1998 International Computer Symposium
Workshop on Software Engineering and Database Systems
December 17-19, 1998, N.C.K.U., Tainan, Taiwan, R.O.C.

A HYPER-CONTROLLABLE AND HYPER-RECORDABLE MEDICAL IMAGE APPLICATION BASED ON PATTERN TECHNOLOGY

Ku-Yaw Chang[†], *and Lih-Shyang Chen*[*]

Department of Electrical Engineering,
National Cheng Kung University, Tainan, Taiwan, R.O.C.
Email : canseco@mirac.ee.ncku.edu.tw[†], chens@mail.ncku.edu.tw[*]

ABSTRACT

The basic idea of hyper-control is to use a multimedia document, called a hyper-control document, to control other application systems. In this paper, we propose a new concept, called hyper-record, to facilitate the creation of hyper-control documents. Based on an extension of the Command Processor pattern and other related patterns, we have developed a medical image system, called Discover, which can support both the hyper-control and the hyper-record mechanisms. Examples are given to illustrate the mutual relationship between Discover and its hyper-control documents. Our experience in using patterns is also addressed.

1. INTRODUCTION

When an application system is getting complicated, it is difficult for an end user to use the application. One way to overcome the system's complexity is to use a multimedia document, called a hyper-control document, to guide users to operate the system. This concept was called hyper-control[1]. However, the creation of such hyper-control documents usually needs much effort. In this paper, we propose a new concept, called hyper-record, to solve this problem. Its main idea is to allow an application to log activity automatically and save the associated information as a hyper-control document.

One important issue for an application to support the hyper-control mechanism is to provide an external control channel in addition to the internal control built in the original application. The Command Processor pattern [2], which allows an application to support different modes of user interaction, is a good approach to support not only the hyper-control mechanism, but also the hyper-record mechanism. But it causes problems because of the fixed execution sequence of computation codes and dialog codes inside the body of a command object.

This paper describes the design of a medical image application system, called Discover, that supports both the hyper-control and hyper-record mechanisms, and how it required that we extend the Command Processor pattern. In addition, our design also uses the Document-View pattern, the Memento pattern, the Visitor pattern, and the Singleton pattern.

The remainder of this paper is organized as follows: Section 2 describes the hyper-control and hyper-record mechanisms and gives an overview of the Command Processor pattern. In section 3, we illustrate the system architecture of Discover, which supports both the hyper-control and the hyper-record mechanisms. Section 4 gives examples to illustrate the cooperation between Discover and its hyper-control documents. In section 5, we summarize our experiences in applying pattern-based strategy to our system.

2. BACKGROUND

Discover is a distributed interactive visualization system[3][4], which has been running in National Cheng-Kung University hospital since 1993. Shortly after Discover was implemented, we observed that physicians were having trouble understanding and navigating through Discover processes for image analysis and generation. Even with the aid of help documents, physicians still have to go back and forth between working with Discover and reading through its help documents while learning how to use the Discover. During such a process of learning, some misunderstandings and mismatches may occur. These problems go far beyond the ability of general help systems. In other words, there still exists some physical as well as conceptual gaps from the documents to its associated application system, especially when the application is very complicated.

2.1 Hyper-control

The concept of hyper-control was first proposed in [1] to fill the gaps mentioned above. Its basic idea is to use a multimedia document (written by non-programmers working with engineers) to control other hardware or software systems. The control commands are installed in anchors, which is similar to that of "hyperlink". The major difference is that when such an anchor is clicked, a command is issued to activate an operation of its associated application system, which is originally a stand-alone application system with its own user input, i.e. internal control, and is currently running simultaneously on the same machine. Such a multimedia document with the ability to control an application is called **a hyper-control document**. **A hyper-controllable application system** is the one that supports the hyper-control mechanism.

In order to clarify the ideas of hyper-control and to explain how it works, we first give a simple example here. A hyper-controllable medical image application is shown in

Fig.1. It can process two different medical image data types: gray-level images(12 bits/pixel) and true-color images.

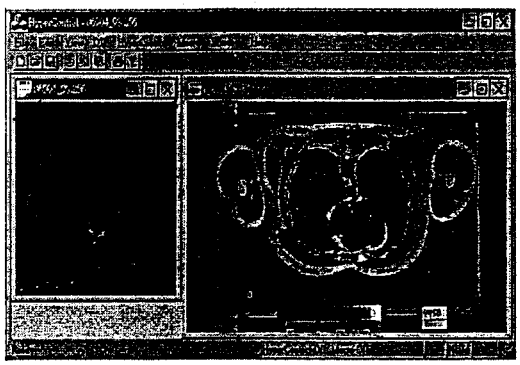

Figure 1. A hyper-controllable and hyper-recordable medical image application.

