VISUAL LANGUAGES

Languages that let users create custom icons and iconic/visual sentences are receiving increased attention as multimedia applications become more prevalent. Visual language systems let the user introduce new icons, and create iconic/visual sentences with different meanings and the ability to exhibit dynamic behavior. Furthermore, visual programming systems support problem solving and software development through the composition of basic software components using spatial operators such as "connect port #1 of component A to port #2 of component B."

We will first introduce the elements of visual languages, then describe how visual languages can be extended to deal with multimedia. We will illustrate a visual programming language for general purpose problem solving and for special application to database querying. Finally, on-line bibliographies for further reference and some thoughts concerning the future of visual languages and visual programming languages are provided.

ELEMENTS OF VISUAL LANGUAGES

A visual language is a pictorial representation of conceptual entities and operations and is essentially a tool through which users compose iconic, or visual, sentences (1). The icons generally refer to the physical image of an object. Compilers for visual languages must interpret visual sentences and translate them into a form that leads to the execution of the intended task (2). This process is not straightforward. The compiler cannot determine the meaning of the visual sentence simply by looking at the icons. It must also consider the context of the sentence, how the objects relate to one another. Keeping the user's intent and the machine's interpretation the same is one of the most important tasks of a visual language (3).

Icons

A visual sentence is a spatial arrangement of object icons and/or operation icons that usually describes a complex conceptual entity or a sequence of operations. Object icons represent conceptual entities or groups of object icons that are arranged in a particular way. Operation icons, also called process icons, denote operations and are usually context-dependent. Figure 1(top) illustrates a visual sentence that consists of horizontally arranged icons, with a dialog box overlaid on it. This particular location-sensitive visual sentence changes meaning when the locations of icons change [see Fig. 1(bottom)], and can be used to specify to-do items for TimeMan, a time-management personal digital assistant.

Figure 2 illustrates a content-sensitive visual sentence for TimeMan. The fish in the tank are object icons, each of which represents a to-do item, and the cat is an operation icon that appears when there are too many fish in the tank (the to-do list is too long). Figure 3 illustrates a time-sensitive visual sentence that changes its meaning with time. The icons (circles and vertical bars) in this visual sentence are connected by arcs. Thus this visual sentence is the visual representation of a directed graph called *Petri net.* When tokens flow in this directed graph, this visual sentence changes its meaning.

Operators

Icons are combined using operators. The general form of binary operations is expressed as x_1 op $x_2 = x_3$, where the two icons x_1 and x_2 are combined into x_3 using operator *op*. The operator $op = (op_m, op_p)$, where op_m is the logical operator, and op_p is the physical operator. Using this expanded notation, we can write $(x_{m1}, x_{p1})\ op\ (x_{m2}, x_{p2}) = ((x_{m1}\ op_m\ x_{m2}), (x_{p1}\ op_p\ x_{p2}))$. In other words, the meaning part x_{m1} and x_{m2} are combined using the logical operator op_{m}, and the physical part x_{p1} and x_{p2} are combined using the physical operator op_{p}.

Operators can be visible or invisible. Most system-defined spatial/temporal operators are invisible, whereas all user-defined operators are visible for the convenience of the user. For example, excluding the dialog box, the visual sentence in Fig. 1(a) is the horizontal combination of three icons. Therefore, it can be expressed as:

(CHILDREN *hor* SCHOOL_HOUSE) *hor* SUNRISE

where *hor* is an invisible operator denoting a horizontal combination. But if we look at Fig. 2, the cat is a visible operator denoting a process to be applied to the fish in the fish tank. An operation icon can be regarded as a visible operator.

The four most useful domain-independent spatial icon operators are *ver,* for vertical composition; *hor,* for horizontal composition; *ovl* for overlay; and *con,* for connect. The operators *ver, hor,* and *ovl* are usually invisible (see Fig. 1 for an example, where the *hor* operator is invisible). On the other hand, the operator *con* is usually visible as a connecting line (see Fig. 3 for an example, where the connecting lines among the icons called places and transitions are visible). This operator *con* is very useful in composing visual programs (see the Visual Programming Languages section).

