

# TOWARDS A THEORY OF ACTIVE INDEX

Shi-Kuo Chang

Visual Computer Laboratory

Department of Computer Science, University of Pittsburgh  
Pittsburgh, PA 15260 USA (Email: chang@cs.pitt.edu)

**ABSTRACT:** This paper introduces a theoretical framework for the active index. The active index facilitates the accessing and automatic manipulation of visual objects. With an active index, we can effectively and efficiently handle smart images that respond to accessing, probing and other actions. Another important application is information retrieval in hyperspace. The active index can also be used to realize Petri nets, generalized Petri nets such as G-nets, B-trees, etc., but the dynamic nature of the active index is its most important characteristic. Implementation considerations and research issues are discussed.

**KEYWORDS:** Active index, feature-based index, smart images, visual information systems, information retrieval in hyperspace.

## 1. INTRODUCTION

In visual computing, an important issue is how to index visual objects, so that the visual objects can be accessed quickly and certain actions can be performed automatically. In conventional database systems, keyword-based indexing techniques are adequate to support users' needs. In visual information systems, there are many applications that cannot be properly supported by keyword-based techniques. In addition to keywords, users often want to access/manipulate visual objects by shape, texture, spatial relationships, etc. Iconic Indexing Using Generalized Pattern Matching Techniques Smart Image Design for Large Image Databases Shape Matching Utilizing Indexed Hypothesis A Hierarchical Approach to Feature Indexing A Geometric Indexing Schema Tagare That is, certain features of the visual objects are used as indexes, and, in many cases, they cannot be represented as keywords. The representation of these feature-based indexes poses some special problems:

- (1) Indexes are approximately represented.
- (2) Indexes do not have an implicit ordering, in the sense that if  $a$ ,  $b$  and  $c$  are three index values and  $a < b < c$ , it does not mean that visual object  $b$  is more similar to visual object  $a$  than visual object  $c$ .
- (3) Indexes may have interrelated multiple attributes. That is, if  $a_1$  and  $a_2$  are two attributes of an index,  $\text{result}(a_1 \text{ op } a_2)$  is not equal to  $\text{result}(a_1) \text{ op } \text{result}(a_2)$ .

Faced with these problems, the conventional indexing structures such as B-tree, hashing, etc. cannot be used for the organization of indexes for visual objects. New indexing structures must be explored which should also support similarity retrieval. Klinger Pizano Visual Structure Jagadish Retrieval Techniques for Similar Shapes Moreover, the index structures should be highly flexible and dynamic, with the following characteristics:

- (a) **Active index instead of Passive index:** The index can be used to perform actions.
- (b) **Partial index instead of Total index:** Only a few visual objects are indexed.
- (c) **Dynamic index instead of Static index:** The index can evolve, grow and shrink.
- (d) **Visible index instead of Transparent index:** The user is aware of the existence of the index, perhaps as part of the knowledge structure. So the index is not necessarily transparent.
- (e) **Imprecise index instead of Precise index:** The index can be used in processing imprecise or approximate queries.

This paper introduces a theoretical framework for the active index, based upon and extending our previous formulation of the active index structure. Smart Image Design for Large Image Databases The theoretical framework is introduced in Section 2. To illustrate its application to the Smart Image System, in Section 3 we present a three-level active index. With an active index, we can effectively and efficiently handle smart images that can respond to accessing, probing, and other actions. The application to information retrieval in hyperspace is discussed in Section 4. The computation power of the active index is analyzed in Section 5. The active index can be used to realize Petri nets, generalized Petri nets such as G-nets, B-trees, etc., but the dynamic nature of the active index makes it even more powerful and flexible. The reversible index introduced in Section 6 facilitates feature-based indexing. An experimental active index system has been implemented, whose main features are described in Section 7. In Section 8, further research topics are discussed.

## 2. FORMAL DEFINITION OF THE ACTIVE INDEX

An index cell base (ICB) consists of a (possibly infinite) number of index cells. An index cell (ic) accepts input messages and performs some computation. It then activates another group of index cells, and posts the output message to these output index cells. If some of these output index cells have already been activated, they may simply accept the output from the current index cell. The first output cell that accepts the output message will remove it from the output list of the current cell. After its computation, the index cell may remain active (live), or de-activate itself (dead). An index cell will also become dead, if it remains inactive for a certain period of time, i.e., if no other cells (including itself) send messages to it.

An active index (IX) consists of a finite number of index cells ic from ICB. Thus an active index IX is a finite subset of the (possibly infinite) index cell base ICB. When the active index is in actual computation, it consists of a time-varying collection of index cells in different states, accepting certain input messages and posting output messages to the output lists. To describe precisely the behavior of the active index, we will first formally define an index cell.