Suppose a physician wants to segment out the spine of the left gray-level image, he/she may need to do the following steps after loading the image data: (without the aid of a hyper-control document)

1. On the **DIP** menu, choose **Histogram Equalization** to obtain a better image quality.
2. On the **DIP** menu, choose **Threshold**. The **Threshold** dialog box will show up on the screen.
3. Move two scroll bars on the dialog box to adjust two threshold values according to the result he/she sees.
4. Click **OK** to complete this value setting. As a result, the spine will be segmented from the image.

A physician has to remember each step in the above processing procedure and choose a corresponding command under a correct menu. Different clinical cases have different processing procedures. As the number of these procedures increases, it is virtually impossible for a physician to remember all the detailed steps correctly.

With the aid of the hyper-control mechanism, a physician needs to record each function he/she uses to process the image as a hyper-control document, as shown in Fig.2, and tries to come up with a procedure that would apply to a similar data set. If necessary, a physician can work with engineers. A physician can apply the same procedure to many data sets to make sure the procedure consistently produces reasonable results. If so, the procedure becomes a diagnostic protocol for the particular case. After the establishment of a hyper-control document, physicians can browse it and apply the same procedure to other similar data sets by clicking on each anchor in the document.

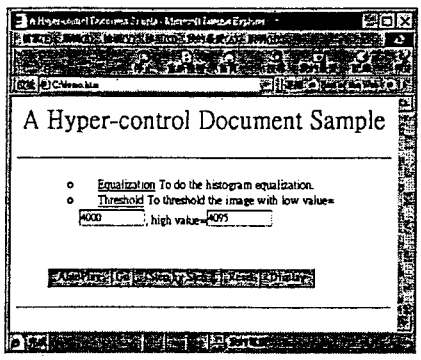

Figure 2. Using a web browser to browse (Internet Explorer) a hyper-control document.

In other words, the hyper-control mechanism can help us standardize the processing procedure. Other users simply follow through each instruction in the procedure, click on each anchor, and interact with the associated application as prompted. The hyper-control can also shorten training time, ensure that users operate the system in a consistent way and at a uniform quality level. In some sense, it also provides users with a more friendly user interface(instead of remembering each step in the processing procedure and clicking on the pull-down menus for each operation).

When applied to the on-line documentation, the hyper-control mechanism can also fill up the physical as well as conceptual gaps from the documents to its associated application system and increases the effectiveness of the on-line documentation. More generally, the hyper-control mechanism enables us to tailor a general-purpose application system(that is a hyper-controllable application system) into a turn-key system. That is, users can completely ignore the original general-purpose user menus and work with the hyper-control document that has been tailored to solve a particular problem.

However, a hyper-controllable application does not come for free. If we partition an application according to Document/View pattern as shown in Fig.3, the key to support the hyper-control mechanism is to provide an alternative user input channel to accept external control from other applications, in addition to the original user input channel, i.e. the internal control. In fact, a hyper-controllable application is a stand-alone one with its own input /output and can still work well even without any hyper-control documents.

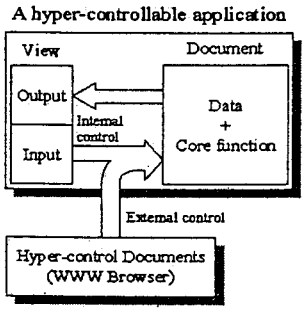

Figure 3. The key to support the

In our implementation, hyper-control documents are in the HTML format. By doing so, we can avoid "re-inventing the wheel" and use existent web browsing/authoring tools to browse/edit these documents. Besides, it also becomes very easy and natural to integrate hyper-control documents with some other HTML-based help systems, which are the de facto standard format for the majority of on-line documentation. Therefore, a web browser is just a tool to issue commands to control a hyper-controllable application through the external input channel. Theoretically, any other application can control a hyper-controllable application if it knows how to send commands to the external input channel.