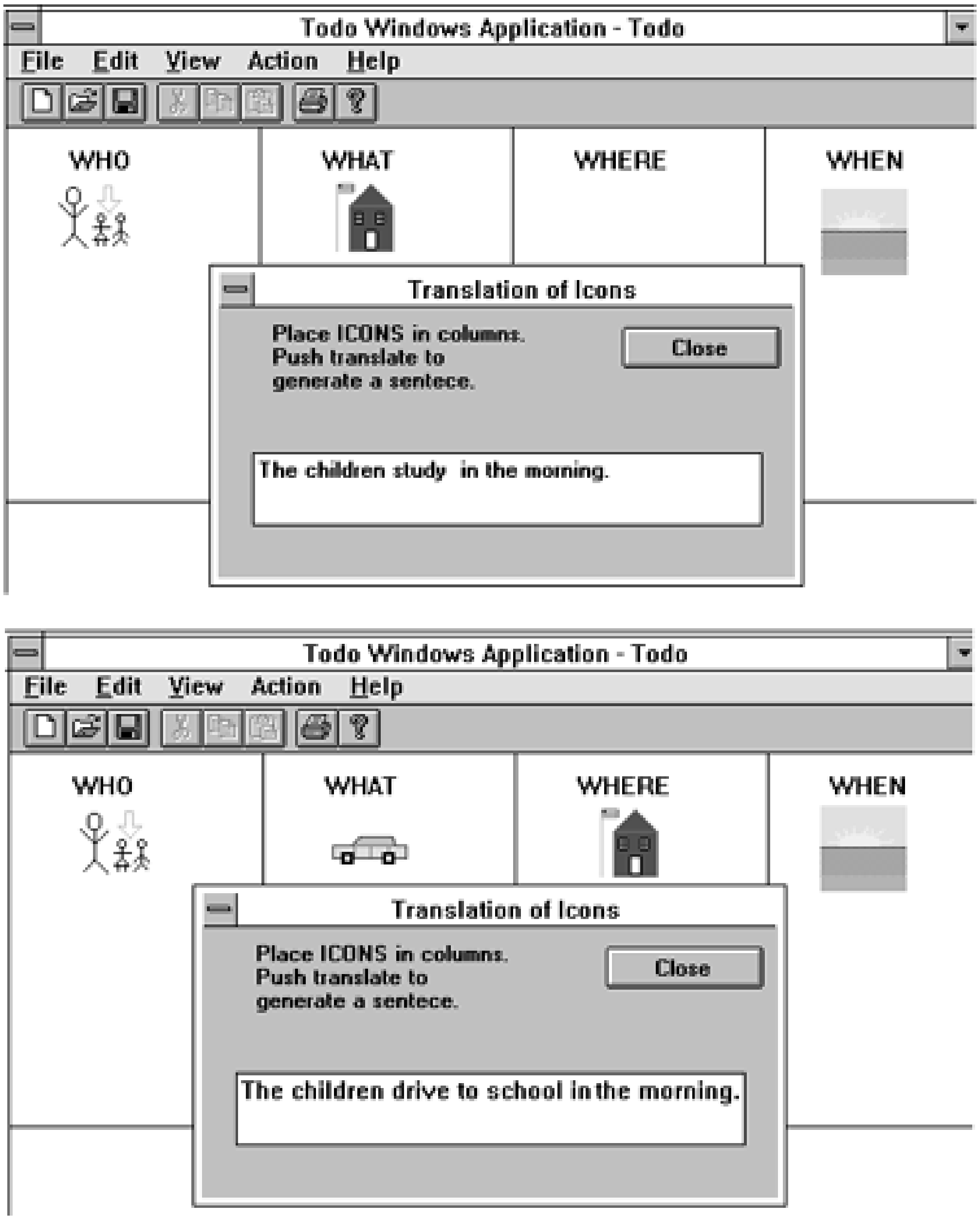
Grammar

A visual language has a grammar *G*, which a compiler uses to generate sentences belonging to this visual language:

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

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.

J. Webster (ed.), *Wiley Encyclopedia of Electrical and Electronics Engineering.* Copyright © 1999 John Wiley & Sons, Inc.


Todo Windows Application - Todo
File Edit View Action Help
WHO
WHAT
WHERE
WHEN
Translation of Icons
Close

Place ICONS in columns.
Push translate to
generate a sentece.

The children study in the morning.

Todo Windows Application - Todo
File Edit View Action Help
WHO
WHAT
WHERE
WHEN
Translation of Icons
Close

Place ICONS in columns.
Push translate to
generate a sentece.

The children drive to school in the morning.

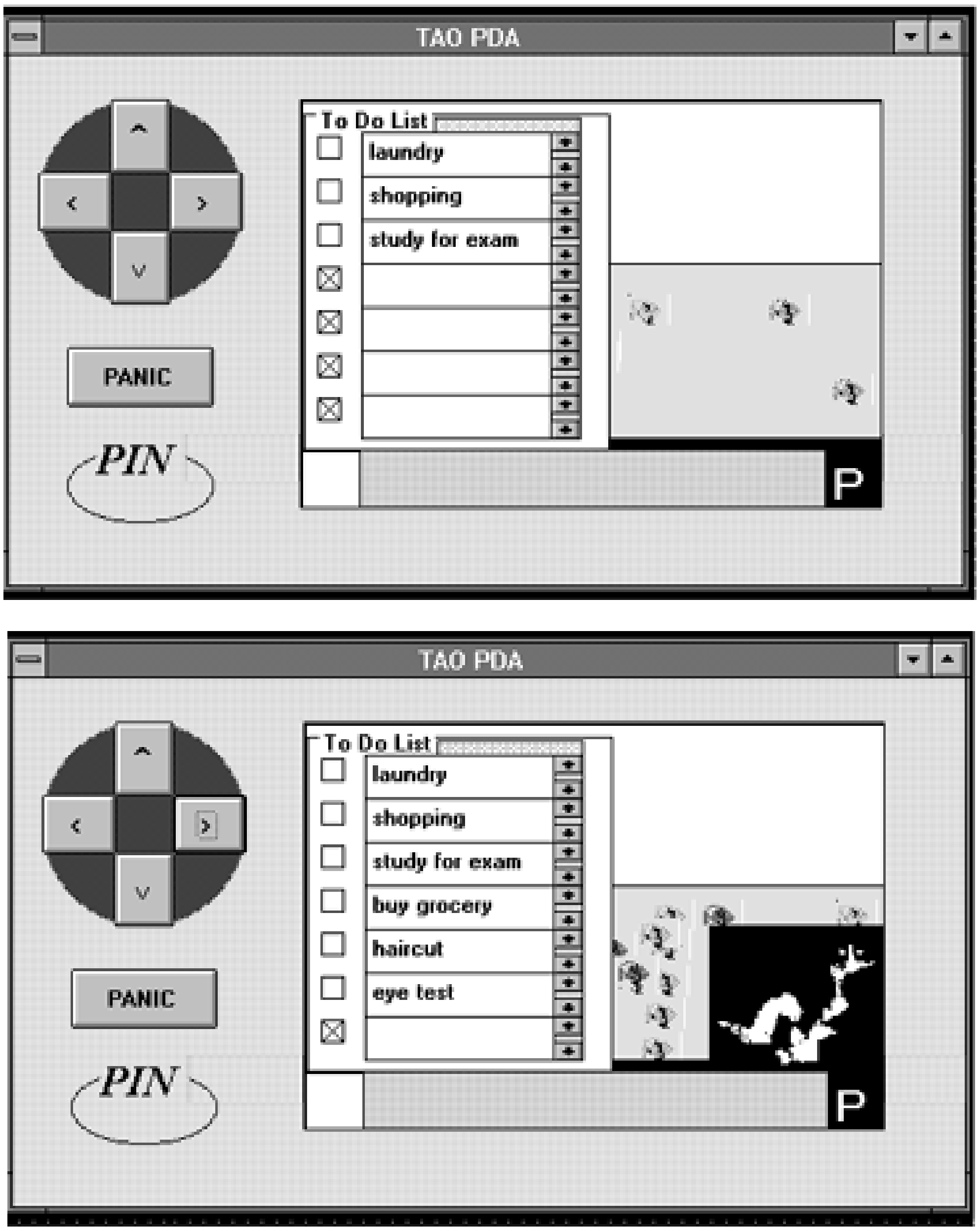
Figure 1. A visual sentence whose meaning changes when the icons change their positions is called a location-sensitive visual sentence. The visual sentence (top) has the meaning "The children study in the morning," and (bottom) has the meaning "The children drive to school in the morning." Comparing the two, this example shows how the placement of the "school" icon changes the meaning. Such visual sentences can be used to specify to-do items for the time management personal digital assistant TimeMan.