**Definition 1:** An index cell is described by  $ic = (X, Y, S, s_o, A, t_{max}, f, g)$  where:

$X$  is the (possibly infinite) set of input messages including dummy input  $d$ .  $Y$  is the (possibly infinite) set of output messages including dummy output  $d$ .  $S$  is the (possibly infinite) set of states.  $S$  includes a set of ordinary states  $\mathbf{S}$  and a special state  $s_{\text{dead}}$  called the dead state. If an index cell is in the dead state, it is a dead index cell. Otherwise it is a live index cell.  $s_0$  in  $\mathbf{S}$  is the initial state of the index cell  $ic$ .  $A$  is the set of action sequences that can be performed by this index cell.  $t_{\text{max}}$  is the maximum time for the cell to remain live, without receiving any messages. If  $t_{\text{max}}$  is infinite, the cell is perennial.  $f$  is a function:  $2^X \times \mathbf{S} \rightarrow \{0,1\}$  where  $2^X$  is the power set of input  $X$ . If  $f(\{x_1, \dots, x_m\}, s)$  is 1, then the cell accepts the input set  $\{x_1, \dots, x_m\}$  and  $x_1, \dots, x_m$  are removed from the output lists of those cells that produce these output messages. The removal of messages is an atomic action which will occur simultaneously. If  $f(\{x_1, \dots, x_m\}, s)$  is 0, the input messages are not accepted. When several input sets can be accepted, one is chosen non-deterministically.  $g$  is a function:  $2^X \times \mathbf{S} \rightarrow 2^{\text{ICB} \times Y \times \mathbf{S} \times A}$  such that given input messages  $\{x_1, \dots, x_m\}$  which have been accepted, i.e.,  $f(\{x_1, \dots, x_m\}, s) = 1$ , and current state  $s$ ,  $g(x, s)$  is a quadruple  $(I_c, y, s', a)$  where

- (1)  $I_c$  is a set of output index cells to be activated. If an output index cell is in the dead state, it is changed to the initial state so that it becomes a live cell, and the clock  $t$  is initialized to be  $t_{\text{max}}$ . If an output index cell is already live, its current state remains unchanged, but its clock  $t$  is re-initialized to be  $t_{\text{max}}$ . If an output index cell is the special symbol  $nil$ , no output index cell is actually activated.
- (2)  $y$  is the output message for the output index cells in  $I_c$ . The output could be the dummy message  $d$ , when there is no real output to the output index cells. The first output index cell that accepts this output message  $y$  will remove it from the output list of  $ic$ .
- (3)  $s'$  is the computed next state of  $ic$ . The true next state  $s''$  of  $ic$  is the dead state if clock time  $t$  becomes zero or negative, and  $s'$  otherwise. If the next state  $s'$  is the dead state, the index cell becomes dead.
- (4)  $a$  is the action-sequence performed by this index cell, which can be regarded as the output of the cell to the external environment.

**Definition 2:** The *output list*  $oL$  of an  $ic$  is of the following form:  $[(I_{c_1}, y_1), (I_{c_2}, y_2), \dots, (I_{c_m}, y_m)]$ , where  $y_i$  is the output message posted to the  $ic$ 's in the set  $I_{c_i}$ . If any  $ic$  in  $I_{c_i}$  accepts  $y_i$ , the tuple  $(I_{c_i}, y_i)$  is removed from the output list.

**Definition 3:** An *index cell base*  $ICB$  is a (possibly infinite) collection of index cells. Given an index cell base  $ICB$ , an *active index*  $IX$  is a finite subset of  $ICB$  with  $n$  index cells, denoted by an  $n$ -place  $ic$  vector  $\langle ic \rangle = (ic_1, ic_2, \dots, ic_n)$ , where the  $ic$ 's are ordered by their (arbitrary) subscripts in  $ICB$ .

**Definition 4:** The *instantaneous description*  $id$  of an active index  $IX$  is denoted by  $id = (\langle ic \rangle, \langle s \rangle, \langle oL \rangle)$ , where  $\langle ic \rangle$  is the  $ic$  vector,  $\langle s \rangle$  is the corresponding state vector, and  $\langle oL \rangle$  is the corresponding output list vector.

**Definition 5:** The *trace* of an active index  $IX$  with respect to  $(\langle ic_0 \rangle, \langle s_0 \rangle, \langle oL_0 \rangle)$  is:

$$\begin{aligned} &(\langle ic_0 \rangle, \langle s_0 \rangle, \langle oL_0 \rangle) \Rightarrow \\ &(\langle ic_1 \rangle, \langle s_1 \rangle, \langle oL_1 \rangle) \Rightarrow \\ &\dots \\ &(\langle ic_n \rangle, \langle s_n \rangle, \langle oL_n \rangle) \end{aligned}$$

where  $(\langle ic_i \rangle, \langle s_i \rangle, \langle oL_i \rangle) \Rightarrow (\langle ic_{i+1} \rangle, \langle s_{i+1} \rangle, \langle oL_{i+1} \rangle)$  due to the acceptance of input messages by an index cell. The  $\Rightarrow$  symbol reads as "is transformed into". Such transformations may occur in any arbitrary order. Each transformation step in the trace takes exactly one clock cycle. If the trace is finite, the active index  $IX$  is *terminating* with respect to  $(\langle ic_0 \rangle, \langle s_0 \rangle, \langle oL_0 \rangle)$ ; otherwise it is *nonterminating*.