2.2 Interception Points for Hyper-Control

In fact, the basic operations of an interactive application can be divided into those that take arguments and those do not. A more complicated operation can be composed by these basic operations. In the above example, the histogram equalization command does not take any arguments, while

the threshold command does. There is only one way to issue commands that take no arguments from a hyper-control document to its associated application. That is to invoke the algorithm of the command directly. However, there are three different ways to issue commands that take arguments (take the above threshold command as an example):

1. **Without parameters / dialog displayed**: a threshold command message without parameters is sent to the application. The **Threshold** dialog box is displayed and contains two scroll bars initialized at 2000 and 2048 respectively. The initial values are provided by the application itself.
2. **With parameters / dialog displayed**: a threshold command message with parameters 4000 and 4095, which is kept on the hyper-control document as shown in Fig.2, is sent to the application. The **Threshold** dialog box is still displayed, but its two scroll bars are initialized at 4000 and 4095 respectively according to the parameter values coming along with the command message. Note that in case 1 and 2, a user still has a chance to adjust the parameters through the dialog interaction before applying it.
3. **With parameters / no dialog displayed**: a threshold command message with parameters 4000 and 4095, which are kept on the hyper-control document, is sent to the application. The threshold operation (with parameters 4000 and 4095) is carried out directly without displaying the **Threshold** dialog box.

In other words, a hyper-controllable application should be able to accept not only commands from itself(internal control), but also those issued in different ways from hyper-control documents(external control). Generally speaking, an interactive application system is composed of a dialog component and a computation component[5], as illustrated in Fig.4. In order to achieve the goal mentioned above, we divide the dialog component into three constituents as follows:

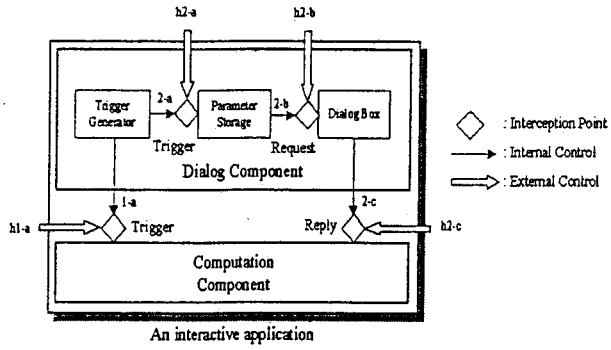

Figure 4. Four interception points for hyper-control.

1. **Trigger Generator**: interfaces through which users can trigger commands of the application, such as menus, buttons and even a keyboard.
2. **Parameter Storage**: a place to store parameters of each operation that takes arguments. When an operation is complete, the final parameters should be stored back here. Next time when the same operation is applied again, the parameters on the dialog box are initialized according to the pre-stored parameters on this storage. Thus, users can see exactly what they set last time the same operation was applied.
3. **Dialog Box**: to create and display dialog boxes.

When a user clicks a menu item or strikes a keyboard shortcut through Trigger Generator, the flow of the control is the following: (refer to Fig.4)

(1) an operation that requires no parameters invokes a Trigger on the Computation Component to carry out the associated computation directly (1-a).

(2) an operation that requires parameters first invokes a Trigger(2-a) to fetch pre-stored parameters from the Parameter Storage. After the parameters are obtained, the Request(2-b) is invoked. As a result, the corresponding dialog box will be displayed with the parameters as the initial values. At this moment, a user can adjust the parameters according to his/her own needs. After the confirmation of parameter setting, a Reply(2-c) with final parameters will be invoked on the Computation Component to complete the whole operation. Of course, the final parameters should be stored back to the Parameter Storage after the computation is done, as stated previously.

According to the above analysis, we found that there are four interception points available in the course of each operation for a hyper-control document to activate an invocation to control the application directly:

(1) **Trigger**: to simulate the effects of choosing menus or keyboard shortcuts by invoking a corresponding Trigger on Computation Component(h1-a) or Dialog Component(h2-a).

(2) **Request**: to invoke a Request with parameters on Dialog Component(h2-b). A corresponding dialog box of the application system will be displayed. And its parameters are initialized according to the parameter values coming along with the Request invocation.

(3) **Reply**: to invoke a Reply with parameters on Computation Component(h2-c). The major difference from a Request invocation is that the computation is carried out immediately without showing any dialog box.

Any invocation coming from one of the above interception points will also activate its ensuing invocations or actions. Therefore, an invocation from a hyper-control document has exactly the same effects as if the invocation were activated by the application itself. More importantly, such an arrangement can meet all the requirements that a hyper-controllable application can accept commands issued in different ways from hyper-control documents.

2.3 Hyper-record

Although physicians report that hyper-control documents make Discover easier to use, they still need to write hyper-control documents with much effort and, sometimes, even need to work with engineers. Therefore, physicians hope that the Discover can record whatever they are experimenting automatically, so that the same procedure, which

turns out to be a good solution, can become a standard one to solve the particular problem at hand and be reused in the future. Even though a procedure may fail to work well, physicians still can make reference to the procedure for further research.