Informally, a visual language is a set of visual sentences, each of which is the spatial composition of icons from the set *X*, using spatial relational operators from the set *OP*.

Representing Meaning. To represent the meaning of an icon, we use either a frame or a conceptual graph, depending on the underlying semantic model of the application system being developed. Both are appropriate representations of meaning, and can be transformed into one another. For example, the `SCHOOL_HOUSE` icon in Fig. 1(top) can be represented by the following frame:

```
Icon SCHOOL_HOUSE
WHO: nil
DO: study
WHERE: school
WHEN: nil
```

In other words, the SCHOOL_HOUSE icon has the meaning "study" if it is in the DO location, or the meaning "school" in the WHERE location. Its meaning is "nil" if it is in the WHO or WHEN location. An equivalent linearized conceptual graph is as follows:



TAO PDA

To Do List

□ laundry

□ shopping

□ study for exam

☒

☒

☒

☒

TAO PDA

To Do List

□ laundry

□ shopping

□ study for exam

□ buy grocery

□ haircut

□ eye test

☒

Figure 2. Content-sensitive visual sentences (top) and (bottom) show the fish tank and cat metaphor for the time management personal digital assistant TimeMan. Each fish represents a to-do item. When the to-do list grows too long, the fish tank is overpopulated and the cat appears. The fish tank icon and cat operation icon have corresponding index cells receiving messages from these icons when they are changed by the user.

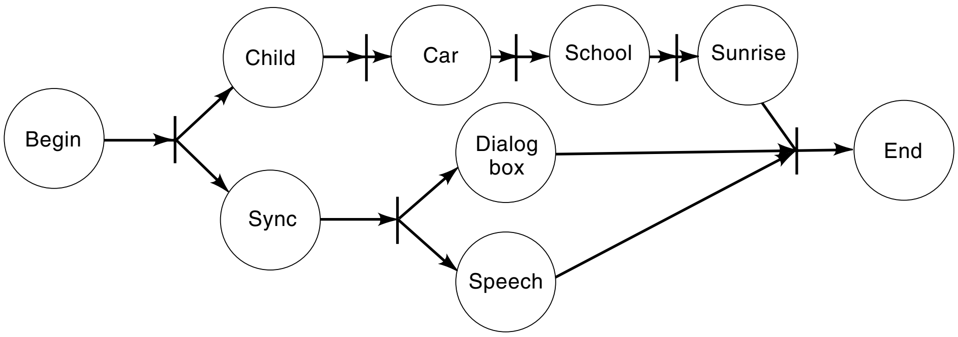


Figure 3. A time-sensitive visual sentence for the Petri net controlling the presentation of the visual sentence shown in Fig. 1(bottom).

```
[Icon = SCHOOL_HOUSE]
  --(sub)--> [WHO = nil]
  --(verb)-> [DO = study]
  --(loc)--> [WHERE = school]
  --(time)-> [WHEN = nil]
```

The meaning of a composite icon can be derived from the constituent icons, if we have the appropriate inference rules to combine the meanings of the constituent icons. Conceptual dependency theory can be applied to develop inference rules to combine frames (4). Conceptual operators can be used to combine conceptual graphs (5). As a simple example, the merging of the frames for the icons in the visual sentence shown in Fig. 1(top) will yield the frame:

```
Visual_Sentence vs1
WHO: children
DO: study
WHERE: nil
WHEN: morning
```

We can derive this frame by merging the frames of the four icons using the following rule:

The ith slot gets the value from the corresponding slot of the ith icon.

Thus, the first slot with slot_name WHO gets the value "children" from the corresponding slot of the first icon CHILDREN, the second slot with slot_name DO gets the value "study" from the corresponding slot of the second icon SCHOOL_HOUSE, and so on.

For visual sentences that are directed graphs, the syntax and semantics can be specified using various kinds of graph grammars. Graph grammars can be used to define the concrete and the abstract syntax of visual languages, but the problem of efficient parsing of visual sentences based upon graph grammars still requires the continued effort of researchers, because most graph parsers work in exponential time. As a starting place for further study, (6) presents layered graph grammar and its parsing algorithm, and also surveys various graph parsing algorithms.

EXTENDING VISUAL LANGUAGES FOR MULTIMEDIA

Visual languages, which let users customize iconic sentences, can be extended to accommodate multimedia objects, letting users access multimedia information 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 a virtual library.

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. We have since extended the framework to include the design of multidimensional languages—visual 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 multidimensional language by combining these icons and have direct access to multimedia information, including animation.