Therefore, an active index is initially specified by  $(\langle ic_0 \rangle, \langle s_0 \rangle, \langle oL_0 \rangle)$  where  $\langle ic_0 \rangle$  is the initial  $ic$  vector,  $\langle s_0 \rangle$  is the initial state vector and  $\langle oL_0 \rangle$  is the initial output list vector.

For example, we can start with a single index cell, so that the active index initially starts with  $(ic_0, s_0, oL_0)$ , where  $s_0$  is the initial state of  $ic_0$ , and the output list  $oL_0$  is empty. Let  $f(\{\}, s_0)$  be 1, so that  $ic_0$  will accept an empty set as input. Let  $f(V, s_0)$  be 0 for any non-empty  $V$ . If we intend to activate index cells in  $I_c$  and post a message  $y$  to them, it can be done by an appropriate  $g$  function. After that,  $ic_0$  enters a state  $s_{\text{sleep}}$ , where no input will be accepted. The  $t_{\text{max}}$  can be set to infinity. In other words, the sole purpose of  $ic_0$  is to activate some index cells and post a message to them. By adding states to  $ic_0$  appropriately, we can also make  $ic_0$  post individual messages to each of the activated index cells.

**Observation 1:** An index cell  $ic$  can be modified to post  $n$  messages individually to  $n$  output index cells.

**Proof:** Suppose  $f(\{x_1, \dots, x_m\}, s) = 1$  and we want to post messages  $y_i$  individually to  $ic_i$ ,  $1 \leq i \leq n$ , and then change state to  $s'$ . Let  $s_1, s_2, \dots, s_{n-1}$  be  $n-1$  new states. Replace the original  $g(\{x_1, \dots, x_m\}, s) = (I_c, y, s', a)$  by the following:  $g'(\{x_1, \dots, x_m\}, s) = (\{ic_1\}, y_1, s_1, a)$

We can construct  $f'$  and  $g'$  as follows:

$$\begin{aligned} f'(\{\}, s_1) &= 1 \text{ and } f'(\{\}, s'') = 0 \text{ if } s'' \neq s_1 \\ g'(\{\}, s_1) &= (\{ic_2\}, y_2, s_2, nil) \\ f'(\{\}, s_2) &= 1 \text{ and } f'(\{\}, s'') = 0 \text{ if } s'' \neq s_2 \\ g'(\{\}, s_2) &= (\{ic_3\}, y_3, s_3, nil) \\ &\dots \end{aligned}$$

$$f'(\{ \}, s_{n-1}) = 1 \text{ and } f'(\{ \}, s'') = 0 \text{ if } s'' \neq s_{n-1}$$

$$g'(\{ \}, s_{n-1}) = (\{ic_n\}, y_n, s', nil)$$

Therefore, the  $ic$  will go through the states  $s_1, \dots, s_{n-1}$  and post the messages individually to the output  $ic$ 's, and then change state to  $s'$ .

#### Q.E.D.

**Notation 1:** As a notational convenience, we will write the quadruple as  $(Ic, W, s', \mathbf{a})$ , where the cardinality of  $Ic$  and  $W$  must be identical, i.e.,  $Ic = \{ic_1, \dots, ic_n\}$ ,  $W = \{y_1, \dots, y_n\}$ , to indicate that each  $y_i$  is posted to each  $ic_i$  individually.

**Notation 2:** As a further notational convenience, we will allow  $Ic$  and  $W$  to be lists. If  $Ic$  is of the form  $[ic, \dots, ic]$ , this means the messages  $y_i$  are all posted to the same  $ic$ . If  $W$  is of the form  $[y, \dots, y]$ , this means the same message  $y$  is posted to each  $ic_i$  individually.

The above notation enables us to specify the posting of a message  $y$  either to an individual  $ic$ , or to a group of  $ic$ 's. In particular, a message can be posted to a certain *type* of  $ic$ , if we do not yet know the identity of the individual  $ic$ .

The external environment may also send messages to the active index. In particular, the action sequence may cause the external environment to send messages to some of the index cells, including the index cell that performs the said action sequence. This can be modeled by activating a special  $ic$ , similar to the  $ic_0$  described above, to send messages to some of the index cells.

### 3. THE ACTIVE INDEX FOR THE SMART IMAGE SYSTEM

An active index is a dynamically changing net. As we shall see in Section 5, active index can be used to realize Petri nets, generalized Petri nets (G-nets), B-trees, etc. But its primary purpose is to serve as a dynamic index. We now illustrate by an example.

In current visual information systems, images don't have the capabilities to automatically respond to situational changes occurred in their environments. With advances in software and hardware technologies, images can play a more active role in their applications. For example, in the medical domain, after the examination of a patient's nuclear image, a doctor may want to compare images of the same patient at different states (exercising, normal, excited, etc.), then to examine images in the time domain (past histories), and finally to check images from other modalities. Instead of having the doctor to retrieve these relevant images with explicit queries and to convert and highlight the images properly, an active image can monitor the doctor's actions and provide the necessary information in proper formats on time. To improve the effectiveness and efficiency of visual information systems, images should invoke actions by themselves. Depending upon applications, they can move themselves into proper local storage, pre-process themselves into the appropriate representations, and display themselves on the screen at the right time.