In the current working environment, there is no way to fulfill the function described above and to automate such a working flow. In other words, there seems to be a gap from an application system to its on-line documentation. What has been missing is there is no way to transform the information from the operations to a document that can be used as part of the on-line manuals, references, or standard procedures for further training and research purpose in the future.

In order to overcome the above problem, we propose a new concept, **hyper-record**, to fill-up the gap mentioned above. The main idea of hyper-record is to allow an application to log a user's operations automatically and save the associated information as hyper-control documents which can be edited later if necessary. After that, users can control the application through hyper-control documents. With the aid of the hyper-record mechanism, the creation of a hyper-control document, i.e. a standard processing procedure, becomes an easy job. A physician now can create such a document simply by performing all operations once, without knowing how to create it in detail. **A hyper-recordable application system** is the one that supports the hyper-record mechanism.

In fact, there are many other complex or general-purpose applications, such as word processing, spreadsheets, and database management, suffering these same problems. Usually these systems have a lot of generic functions and allow users to solve many different problems through diverse combinations of these generic functions. What on-line documents can provide is the explanation of these generic functions and an easy way to navigate through the contents. In such a situation, it is not sufficient for users to learn only the basic functionality of each command to cope with various problems. For many users, what they still lack is a convenient way to dynamically create a hyper-control document, which can guide them to complete a whole processing procedure and even can be stored back to become part of their on-line documents. This will be very helpful not only in the standardization of an operation, but also in the conveyance of experiences.

2.4 Command Processor Pattern

The basic idea of Command Processor pattern, which builds on the Command pattern in [6], is to encapsulate service requests into command objects and to separate the request for a service from its execution. The Command Processor Pattern illustrates more specifically how command objects are managed. The structure of Command Processor pattern is shown in Fig.5.

The Abstract Command component defines the interface of all command objects. An indispensable procedure of this interface is the one to execute a command object. For each function, a concrete command component is derived from the Abstract Command. A Command component implements the interface of the Abstract Command by using zero or more Supplier components. The Controller represents the interface of the application. It accepts requests and creates the corresponding command objects. The command objects are then transferred to the Command Processor for execution. The Command Processor receives command objects from the Controller and takes responsibility of managing them, including starting their execution. It is also the key component that implements additional services such as the storing of request objects for later undo. The Supplier components provide most of the functionality required to execute concrete commands. When an undo mechanism is required, a supplier usually provides a means to save and store its internal state.

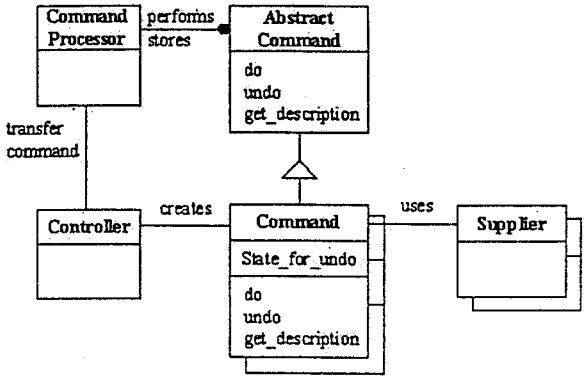

Figure 5. The structure of Command Processor pattern.

2.5 Other Related Patterns

Other related patterns used in our system include the Document-View, the Memento, the Visitor, and the Singleton patterns. Please refer to [2] and [6] if necessary.

3. SYSTEM ARCHITECTURE

3.1 Separation of computation and dialog command objects

Since one of the basic ideas of the Command Processor pattern is to separate the request for a service from its execution to support different modes of user interaction like external control of the application, it seems to be suitable for supporting the hyper-control mechanism. However, the invocations of the elements in the computation component and dialog component are usually interwoven inside the body of a command object. In other words, the execution sequence of the invoked elements in the computation components and dialog components is fixed in a command object. A piece of computation codes is tightly bound with its preceding menu selection or dialog activity. Only after a menu selection or a dialog activity, will the corresponding computation codes be executed. Thus it is virtual impossible for the application to be fully controlled by hyper-control documents through all the interception points shown in Fig.4.

In order to overcome above limitations, we separate the computation codes and dialog codes completely by duplicating the whole Command Processor constituent components into two different parts: one is the computation component for core computation, the other is the dialog component for dialog activities. Each part functions as the original Command Processor pattern and has its own Command Processor, Controller, Supplier and Command.