We have successfully implemented this framework in developing BookMan, 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 (7). We applied this approach to emergency management, where the information system must react to flood warnings, fire warnings, and so on, to present multimedia information and to take actions (8). An Active Medical Information System was also developed based upon this approach (9).

Figure 4 shows the search and query options available with BookMan. 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). Much of this power stems from the use of Teleaction Objects (TAOs).

Teleaction Objects

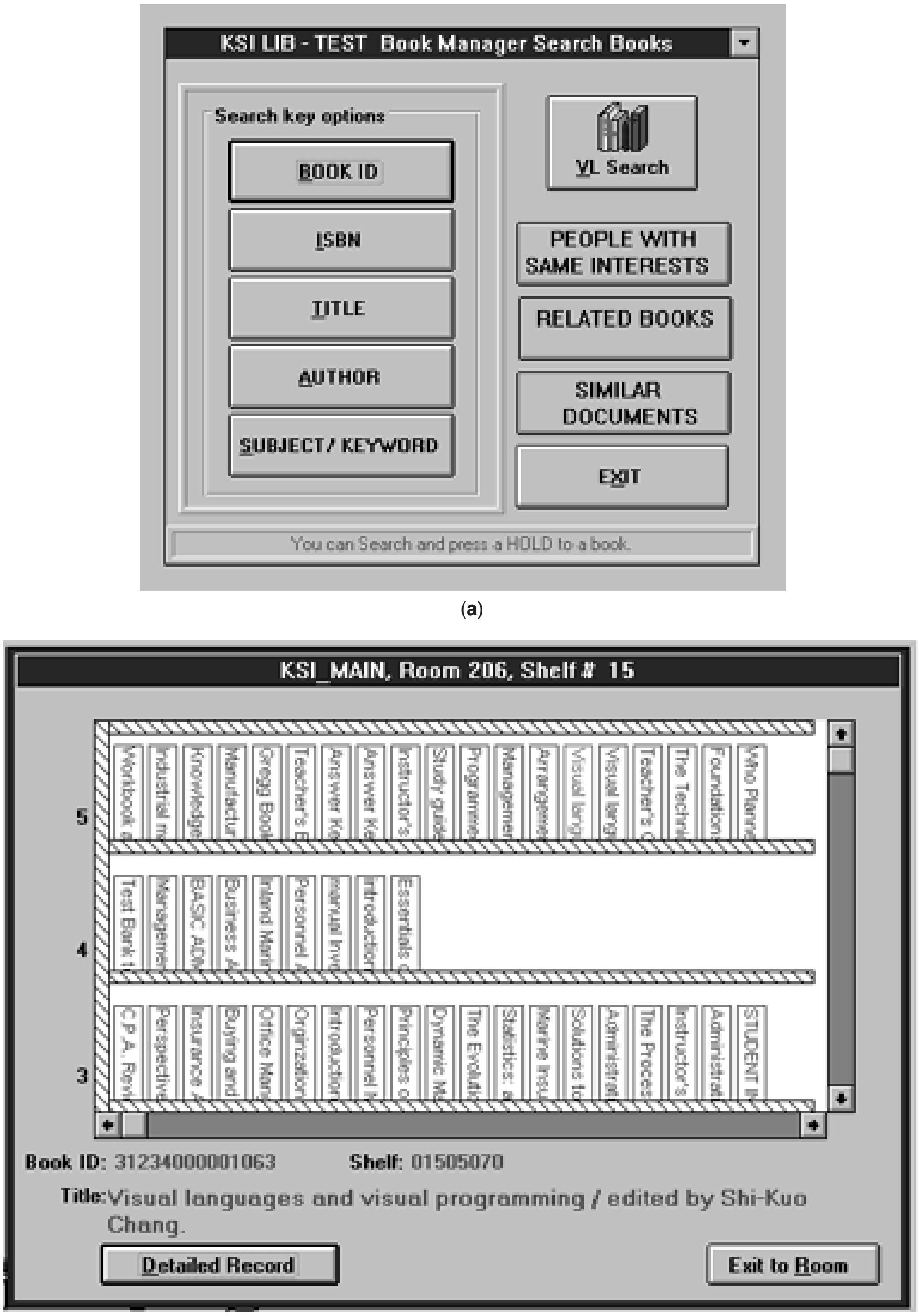
To create a TAO, we attached knowledge about events to the structure of each multimedia object—a complex object that comprises some combination of text, image, graphics, video, and audio objects. TAOs are valuable because they greatly improve the selective access and presentation of relevant multimedia 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 a wide 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 external appearance of a TAO is an event that causes the active index to react. As we will describe later, the active index itself can be designed using a visual-language approach.

Generalized Icons and Multimedia Operators

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



You can Search and press a HOLD to a book.

(a)