A *smart image* is an image with an associated knowledge structure, where knowledge includes attributes, routine procedures for how the image is used, and dynamic links to other objects for performing related actions. A smart image knows what actions to take based on the user's interaction with the image and on the environmental changes to the images. Smart Image Design for Large Image Databases To illustrate how the active index can be applied to the Smart Image System, let us describe the 3-level active index for the Smart Image System, using the theoretical framework presented in Section 2.

**3.1. Level-1 Index:** The level-1 index is to pre-perform certain operations and specifically to prefetch image data. For each smart image, only one level-1 index cell can be activated. The input to this  $ic$  is the set of user messages of interest. A relevant user message regarding a smart image will be sent to this  $ic$ . This user message will (1) cause the  $ic$  to activate the appropriate level-2  $ic$ , (2) post an output message to that level-2  $ic$ , and (3) change state to the appropriate next state. For example, if the  $ic$  is in  $s_{angio}$  and the user message indicates a "Stenosis" condition, then the  $ic$  will activate the level-2  $ic_{multi-modality}$ , send an output message "Image:Angio, Abnormality:Stenosis" to  $ic_{multi-modality}$ , and change state to  $s_{muga}$ . The action of this  $ic$  in state  $s_{angio}$  is to prefetch all muga images of the patient.

The states in this level-1 index cell are the "global" states. Once the  $ic$  enters a global state, a selected group of the next level index cells can be activated. In the above example, the states correspond to the different *image modalities*, because when the user is viewing an image of a given modality, we know the index cell must be in that state. In other words, such states are observable.

Figure 1 illustrates the relationships among images, hotspots and level-1 active index. The state transitions for the level-1 index cell are given in Figure 2. As illustrated in Figure 2, from the current state, depending upon the user's input message (the condition), we can prefetch all relevant images of a given modality. Thus from State 2, if the condition is "Stenosis", then we prefetch all the muga images of a specific patient and go to State 3. There may be too many muga images to be prefetched. Can we prefetch only a subset of these muga images? It depends on the following: (i) the filtering algorithm, (ii) the way images are organized in the class hierarchy, and (iii) the index cell construction algorithm. Since the level-1 index cell is essentially a finite-state machine, there are effective learning algorithms to construct the cell from the past *history* of user messages. In principle, we can record every click made by the user as well as every text, voice or annotation messages. In practice, we use *filters* to extract the appropriate user messages and record them in the history. A moving window is kept, so that the recent history is used by the learning algorithm to construct the finite-state machine for the index cell. For example, the filtering algorithm may only extract user's identification of abnormality and accessing of image data:

Abnormality="Stenosis", Retrieve="Muga image taken on date-x"

from the following history:

Doctor Name: Douglass A. Young Patient Name: David Straker ID:123-45-6789 SEX:M Date of Birth: Sep 15 1953 Angio Image: Ang.Hrt.001, taken on Dec 19 1991, EID#=MHT-00010 CREATE\_HS Abnormality: Stenosis RETRIEVE\_IMAGE Muga Image: Mug.Hrt.003, taken on Dec 10 1991, EID#=MHT-00030

In the smart image class hierarchy, images are divided into:

- (a) recent images (within one month),
- (b) fairly recent images (within one year), and
- (c) archival images (within ten years).

The simplest index cell will just define a next state corresponding to ALL nuclear images. The more sophisticated index cell will have next states corresponding to (a) (b) and (c). The index cell construction algorithm will test whether date-x satisfies (a) (b) or (c) and then constructs the cell's next state(s). With this refined construction algorithm, only those images that are in (a), (b) or (c) will be prefetched.

**3.2. Level-2 Index:** The level-2 index is to perform hotspot-triggered actions in multi-modality study. If the user will make known to the system what study is being conducted, such as Coronary Artery Disease, Ventricular Function, and so on, the appropriate level-2 index cells will be activated based on the particular study.

A hotspot in a smart image, when triggered, may send messages to a level-2 index cell. For example, the input to an  $ic_{\text{left\_ventricular\_study}}$  is the set of hotspot conditions such as abnormality, and quantitative data obtained from the image processing routine. An appropriate hotspot condition will (1) cause the ic to activate another ic in level-2 or an ic in level-3, (2) post an output to that ic, and (3) change to dead state to de-activate itself.

Figure 3 illustrates the level-2 active index. In Figure 3, the active index is shown as a net of index cells. It should be emphasized that the arcs in this net are dynamic. Output arcs are specified when a live cell accepts and processes the input. They can change dynamically. For example, the hotspot condition: LV\_enlargement\_abnormality, and heart volume quantitative data in nuclear image and the hotspot condition: Stenosis\_abnormality, and low ejection fraction quantitative data in angio image, when triggered, will (1) cause the ic to activate the level-3 ic, (2) post the appropriate output message to the level-3 ic, and (3) de-activate  $ic_{\text{left\_ventricular\_study}}$ . Another hotspot may cause the ic to activate different output cells. By using the technique of abstraction, Pictorial Information Systems Design we can combine the simpler cells into more complex cells with multiple inputs such as for multi-modality study.

