行政院國家科學委員會專題研究計畫　期中精簡報告

視覺化可再用軟體元件為基礎的軟體建構環境系統之建置

(1/3)

計畫類別： 個別型計畫
計畫編號： NSC91-2213-E-009-079-
執行期間： 91 年 08 月 01 日至 92 年 07 月 31 日
執行單位： 國立交通大學資訊工程學系

計畫主持人： 陳登吉

計畫參與人員： 陳登吉 李東寅 卓勇君 嚴承宗 彭世榮 梁碧峰

報告類型： 精簡報告

處理方式： 本計畫涉及專利或其他智慧財產權，2 年後可公開查詢

中　華　民　國 92 年 5 月 7 日

視覺化可再用軟體元件爲基礎的軟體建構環境系統之建置

(1/3)

A Visual and Reuse-based paradigm for Software Construction (1/3)

計畫編號：NSC 91-2213-E-009-079

執行期限：91 年 8 月 1 日至 94 年 7 月 31 日

主持人：陳登吉　交通大學資訊工程系教授

一、摘要

中文摘要

軟體生產力與軟體品質是發展軟體系統時面臨的最大問題，運用*軟體再利用*的方法可以有效改善這些問題，*可再用軟體元件*是應用軟體再用技術發展軟體系統時的基本建構單元。物件導向技術包含四種機制，分別是資料抽象化、資訊隱藏、繼承、和動態繫結，這些機制將可協助我們設計和製作可再用軟體元件。經由視覺化程式環境之協助，我們可將這些軟體元件使用一個視覺化的圖像來代表，藉由組合這些圖像來開發軟體系統，可以改變過去 programming-in-the-small 的思維模式而成爲 programming-in-the-very-large。爲了達成這個目的，我們必須定義出這種程式思維模式和製作出視覺化輔助語言。

多媒體技術在現代電腦軟體應用上佔有重要之角色，它具有較佳的使用者親和性，同時也較能反映出真實世界的模式。因此，我們將多媒體資料結合可再用軟體元件而發展出*多媒體可再用元件*，這一類軟體元件除了包含程式碼和說明文件之外，它還包含媒體資料、運作程序和訊息機制。基於這些多媒體可再用元件，我們提出革新的軟體需求表示方式 -- 使用一系列的動畫、影像和聲音表示一段軟體需求，取代傳統大量的文字描述。此種軟體需求表示方式可以提供使用者一個視覺化的方式來檢視軟體需求以及早獲致使用者對該軟體需求的回饋，同時，此方式也提供一個更自然的溝通模式給系統分析者和系統使用者。

本計劃提出一爲期三年的研究（ 包含下列主題)：

1) 設計及製作可再用軟體元件及多媒體可再用元件，並以 icon 方式表達。
2) 提出一個基於可再用軟體元件的視覺化軟體建構思維模式。
3) 提出新的軟體需求描述模式並製作其輔助工具。
4) 評估上述軟體需求描述輔助工具之效益。
5) 製作以視覺化可再用軟體元件爲基礎的軟體建構環境系統。

第一年將專注於第 1)和 2)項的研究，第二年專注於 3)和 4)項的研究，第三年則將專注於 5)的研究並完成整個軟體建構環境系統。這五項主題不只關係到軟體發展者是否能將軟體發展的策略提升到 programming-in-the-very-large 的層次，本研究成果更可將軟體建構提升到視覺化需求表示的方式及視覺化的程式產生方式。這對軟體生產力和軟體品質有正面的助益。此系統的完成將可讓軟體需求的表達方式透過多媒體元件方式的組合而成爲類似互動動畫說明軟體需求。如此軟體設計者和使用者之間的鴻溝得以有效的塡補。這將是一項創新的軟體需求表達方式。此系統的完成將協助程式設計者以元件化的方式透過視覺化的組裝工具來建構應用軟體系統。這將對軟體的生產力及品質有相當大的助益。

關鍵詞：軟體生產力；軟體品質；軟體再利用；

軟體元件；多媒體軟體元件；物件導向技術（Object-oriented Technology）；視覺化程式；視覺化軟體需求

英文摘要

Software reuse has been considered an effective approach to improve software *productivity* and software *quality*. *Reusable Software Components (RSC)* are the basic building blocks for software construction based on the software reuse practice. Object-oriented mechanisms-data abstraction, information hiding, inheritance and dynamic binding-are the base for the design and implementation of software reusable components. With the aid of a visual programming model, one can treat each reusable component as an icon. The programming paradigm can therefore be moved from programming-in-the-small to programming-in-the-very-large by visually manipulating these available icons (reusable components). To accomplish this objective, a visual programming paradigm and an icon interconnection language must be defined and implemented.

Multimedia technology has played an important role in today's application software due to its friendly user interaction as well as its naturally fitting on real world modeling. We, therefore, extended reusable software components to incorporate with multimedia. In other words, the reusable software components include not only code and documents, but also voice narration, animation sequences and message mechanisms. We called such software components as *Multimedia Reusable Components (MRCs)*. Based on these developed MRCs, a novel software requirement representation paradigm is introduced. With this novel representation paradigm for requirement representation, one can view the software requirement representation as sequences of animation instead of reading voluminous software requirements. Such a novel software requirement representation paradigm will provide users a visual effect and to have an earlier feedback from users. Also, it will provide an easy and natural form of the communication between designers and users.