Book ID: 31234000001063 Shelf: 01505070

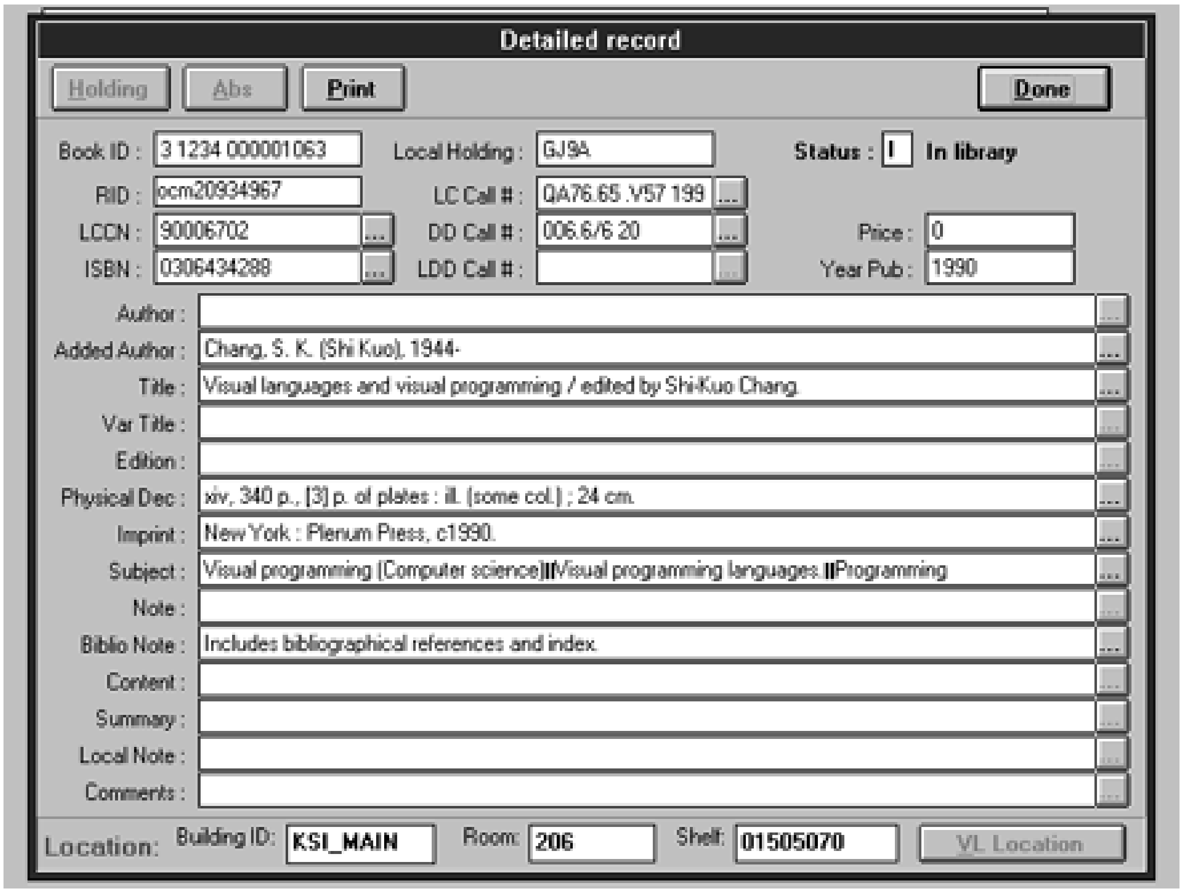
Title: Visual languages and visual programming / edited by Shi-Kuo Chang.

(b)

Figure 4. The virtual library BookMan lets the user (a) select different search modes, (b) browse the virtual library and select desired book for further inspection, and (c) switch to a traditional form-based query mode.

The “Elements of Visual Languages” section described the icons and operators in a visual (not multidimensional) language. In a multidimensional language, we need 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, the physical appearance x_p may 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



Book ID : 3 1234 000001063
Local Holding : GJ9A
Status : I In library
RID : ocm20934967
LC Call # : QA76.65 .V57 199
LCCN : 90006702
DD Call # : 006.6/6 20
Price : 0
ISBN : 0306434288
LDD Call # :
Year Pub : 1990
Author :
Added Author : Chang, S. K. (Shi Kuo), 1944-
Title : Visual languages and visual programming / edited by Shi-Kuo Chang.
Var Title :
Edition :
Physical Dec : xiv, 340 p., [3] p. of plates : ill. (some col.) ; 24 cm.
Imprint : New York : Plenum Press, c1990.
Subject : Visual programming (Computer science)‖Visual programming languages.‖Programming
Note :
Biblio Note : Includes bibliographical references and index.
Content :
Summary :
Local Note :
Comments :
Location: Building ID: KSI_MAIN Room: 206 Shelf: 01505070

(c)

Figure 4. (*Continued*)

- 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 we described previously, 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 (10) among generalized icons. For example, if one wants to watch a video clip and at the same time listen to the audio, one 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 temporal operators are used 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, so that the resulting multidimensional language can fully reflect all multimedia types.

Multidimensional Language

Multidimensional languages can handle temporal as well as spatial operators. As we described in the Elements of Visual Languages section, a visual language has a grammar, $G = (N, X, OP, s, R)$.

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.

Figure 1(bottom) without the dialog box illustrates a simple visual sentence, which describes the to-do item for TimeMan. With the dialogue box, the figure becomes a multidimensional sentence used by TimeMan 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 5 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.

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.

Transformation rules for spatial and temporal operators can be defined to transform the hypergraph in Fig. 5 to a Petri net that controls the multimedia presentation. Figure 3 represents the Petri net of the sentence in Fig. 1(bottom). As such, it is also a representation of the dynamics of the multidimensional sentence in Fig. 1(bottom). The multimedia presentation manager can execute this Petri net dynamically to create a multimedia presentation (11). For example, the presentation manager will produce the visual sentence in Fig. 1(bottom) as well as the synthesized speech.

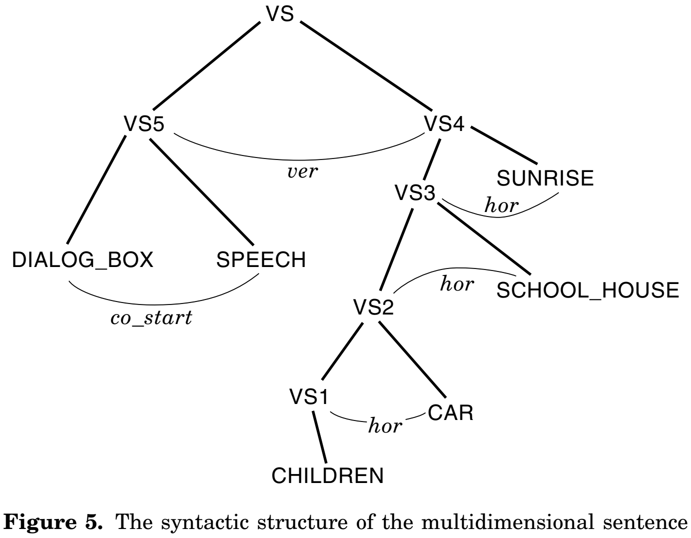


Figure 5. The syntactic structure of the multidimensional sentence shown in Fig. 1(bottom). This structure is a hypergraph because some relational operators may correspond to lines with more than two end points.