Similarly, some image processing functions may require multiple images as input. The detection of LV Enlargement and Stenosis in two different images may require two separate image processing functions. The two functions may be disjoint, and no image fusion is required. We will just test the logical predicates in the above example. On the other hand, for some cases image fusion will be required, and we must register the images, perform nonlinear transformations to correlate images, etc.

**3.3. Level-3 Index:** The level-3 index is to perform automatic linking and the retrieval of related and sometimes unanticipated information. When the user requests information (by clicking on some button), a message is sent to the ic. The input to an ic is therefore the set of retrieval requests. An appropriate retrieval request will (1) cause the ic to activate another (possibly remote) ic, (2) post an output message to that ic, and (3) change to initial state. The action is to send information to the original requester. For example, the  $ic_{\text{tumor}}$ , with the initial input message "tumor\_found", may initiate a retrieval request, to retrieve all related information on that patient, and present it to the original requester (the physician who is interacting with the Smart Image System).

Annotation as illustrated in Figure 4 could be considered "unanticipated information". When an active index automatically performs linking and prefetching operations, unanticipated information can be included or *not* included. When the physician is making a decision, he needs the right amount of unanticipated information, but certainly he does not want every single new case in the medical journals. Thus, the active index with the appropriate action sequence determines what links to be established, and what amount of information to be prefetched. Such flexibility makes the Smart Image System responsive to users' needs.

The function of the level-3 index is quite similar to that of the active index for information retrieval in hyperspace, which will be explained in the following section.

#### 4. THE ACTIVE INDEX FOR INFORMATION RETRIEVAL IN HYPERSPACE

To retrieve information in the hyperspace which is represented by a hyperstructure, we can associate an index cell with every recently accessed node in this hyperstructure. Thus the ICB corresponds to the set of all nodes in the hyperstructure, and IX a finite set of recently accessed nodes.

In a recent experiment, two months of tracing Mosaic usage in a university department show that about 40-45% of Mosaic files are accessed with high frequency. Marino A relevant subset of the Mosaic objects transferred from remote servers are often used by other NCSA Mosaic clients. (In the university environment, the often looked-for information items are call-for-papers, books or technical announcements, new computer systems, etc.) Therefore, to improve the system performance, frequently accessed information items should be prefetched and kept in the local cache.

The index cell can be constructed as follows: It accepts a query  $q_k$  if  $k > 0$ , and activates the adjacent index cells, and posts  $q_{k-1}$  to them. The action performed is to prefetch information items satisfying the query. A further refinement is to prefetch information items above a certain size. The justification is that we need only prefetch large information items, and small information items need not be prefetched.

As an example, if the original query is  $q_3$ , only cells within a distance of two links may be activated. Since we are posting a query to all the adjacent cells, if one of them accepts the query, the rest will no longer be able to process this query. Therefore, only three ic's on the following single path will be activated:

$ic_{q_3} - ic_{q_2} - ic_{q_1}$

If we post queries individually to the adjacent cells, the result is to activate all ic's where the distance from any ic to  $ic_{q_3}$  is no more than two links. Consequently, more information items will be prefetched.

Both the viewer and the designer of a hyperstructure can add knowledge to the index cell as follows. The viewer of the hyperstructure can send messages to the active index. For example, a clicking on a document indicates the invocation of a hyperlink. Certain index cell can then be activated. We may also allow the viewer to add annotation to certain objects. In this case, the viewer can modify (a part of) g

to activate the cell corresponding to the annotation object. The designer of the hyperstructure can of course modify the  $g$  function to decide what new cells to activate. Thus,  $g$  contains the designer's knowledge. Therefore, the  $g$  function is central in capturing both the viewer's knowledge and the designer's knowledge.

The function  $g$  allows us to either add new cells to the system, or to stay with a predefined set of cells. We can set  $t_{\max}$  to infinity, and exclude from  $g$  the dead state, so that index cells always remain live. We can further stipulate that  $g$  maps only to cells in  $IX$ . Thus the system can become a static index system. On the other hand, if no query is posed, with finite  $t_{\max}$ 's after a while all index cells will become dead. In other words, the active index is active, only as long as there are messages sent to the cells (or, to put it simply, only as long as there are users interested in certain information nodes of the hyperstructure).

>From the above two examples, it can be seen that the major difference between an active index and a static index is that the active index is a dynamic collection of live index cells. The active index will change with time, as new index cells are activated and current index cells are de-activated.

## 5. THE COMPUTATION POWER OF THE ACTIVE INDEX

The active index is a powerful computing device. Its interconnections are dynamic, which makes it different from many other computing devices. By suitable restrictions, it can be used to realize Petri nets, the previously defined active index structure, conventional index structure such as the B-tree and a generalized Petri net called the G-net. However, it is more general than all of the above, because in the active index the arcs are not fixed and static and may change dynamically.