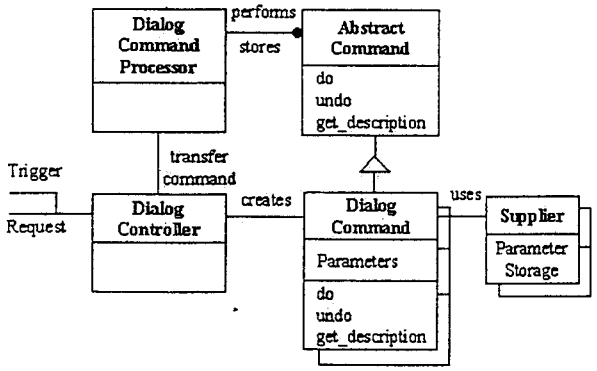

Figure 6. Dialog Command Processor.

When the Dialog Controller receives a Trigger or a Request invocation, it will create a corresponding command object, called a Dialog Command Object, as shown in Fig.6. (The Trigger and Request invocations stand for the Trigger and Request interception points provided by the dialog component in Fig.4.) The controller then transfers this new command object to the Dialog Command Processor for execution. The processor activates the execution of the dialog command object, whose main job is to prepare a set of parameters for the ensuing computation. Usually, there are three major steps during the execution process of a dialog command object:

(1) To retrieve pre-stored parameters from its supplier, i.e. Parameter Storage. This step is skipped if what the Dialog Controller receives is a Request invocation, which comes with parameters.

(2) To display the corresponding dialog box with parameters. The final parameters will be kept inside the body of the dialog command object.

(3) To invoke a Reply with final parameters to the Computation Controller to carry out the computation. This invocation is also the only connection between the Dialog and the Computation Command Processor.

When the Computation Controller receives a Trigger or a Reply invocation, it will create a corresponding command object, called a Computation Command Object, as shown in Fig.7. The computation command object will be transferred to the Computation Command Processor for actual execution. (The Trigger and Reply invocations stand for the Trigger and Reply interception points provided by the computation component in Fig.4.) Of course, the parameters coming along with Reply invocation will be delivered to the computation command object.

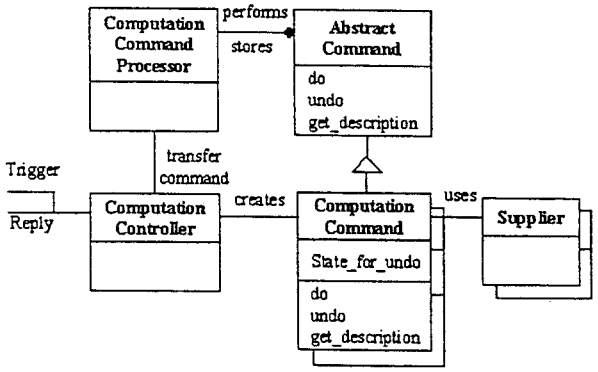

Figure 7. Computation Command Processor.

By providing two sets of command processors, we can separate the computation codes and dialog codes completely. Their execution sequence is no longer fixed in a command object, but separated into two command objects. With this arrangement, it becomes more flexible for hyper-control documents to control a hyper-controllable application. We will give an example to explain the scenarios in the next section.

3.2 Reuse a Command Object

Our application domain has at least two types of medical image data: gray-level images and true-color images. In terms of the Document-View architecture, they represent two image documents. In other words, a command object has to face different supplier types. When the same operation is applied to different documents(suppliers), it may take different arguments. That is, the interfaces to implement the same function on different suppliers may be different. Therefore, we may need to provide different command objects for the same operation. For example, when a **Threshold** command is applied to a gray-level image, part of the execution codes may look like this:

```
// pSupplier: a pointer to a supplier
// nLow, nHigh: two threshold parameters
pSupplier->Threshold(nLow, nHigh);
```

or like this when applied to a true-color image:

```
// nRLow, nRHigh: threshold values for Red color
// nGLow,nGHigh: threshold values for Green color
// nBLow, nBHigh: threshold values for Blue color
pSupplier->Threshold(nRLow,  nRHigh,  nGLow,
                     nGHigh,  nBLow,  nBHigh);
```

A better solution to this problem is that the same command object can be applied to different supplier types without any change. In order to achieve this goal, we use Visitor pattern to package related operations from each document in a separate visitor and organize the documents as elements, as illustrated in Fig.8.