VISUAL PROGRAMMING LANGUAGES

Visual programming is programming by visual means. Typically, a programmer or an end user employs some visual programming tool to define and/or construct basic software components such as cells, circuits, blocks, and so on and then put these components together to compose a visual program. The constructed visual program is then interpreted and executed by a visual programming system.

The basic software components can be defined by the programmer/user or obtained from a predefined software component library. Each software component has a visual representation for ease of comprehension by the user. Therefore, software components are generalized icons, and a visual program is a visual sentence composed from generalized icons that are software components. Since the software components are connected together to form a visual program, a visual program can be represented by graph where the basic components in the graph may have multiple attachment points. Examples of commercially available visual programming systems include Prograph which is an object-oriented programming language with dataflow diagrams as its visualization (12), LabVIEW which supports the interconnections of boxes representing software/hardware components (13), and others.

Visual programming is appealing because the programmer or end user can easily manipulate the basic software components and interactively compose visual programs with the help of visual programming tools. Some would claim that visual programming is more intuitive and therefore simpler than traditional programming. Some would further claim that even untrained people can learn visual programming with little effort. However such claims remain to be proven, especially for large-scale software development (14).

As described in the previous two sections, visual languages and multidimensional languages are useful in specifying the syntactic structure, knowledge structure, and dynamic behavior of complex multimedia objects such as TAOs (teleaction objects). We can also construct visual programs using active index cells, which are the key elements of TAOs (15). Without the active index cell, a TAO would not be able to react to events or messages, and the dynamic visual language would lose its power. As an example of visual programming, we can specify index cells using a visual programming tool to be described in a later section. The index cells can thus be connected together as a visual program to accomplish a given task.

Index Cells as Basic Components for Visual Programming

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, the user 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 reinitialized 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 reinitialize 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 reinitialized 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.

A Visual Programming Tool for Index Cell Construction

To aid multimedia application designers in constructing index cells, we developed a visual programming tool, IC Builder, and used it to construct the index cells for BookMan. Figure 6 shows a prefetch index cell being built. Prefetch is used with two other index cell types to retrieve documents (15). If the user selects the prefetch mode of BookMan, 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 6(a) 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(b) shows the result of clicking on the input message icon. 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(c) shows what happens if the designer clicks on the output message icon in Figure 6(a). 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). 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.

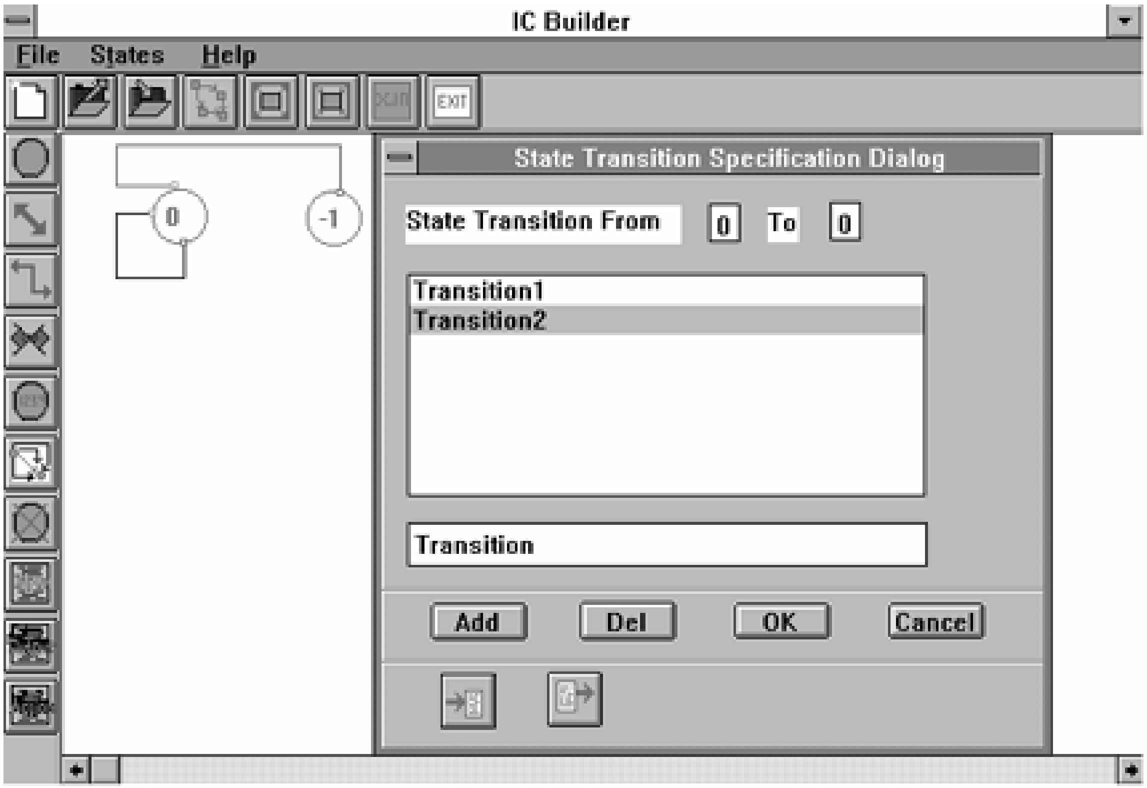
The construction of active index from index cells is an example of visual programming for general purpose problem solving—with appropriate customization the active index can do almost anything. In the following, we will describe a special application of visual programming to database querying.