**5.1. It can realize the Petri net:** Suppose the Petri Net is specified by  $(P, T, I, O)$  where  $P$  is the set of places,  $T$  is the set of transitions,  $I$  is the input function for the transition, and  $O$  is the output function for the transition.

We can construct an active index as follows: in the index cell base ICB, there are cells  $ic_{pi}$  corresponding to the places  $p_i$ , and cells  $ic_{tj}$  corresponding to the transitions  $t_j$ . The active index  $IX$  consists of  $ic_0$  and these cells  $ic_{pi}$  and  $ic_{tj}$ . Furthermore, they are perennial. The cell  $ic_0$  is used to initialize the active index.

For each index cell  $ic_{pi}$  corresponding to the place  $p_i$ ,  $ic_{pi}$  will accept any input  $x$ , because input can only come from transitions. After acceptance of input, the cell  $ic_{pi}$  activates the output index cells in  $Tr_{pi}$ , where  $ic_{tj}$  is in  $Tr_{pi}$  if  $t_j$  is the output transition of  $p_i$ . The cell  $ic_{pi}$  then posts  $(Tr_{pi}, x_{pi})$  to the output list.

For each index cell  $ic_{tj}$  corresponding to the transition  $t_j$ , it will accept the input set  $\{x_{p1}, x_{p2}, \dots, x_{pm}\}$  where each  $p_i$  is the input place of transition  $t_j$ . After acceptance of input, the cell  $ic_{tj}$  activates the output index cells in  $Pl_{tj}$ , where  $ic_{pi}$  is in  $Pl_{tj}$  if  $p_i$  is the output place of  $t_j$ . The cell  $ic_{tj}$  then posts  $(\{ic_{pi}\}, x_{tj})$  to the output list, for each  $ic_{pi}$  in  $Pl_{tj}$ . In other words, by Observation 1 of Section 2, the transition  $t_j$  can send messages individually to each output place  $p_i$ .

**5.2. It can realize the previously defined active index structure:** In Smart Image Design for Large Image Databases an active index structure was defined to be a set of active index cells connected by arcs. Each cell has a number of input slots and output slots. Each input slot is connected to the output slot of another cell, and each output slot is connected to an input slot of another cell. The connected pair of input and output slots must have the same predicate. A cell  $R$  is enabled if tokens satisfying the input predicate flow into the cell. When the cell  $R$  is fired, one token each will flow to the input slot of another cell provided that the token satisfies the output predicate. When several input slots have identical predicates, they must all have tokens satisfying the predicate, before  $R$  is enabled.

The active index structure can be transformed to the equivalent Petri net where input slots with identical predicates are converted to input places for the same transitions, and output slots are converted to transitions leading to output places. Smart Image Design for Large Image Databases But the current formulation of an active index is more natural and can be used directly to describe the originally conceived active index structure. Basically,  $f(\{x_1, x_2, x_3\}, s)$  is 1, if  $x_1, x_2$  and  $x_3$  are inputs to the cell, and  $\text{pred}(x_1, x_2, x_3)$  is true. We can then post the output message to the output slot(s).

The current formulation is more general than the previously defined active index structure. The restriction of fixed input/output relationships has been removed. Index cells can be added/deleted dynamically, so that the active index varies in time.

**5.3. It can realize conventional index structures:** For example, the B-tree can be described by an active index. The technique is to provide different input to the index cell for the B-tree (called a B-cell). One input to B-cell indicates the "insertion" mode, and the other indicates the "search" mode. Other conventional index structures (index sequential, index direct, etc.) can also be described using similar techniques.

**5.4. It can realize the G-Net:** The active index system is conceptually derived from the G-Net system. A G-Net Model for Knowledge Representation where each G-Net may invoke another G-Net. Therefore, each G-Net corresponds to an active index cell. When a G-Net is invoked, it accepts the input message. When it completes its computation, it sends messages back to the invoking G-net, and then de-activates itself. Furthermore, if we introduce the various levels of abstractions into G-net, we can also describe class hierarchies and other abstraction structures. This leads to methodological considerations in specifying the active index cell, the input message space  $X$  and output message space  $Y$ .

## 6. THE REVERSIBLE INDEX FOR FEATURE-BASED INDEXING

The level-2 active index shown in Figure 3 can be used for feature-based indexing. When a feature is detected in an image, a hotspot is triggered to send a message to an index cell, which in turn may send output messages to other cells. Conversely, if we want to retrieve images having that feature, we need to *reverse* the flow in the index structure. In this section, we describe how to construct such a *reversible* index.

Suppose  $ic_i$  posts output message  $x_i$  to  $ic$ ,  $1 \leq i \leq m$ . If we want to make  $ic$  accept these  $m$  messages as input, then  $f(\{x_1, \dots, x_m\}, 1) = 1$ . We can "tag" every input message as  $(ic_i, x_i)$ , so that the input message also indicates where it comes from. Thus we have the following modified  $f$  function,  $f(\{(ic_1, x_1), \dots, (ic_m, x_m)\}, 1) = 1$ .