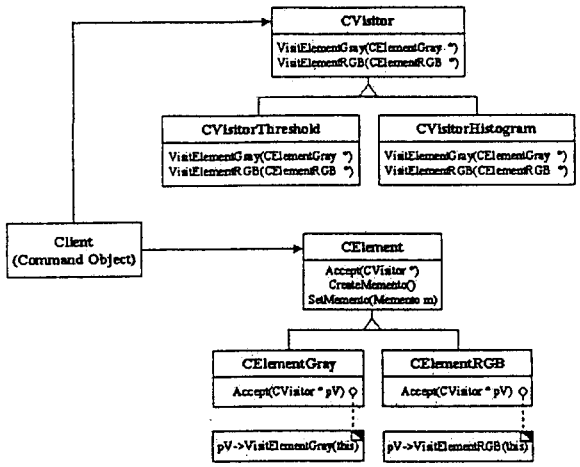

Figure 8. Visitor Pattern

When an operation is executed, a computation command object simply passes a visitor to its associated element - supplier. When an element "accepts" the visitor, it sends a request to the visitor, which will then execute the operation for that element. The execution function of the Threshold command object may look like this(no matter what types its supplier is):

```
void CCmpCmdThreshold::Do()
{    // 1. get and store the memento
     m_pMemento = m_pSupplier->CreateMemento();
     // 2. apply the algorithm
     m_pSupplier->Accept(&m_VisitorThreshold);
}
```

In this example, the visitor m_VisitorThreshold is created in the constructor. The class CCmpCmdThreshold provides two different constructors: one for gray-level images and the other for true-color images. The Computation Controller will choose one of them to create this command object according to the arguments it receives.

3.3 Undo – Memento Pattern

In fact, a computation command object also plays the role of caretaker in Memento pattern[6]. In other words, before the computation is carried out, the command object will ask the supplier to create a memento to save its current internal state (See step 1 in the Do function of CCmpCmdThreshold). Later, when the undo procedure is invoked, this memento will be delivered to the supplier for restoring its internal state, as illustrated as follows:

```
void CCmpCmdThreshold::Undo()
{
     // to restore the memento
     m_pSupplier->SetMemento(m_pMemento);
     m_pMemento = NULL;
}
```

3.4 Combine with Document-View

Fig.9 shows the structure after combining the Document-View pattern and the Command Processor pattern. In order to support the undo/redo mechanism, each image document has its own command history, which is maintained by the command processor. As a result, every image document has an instance of the Dialog Command Processor and an instance of the Computation Command Processor. However, because a command object can be applied to different supplier types by using Visitor pattern, its creator, the Dialog Controller or the Computation Controller, is designed to be a global and unique component by using Singleton pattern. Therefore, a command object is created by the controller and is first transferred to the active document, instead of delivering to the processor directly. The active document will then just forward the command object to its processor.

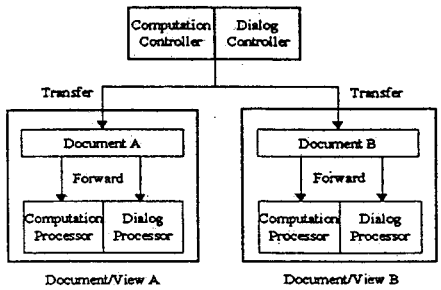

Figure 9. The structure after combing Document-View pattern and Command Processor pattern.

3.5 Proxy Controller

In Fig.9, although both the Dialog Controller and the Computation Controller can be accessed globally, only those objects belonging to the same application, i.e. the same address space, can access them. In order to accept external control from hyper-control documents, two proxy controllers are provided, as shown in Fig.10. These proxy controllers are automation objects, which allows other applications to launch and operate on it directly[7]. Each proxy controller provides exactly the same interfaces as what its corresponding original controller has. When receiving a request, the proxy controller will simply forward the request message to the original controller. In fact, two original controllers cannot tell where the Trigger/Request/Reply invocation comes from: an external control from the proxy controller or an internal control from the application itself.

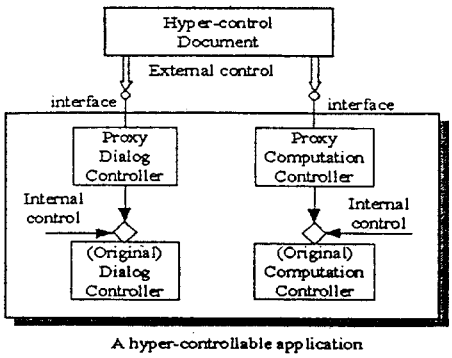

Figure 10. Two proxy controllers.

3.6 Recorder

As shown in Fig.5, every command object has an additional interface - get_description, which is used to generate a description to describe the command object itself. Such a description is the **self-description** of a command object. In order to fit in with hyper-control documents, the self-description format is also in the HTML format. The contents of such a description contain at least an anchor (to send commands when clicked), some explanation and parameters (if any).

A Recorder is a component which cooperates with the Computation Command Processor and is responsible for recording every operation. When the hyper-recording process begins, the processor delivers a computation command to the Recorder after the command object is executed. The Recorder retrieves the self-description of each command object and stores them temporarily. When the hyper-recording process ends, the Recorder will output and save all the self-descriptions as a hyper-control document.