Visual Queries

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.

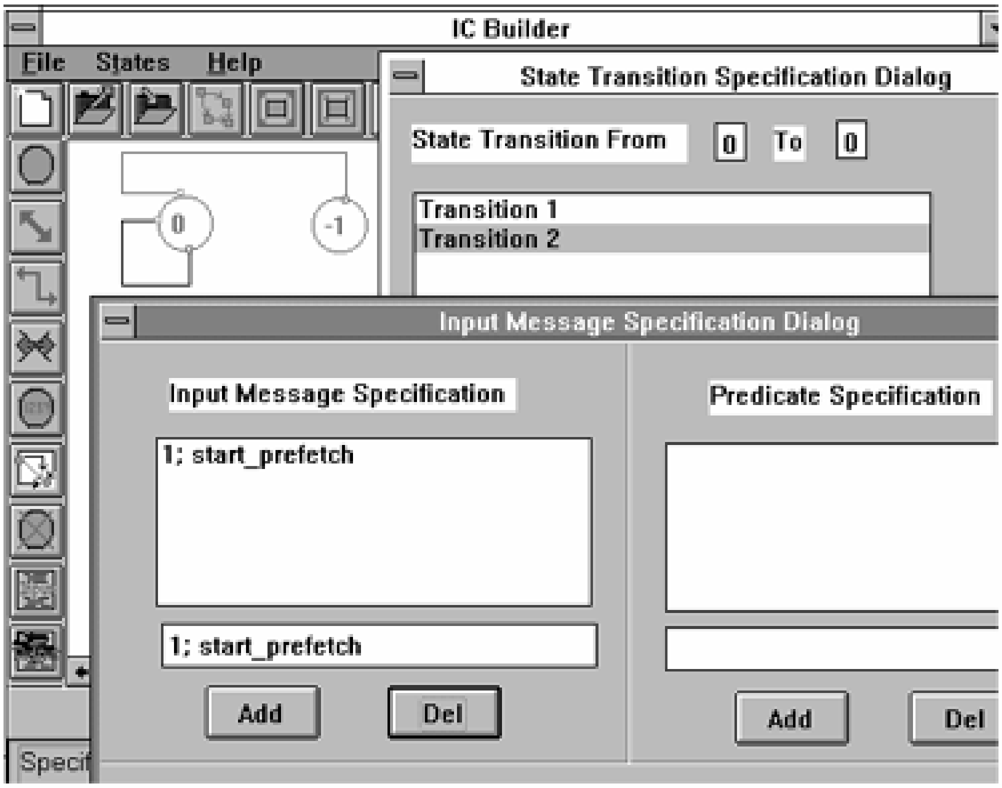
A visual sentence or multidimensional sentence can also be either location-sensitive, time-sensitive, or content-sensitive. In the first section we gave examples of different types of vi-



Transition1
Transition2

Transition

(a)


IC Builder
File
States
Help
0
-1
State Transition Specification Dialog
State Transition From
0
To
0
Input Message Specification Dialog
Input Message Specification
Predicate Specification
Add
Del
Add
Del
Specif

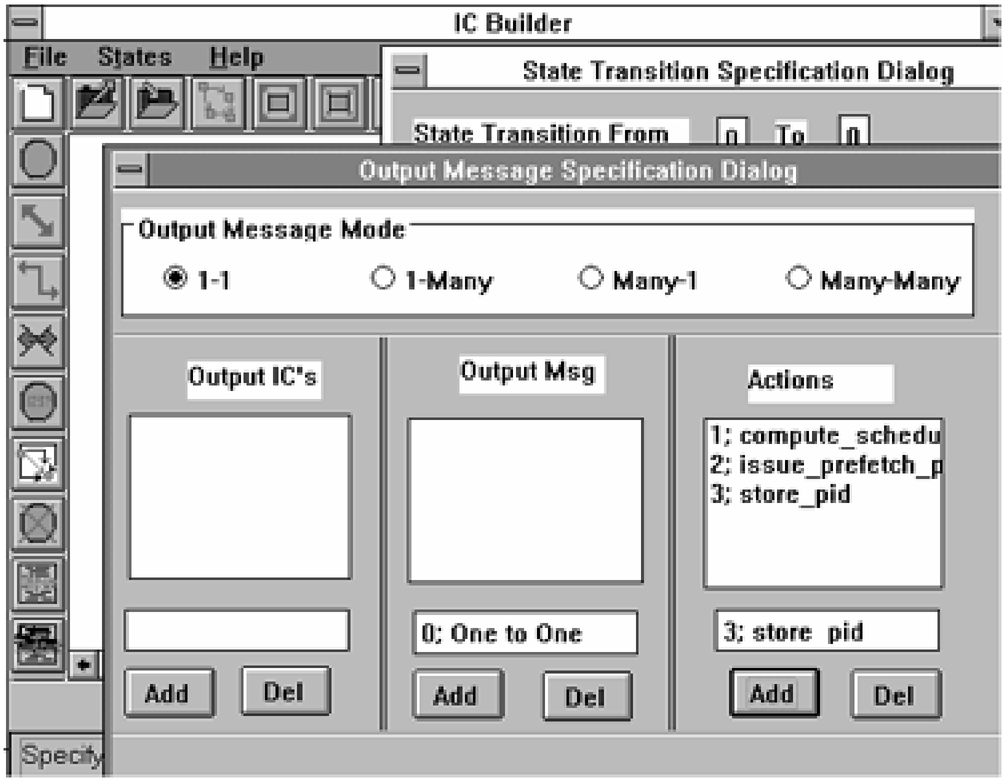
Transition 1
Transition 2

1; start_prefetch

1; start_prefetch

(b)

Figure 6. The visual specification for an active index cell of the virtual library BookMan: (a) the state transitions, (b) input message, (c) output message and actions.