Similarly, we can also tag the output message of this  $ic$  as follows,  $g(\{(ic_1, x_1), \dots, (ic_m, x_m)\}, 1) = (Ic', (ic, y), s', \mathbf{a})$  if one output message  $y$  is posted to all the  $ic'$  in  $Ic'$ ; or  $(Ic', \{(ic'_1, y_1), \dots, (ic'_n, y_n)\}, s', \mathbf{a})$  if the output  $y_j$  is posted individually to each  $ic'_j$  in  $Ic'$ ,  $1 \leq j \leq n$ .

For notational convenience, let  $Ic = \{ic_1, \dots, ic_m\}$ ,  $V = \{x_1, \dots, x_m\}$ ,  $IcV = \{(ic_1, x_1), \dots, (ic_m, x_m)\}$ ,  $Ic' = \{ic'_1, \dots, ic'_n\}$ ,  $W = \{y_1, \dots, y_n\}$ , and  $Ic'W = \{(ic'_1, y_1), \dots, (ic'_n, y_n)\}$ .

We can now describe the reversible index cell as follows.

**Case 1:** One output is posted to  $n$  output cells. For the original index cell, input  $f(IcV, s) = 1$ , and output  $g(IcV, s) = (Ic, (ic, y), s', \mathbf{a})$ . For the reversible index cell, we modify  $f$  and  $g$  as follows: input  $f(\{(ic'_j, y)\}, r) = 1$  for  $1 \leq j \leq n$ , and output  $g(\{(ic'_j, y)\}, r) = (Ic', IcV, r', \mathbf{b})$ , where  $r, r'$  are new states corresponding to  $s, s'$ , respectively. In other words, the reversed  $ic$  will accept  $y$  as the input, and posts  $x_i$  to  $ic_i$  individually as the output.

**Case 2:** An individual output for each output cell. For the original index cell, input  $f(IcV, s) = 1$ , and output  $g(IcV, s) = (Ic', Ic'W, s', \mathbf{a})$ . For the reversible index cell, we modify  $f$  and  $g$  as follows: input  $f(Ic'W, r) = 1$ , and output  $g(Ic'W, r) = (Ic, IcV, r', \mathbf{b})$ , where  $r, r'$  are new states corresponding to  $s, s'$ , respectively. In other words, the reversed  $ic$  will accept  $\{y_1, \dots, y_n\}$  as the input, and posts  $x_i$  to  $ic_i$  individually as the output.

In both cases, the action sequence  $\mathbf{b}$  is left to be designed. If we first apply forward index to detect certain feature in an image, and then apply reverse index to find images having this feature, we can find images that are similar to the said image - "similar" in the sense of having the same features. Likewise, we can use forward index and then reverse index to find documents similar to a given document in the World-Wide-Web.

## 7. AN EXPERIMENTAL ACTIVE INDEX SYSTEM

The active index is a conceptual model. In actual implementation, the active index can be incorporated into almost any application system. For the Smart Image System, for example, the hotspot lends itself to a natural coupling with the active index, in the sense that once a hotspot is triggered, a message is posted and the corresponding index cell is activated. For the Mosaic application, the clicking on a hotword has similar effects.

We have built an experimental active index system. The heart of the active index system is the *IC\_Manager*, which performs the functions of receiving incoming messages, activating index cells, performing actions, and handling outgoing messages. As illustrated in Figure 5, although in theory  $ic_1$  can directly send message  $m_1$  to  $ic_2$ , and  $ic_2$  can directly send message  $m_2$  to  $ic_3$  residing in another machine, in practice every message must go through the *IC\_Manager*. Another implementation approach is to realize each cell as a separate process, but that will result in costly interprocess communication overheads. Since efficiency is a major concern, that approach was not adopted.

The core of the *IC\_Manager* is described as follows:

```
IC_Manager(message)

begin

  if message contains ic_id

    begin /*the message is for a specific ic that should already exist*/

      locate ic_id in IX;

      add message to input_list; end ;

  if message contains ic_type

    begin /*the message is for an ic to be created*/

      locate ic_type in ICB;

      create a new ic_id;

      add a new ic instance to IX;

      add message to input_list of this ic;

      add ic_id to the output_list of the output ic; end ;
```

```

while there is next ic_id in IX

    begin check whether message should be accepted;

        if message should be accepted

            if message has not been accepted by another ic

                begin    accept this message and remove it from output_list;

                    process this ic; end

            end

        end

    end

end

```

In theory, the index cell base ICB can be infinite. In practice, it is necessary to maintain a library ICB of a fairly small number of generic ic's, so that the user can create customized cells with ease. For the Smart Image System, it is also necessary to have a separate collection of generic index cells for each level of the three-level index. The ICB and IX are implemented as linked lists of C structures. Whenever there is a request to activate (or create) a new index cell, a new cell is obtained from an available list space. Conversely, a dead cell is returned to the available list space.

