMan, a personal assistant that performs time-management functions. Just as books in BookMan are teleaction objects, so are calendars in TimeMan. We used a multidimensional language to describe the external appearance of a TAO calendar, and provided an active index to manage to-do items.

In summary, visual languages and multidimensional languages are useful in specifying the syntactic structure, knowledge structure, and dynamic behavior of complex multimedia objects such as TAO. As multimedia applications become widespread, we expect to see more visual query systems in which multidimensional languages will play an important role, both as a theoretical foundation and as a means to explore new applications.