4. EXAMPLES

4.1 Hyper-record

When the hyper-recording process begins, an empty hyper-control document is created and displayed in a dialog box. Next, the user issue two commands: **Histogram Equalization**(without arguments) and **Threshold**(with arguments). Each command will be recorded automatically and displayed in the dialog box, through which the user can see what has been recorded, as shown is Fig.11. When the user click the **OK** button on the dialog box to end this process, this hyper-control document will be saved.

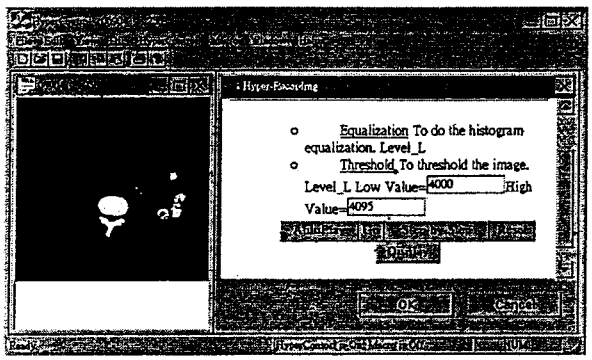

Figure 11. A user can see what has been recorded during the hyper-recording process.

4.2 Hyper-control

A hyper-control document example is shown in Fig.2. In the hyper-control document, the initial low and high values of the threshold parameters are 4000 and 4095.

(1) When a user clicks the Equalization anchor to apply a histogram equalization operation that takes no arguments, a Trigger is invoked on the Proxy Computation Controller through the automation mechanism (h1-a in Fig.4). This invocation is forwarded to the Original Computation Controller, which will then create a computation command object for the histogram equalization operation and execute it. The results are illustrated in Fig.12

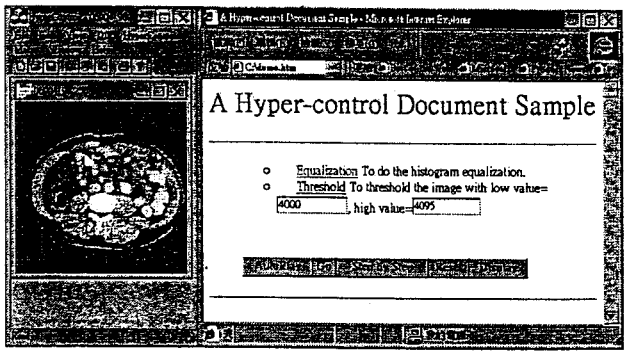

Figure 12. The results after invoking a Trigger(without parameters) from a hyper-control document.

As mentioned in section 2, there are three different ways to issue a command that takes parameters, including **Trigger**, **Request**, and **Reply**. When users click the **Display** button on the hyper-control document, three radio buttons are displayed. Users can choose one of them to issue a command. When the user clicks the **Threshold** anchor that corresponds to Trigger, Request and Reply selection respectively, the results are shown in Fig.13, 14 and 15.

(2) **With Trigger selection**: a threshold command message without parameters is sent to the application through the Trigger interception point on Dialog Controller(h2-a in Fig.4). The **Threshold** dialog box is displayed. Its low and high values are initialized at 2000 and 2048 according to the values retrieved from the Parameter Storage of the application. Users can adjust the parameter values before applying it.

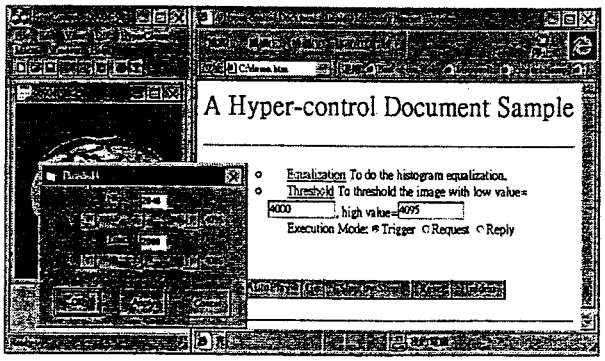

Figure 13. The results after invoking a Trigger(with parameters) from a hyper-control document.

(3) **With Request selection**: a threshold command message with parameters 4000 and 4095 is sent to the application through the Request interception point on Dialog Controller(h2-b in Fig.4). The **Threshold** dialog box is also displayed. But its initial low and high values are initialized at 4000 and 4095 respectively according to the values on the hyper-control document. Users can adjust the parameter setting before applying it.