State Transition From 0 To 0

Output Message Mode

● 1-1 ○ 1-Many ○ Many-1 ○ Many-Many

1; compute_schedu
2; issue_prefetch_p
3; store_pid

0; One to One

3; store pid

(c)

Figure 6. (*Continued*)

sual sentences. The resulting language is a dynamic visual language or dynamic multidimensional language.

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 (16). When the user initially browses the virtual library, the virtual reality (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 Fig. 4(b) shows, BookMan indicates the physical locations of books by marked icons in a graphical presentation of the book stacks of the library. What users see is very similar (with some simplification) to 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 Fig. 4(a), 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, the user can then navigate in the virtual library, and as shown in Fig. 4(b), the result is a marked object. If the user switches to a form-based representation by clicking the DetailedRecord button, the result is a form as shown in Fig. 4(c). 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 Fig. 4(c).

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 admissable 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.”

Visual query systems for multimedia databases, like BookMan, are under active investigation at many universities as well as industrial laboratories (17). These systems are very 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 BookMan we have a mechanism that lets users create similarity retrieval requests that prompt BookMan to look for books similar to the book being selected, and then perform searches on the World Wide Web using a Web browser enhanced with an active index (18).

CONCLUDING REMARKS

Visual languages and visual programming languages are progressing at a rapid pace. Several on-line bibliographies are now available (19,20,21). As far as programming is concerned, visual programming languages may not be appropriate for every situation. An important question is whether visual programming languages can scale up to handle large scale applications (22). Moreover, empirical, systematic evaluation of such languages needs to be done (23).

The average programmer and end user are used to a hybrid mode of human-computer interaction, involving text, graphics, sound, and the like. Thus, "pure" visual programming languages are sometimes hard to justify. On the other hand, languages allowing hybrid mode of interactions are already unavoidable, due to the explosion of multimedia computing and network computing. As multimedia applications become even more widespread, we expect to see more special-purpose or general-purpose visual language systems and visual programming systems in which visual and multidimensional languages will play an important role, both as a theoretical foundation and as a means to explore new applications.

ACKNOWLEDGMENT

This research was supported in part by the National Science Foundation under grant IRI-9224563.

BIBLIOGRAPHY

1. S. K. Chang et al., Visual language system for user interfaces, *IEEE Softw.,* **12** (2): 33–44, 1995.
2. S. K. Chang, A visual language compiler for information retrieval by visual reasoning, *IEEE Trans. Softw. Eng.,* **16**: 1136–1149, 1990.
3. C. Crimi et al., Automating visual language generation, *IEEE Trans. Softw. Eng.,* **16**: 1122–1135, 1990.
4. S. K. Chang et al., A methodology and interactive environment for iconic language design, *Int. J. Human-Computer Studies,* **41**: 683–716, 1994.
5. S. K. Chang et al., A visual language compiler, *IEEE Trans. Softw. Eng.,* **5**: 506–525, 1989.
6. J. Rekers and A. Schuerr, Defining and parsing visual languages with layered graph grammars, *J. Visual Languages Comput.,* **8** (1): 27–55, 1997.
7. H. Chang et al., Management and applications of tele-action objects, *ACM Multimedia Syst. J.,* **3** (5–6): 204–216, 1995.
8. Y. Khalifa, S. K. Chang, and L. Comfort, A prototype spatial-temporal reasoning system for emergency management, *Proc. Int. Conf. Visual Inf. Syst. VISUAL96,* Melbourne, Australia, pp. 469–478, 1996.
9. S. K. Chang, [on-line] Available www: .cs.pitt.edu/~jung/AMIS2
10. J. F. Allen, Maintaining knowledge about temporal intervals, *Commun. ACM,* **26** (11): 832–843, 1983.
11. C. C. Lin, J. X. Xiang, and S. K. Chang, Transformation and exchange of multimedia objects in distributed multimedia systems, *ACM Multimedia Syst. J.,* **4** (1): 2–29, 1996.
12. *Prograph CPX User's Guide,* Pictorius Incorporated, 1993.
13. E. Baroth and C. Hartsouth, Visual programming in the real world, in M. Burnett, A. Goldberg and T. Lewis, eds, *Visual Object-Oriented Programming Concepts and Environments,* Greenwich, CT: Manning Publications, 1995, pp. 21–42.
14. K. N. Whitley, Visual programming languages and the empirical evidence for and against, *J. Visual Languages Comput.,* **8** (1): 109–142, 1997.
15. S. K. Chang, Towards a theory of active index, *J. Visual Languages Comput.,* **6** (1): 101–118, 1995.
16. S. K. Chang, M. F. Costabile, and S. Levialdi, Reality bites—progressive querying and result visualization in logical and VR spaces, *Proc. IEEE Symp. Visual Languages,* St. Louis, October 1994, pp. 100–109.
17. T. Catarci et al., Visual query systems for data bases: a survey, *J. Visual Languages Comput.,* **8** (2): 215–260, 1997.
18. S. K. Chang, [on-line] Available www.cs.pitt.edu/~jung/WAG
19. M. Burnett, [on-line] Available http://www.cs.orst.edu/~burnett/vpl.html
20. R. Korfhage, [on-line] Available www.pitt.edu/~korfhage/vlrefs.html
21. S. Schiffer, [on-line] Available http://www.swe.uni-linz.ac.at/schiffer/buch/literatur.htm.
22. M. Burnett et al., Scaling up visual programming languages, *Computer* **28** (3), *IEEE CS Press,* 45–54, March 1995.
23. J. D. Kiper, E. Howard, and C. Ames, Criteria for evaluation of visual programming languages, *J. Visual Languages Comput.,* **8** (2): 175–192, 1997.

Shi-kuo Chang
University of Pittsburgh