The *IC\_Manager* has a domain-independent part and a domain-specific part. The domain-specific part contains the specific routines used by the ic's to perform predefined actions. It also identifies and structures the external messages to be sent to the *IC\_Manager*. This clean separation of domain-independent and domain-specific parts makes it easy to adapt the *IC\_Manager* to a new application system. The *IC\_Manager* is written in standard C codes and can easily be compiled together with the intended application system, on workstations as well as PCs, to produce a customized application system with built-in active index.

Another important tool is the *IC\_Builder*, which is a visual user interface enabling the designer to visually design new index cells from scratch, or customize an ic based upon a generic ic from ICB. An example of the *IC\_Builder* screen is illustrated in Figure 6.

For a WWW client such as the Mosaic, we can invoke the active index from Mosaic, so that the user's clicks generate retrieval requests. For information retrieval in hyperspace, the simplest approach is to use only level-3 index cells to link and retrieve information. Figure 7 illustrates the experimental *Mosaic-IC* at work, where the background window on the right displays the trace of instantaneous descriptions of the active index, and the *action\_icons* in the upper-right corner show the actions performed.

For further research, the user can be modeled using level-1 index, the information abstracted using level-2 index, and information items linked and selectively presented using level-3 index. Moreover, using the reversible index, we can find documents similar to a given document. Special generic cells can be designed, to do range-based retrieval and incremental knowledge acquisition (see the following section).

In implementing the experimental active index system, we decided to provide each cell with an internal memory. Theoretically, the internal memory and the state together define the true state of the cell. In practice, it is more convenient to have an internal working memory, so that the cells can cope with different situations flexibly. The internal memory is a C structure, so that the user can include special routines in the domain-specific part of the *IC\_Manager* to manipulate it.

Using the experimental active index system, we quickly produced customized Smart Image System (*SIS-IC*), Mosaic (*Mosaic-IC*), B-Tree (*BT-IC*) and Medical Personal Digital Assistant (*MPDA-IC*). Thus the experimental active index system serves as a prototyping tool to enhance application systems with active indexes.

## 8. DISCUSSION

The active index introduced in this paper possesses the desirable characteristics discussed in Section 1: (a) The active index can be used to initiate actions and is active rather than passive. (b) Only a few index cells are activated as needed, so the index is partial rather than total. (c) The index is dynamic and can evolve, grow and shrink. (d) The index cell can send messages to the user in its action sequence and therefore the index can become visible to the user. (e) Finally, with the reversible index, the active index can be used to process imprecise queries and perform similarity retrieval.

The following topics require further research:

(a) The index cell reversal technique described in Section 6 enables us to extract features from an image to construct feature-based index cells, and then retrieve images containing such features using these index cells. The feature-based index cells, like other index cells, have a finite life time. If they don't receive any messages for a while, they die. Dead index cells can either be eliminated, or archived to tertiary storage. Therefore, the system will not be burdened with excessively large indexes. The algorithms for index cell reversal need to be carefully designed, so that we can perform feature extraction and feature-based indexing using the same index structure.

(b) The time bound  $t_{\max}$  limits the size of the active index, so that it will not grow too large. Inactive cells of the active index will be removed or archived automatically. A research issue is to study the stability of time-varying active indexes. Under

what conditions will an active index become dead? Moreover, how can we adjust  $t_{\max}$  so that the index is always below the storage constraint?

(c) The knowledge is contained in the two functions  $f$  and  $g$ . The function  $f$  restricts the inputs to be processed. The function  $g$  specifies the output, what cells to activate, next state and the action sequence. When, for example, the designer or the viewer of a hyperstructure wants to add knowledge to the cells, we need algorithms to allow incremental addition of knowledge by systematically modifying the  $g$  function.

(d) All user-system interactions in user sessions can be recorded and then filtered. In principle, every "click" can be recorded. We must decide what information should be filtered out, and based on what criteria. Since we can add/delete the filters at will, we can filter/retain any information. The user may not have a very rigid protocol, and perhaps we can only define "global states". We need to do careful domain-specific experimentation to test the filters, image class hierarchy and index cell construction techniques. On the other hand, although every "click" can be recorded, if the user keeps everything inside his/her head and does not click on anything, there is no way the system can observe even the global state. We need to design the interaction carefully so that the user MUST click on SOME CONDITION (i.e. hotspot types) to retrieve ANY image. This will force the user to be explicit.

(e) A final research topic is to develop/adopt a dynamic visual specification language to specify the index cells. The *IC\_Builder* is a step in that direction. With a dynamic visual specification language and the *IC\_Builder*, the specification and design of the active index can become much easier and yet still support rigorous formalism.

**Acknowledgements:** This research was supported in part by the National Science Foundation under grant IRI-9224563, and by Siemens Corporate Research. The following people implemented the experimental active index system: Chi-Cheng Lin, Edy Purwoko, Zhengrong Tang and Jie Xing. I would also like to thank the critical comments from Tai-Yuan Hou and Arding Hsu of Siemens Corporate Research.

**Figures:**

Figures 1-3. [click here](#).

Figure 4. [click here](#).

Figure 5. [click here](#).

Figure 6. [click here](#).

Figure 7. [click here](#).