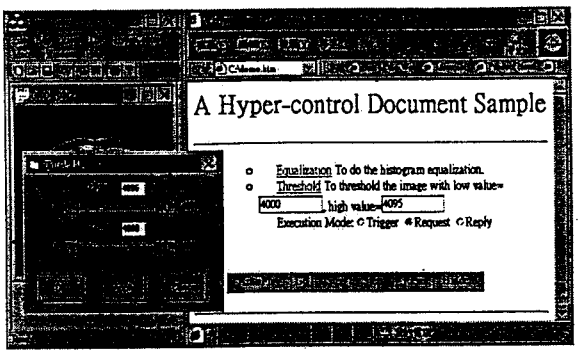

Figure 14. The results after invoking a Request(with parameters) from a hyper-control document.

(4) **With Reply selection**: a threshold command message with parameters 4000 and 4095 is sent to the application through the Reply interception point on Computation Controller(h2-c in Fig.4). No dialog box is displayed. The threshold operation is carried out directly with low value 4000 and high value 4095

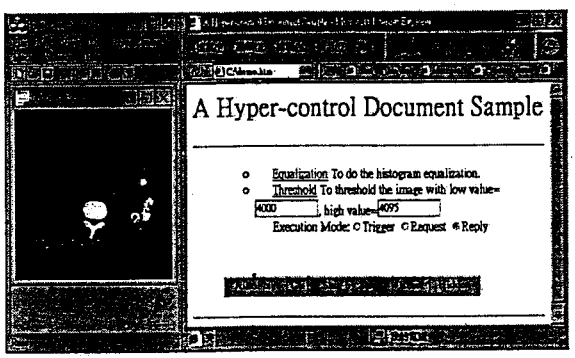

Figure 15. The results after invoking a Reply(with parameters) from a hyper-control document.

5. CONCLUSIONS

Hyper-control is a powerful and flexible mechanism to standardize the processing procedures. It can make a general-purpose application easy to use in different domains. Hyper-record is the counterpart of hyper-control. It allows users to create a hyper-control document easily. The basic requirement for an application to support the hyper-control mechanism is to provide an additional input channel to receive external control from other applications. We extend the Command Processor pattern by separating the computation codes and dialog codes into tow kinds of command objects, i.e. the Dialog and Computation command objects, to support the hyper-control mechanism. At the same time, the hyper-record mechanism is also achieved based on this pattern.

Besides, when combined with Document-View pattern, a command object may have different supplier types (image document types). In such a case, the reusability of a command object becomes an important issue. By adopting Visitor pattern for execution and Memento pattern for the undo function, a command object can be applied to different supplier types without any change.

In order to receive requests from hyper-control documents or other applications, two proxy controllers are also provided. This allows hyper-control documents to become part of the application's user interface and to be integrated with an HTML-based help system easily.

Based on above pattern technology, we have developed an interactive medical image application system to support the hyper-control and the hyper-record mechanisms successfully. During the development process, it was discovered that patterns do provide a very good solution in the design of system architecture to solve our practical problems. Although several patterns work together, they still help us have a clear roadmap of the complicated control flow.

6. REFERENCES

[1] L.S. Chen, P.W. Liu, K.Y. Chang, J.P. Chen, S.C. Chen, H.C. Hong, and J. Liu, "Using Hypermedia in Computer-Aided Instruction," IEEE Computer Graphics and Applications, Vol. 16, No. 3, pp. 52-57, May 1996.

[2] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad and M. Stal, *A System of Patterns - Pattern-Oriented Software Architecture*, John Wiley & Sons Inc., New York, 1996.

[3] L.S. Chen, C.P. Chen, J. Chen, P.W. Liu, and T. Shu, "Distributed and Interactive Three Dimensional Image System," Computerized Medical Image and Graphics, Vol. 18, No. 5, pp.325-337, Sep. 1994.

[4] P.W. Liu, L.S. Chen, S.C. Chen, J.P. Chen, F.Y. Lin, and S.S. Hwang, "Distributed Computing: New Power for Scientific Visualization," IEEE Computer Graphics and Applications, Vol. 16, No. 3, pp.42-51, May 1996.

[5] H. R. Hartson, D. Hix, "Human-Computer Interface Development: Concepts and Systems for Its Management", ACM Computing Surveys, Vol. 21, No. 1, pp. 5-92, March 1989.

[6] E. Gamma, E. Helm, R. Johnson and J. Vlissides, *Design Patterns - Elements of Reusable Object-Oriented Software*, Addison-Wesley Publishing Company Inc., 1995.

[7] Adler, Richard M., "Emerging Standards for Computing Software," Computer, pp. 68-77, March 1995.