Thus, in this project research, we focus our study on the following issues:

1) Discuss reusable software components and propose the multimedia reusable components.

2) Propose a visual software construction paradigm based on multimedia reusable components.

3) Propose a new software requirement representation paradigm and implement a supporting tool.

4) Evaluate the requirements representation tool.

5) Design and Implement a visual programming paradigm and evaluate the feasibility of such a visual programming environment.

Issues 1) and 2) will be studied in the first year project; issues 3) and 4) will be covered in the second year study; while issue 5) will be focused in the third year and eventually, the visual software construction environment will be established.

Based on these research results, software construction can then be conducted at the level of programming-in-the-super-large. These research results will contribute to the improvement of software productivity and quality.

Keywords：Software Productivity, Software Reuse, Reusable Component, Reusable Multimedia Component, Object-oriented Technology, Visual Programming, Visual Requirement.

二、計畫緣由與目的

The software productivity, quality, and maintenance are still the major problems in modern computer software industry. Software reuse is an effective means of overcoming some of the problems. With the evolution (or some may prefer revolution) of the objected-oriented paradigm, software reuse has been extensively studied. Notable examples include Cox's Software IC [6], Booch's Ada components [4], Freeman's classification of software reusability [10], Prieto-Diaz's facet scheme for software reusability classification [7], Chen's interface design for reusable software components and C++ reusable components [1], Common object model (COM) [16], Java Bean, Eric Gamma's Design Patterns [13], Grady Booch's Application Framework [14], and Talph E. Johnson's Frameworks [15]. Above results have subsequently led to commercial software products such as Software IC by Stepstone Corporation, Ada reusable components, Booch's Components, and C++ reusable components.

Multimedia technology has played an important role in today’s application software due to its friendly user interaction as well as its naturally fitting on real world modeling. We, therefore, extended reusable software components to incorporate with multimedia. In other words, the reusable software components include not only code and documents, but also voice narration, animation sequences and message mechanisms. We called such software components as *Multimedia Reusable Components (MRCs)*. Based on these developed MRCs, a novel software requirement representation paradigm is introduced. With this novel representation paradigm for requirement representation, one can view the software requirement representation as sequences of animation instead of reading voluminous software requirements. Such a novel software requirement representation paradigm will provide users a visual effect and to have an earlier feedback from users. Also, it will provide an easy and natural form of the communication between designers and users.

Thus, in this project research, we focus our study on the following issues:

1) Discuss reusable software components and propose the multimedia reusable components.

2) Propose a visual software construction paradigm based on multimedia reusable components.

3) Propose a new software requirement representation paradigm and implement a supporting tool.

4) Evaluate the requirements representation tool.

5) Design and Implement a visual programming paradigm and evaluate the feasibility of such a visual programming environment.

Issues 1) and 2) will be studied in the first year project; issues 3) and 4) will be covered in the second year study; while issue 5) will be focused in the third year and eventually, the visual software construction environment will be established. Based on these research results, software construction can then be conducted at the level of programming-in-the-super-large. These research results will contribute to the improvement of software productivity and quality.

In the next year NSC project (2003.8.1 – 2004.7.31), a novel representation system involving visual software requirements with a requirement-authoring tool based on MRCs will be studied (which are the issues 3 and 4 in the project proposal).

三、結果與討論 (1/3)

The first year research of this 3-year integrated NSC research project will be focused on the following two issues:

1) Discuss reusable software components and propose the multimedia reusable components.

2) Propose a visual software construction paradigm based on multimedia reusable components.

In the following, we present the results from these studies.

3.1 Reusable software components

Reusable software component is a software module containing a standardized interface specification, a functional description on this component, a use format and an example. Most importantly this software module is designed and implemented based on object-oriented programming paradigm that can be repeatedly used in software construction.

3.1.1 Features of reusable software components

A software product has different features than other products, among which include flexibility and extendibility; it must consider future modifications or changes for a user’s requirement as well. Thus, owing to the same reasons, an effective software component contains flexible and extendible features as well.

Parts in many software systems or application systems are frequently the same. During the software construction, many of these repeatable parts can be separated or extracted out for reuse. Thus, parts in software have repeatability and an effective software component does as well.

A software program designed on the basis of a conventional programming approach typically leads to difficulty in locating errors or faults due to global effects. Also, a software program designed using an unorganized structure will resolve the debugging problem. As generally known, effective software must be constructed from an effective software component possessing features such as data abstraction and information hiding.

Another important feature of effective software is that it provides a feasible means for programmers to accumulate repeatable parts. Consider the development of computer hardware. From the evolution of transistors, (SSI, MSI, LSI to VLSI), semiconductor technology not only integrates millions of logic circuits in a module, but also accumulates many reusable parts that provide a viable means for application designers to construct a complex and large hardware system without much difficulty. As mentioned earlier, Mcllory 1976 [9] recommended that the software not be constructed from the beginning every time. A more viable alternative is to use repeatable parts to construct the software. Thus, an effective software component technology must be developed.

Theoretically, a reusable software component or framework must be designed in so that it can be used to construct many different applications (maximizing application) and can easily be reused or adapted by a software designer or programmer for his/her application (easy for tailoring in a specific application). Herein, we use a 2-dimensional configuration to depict an ideal RSC or framework, as illustrated in Fig. 1.

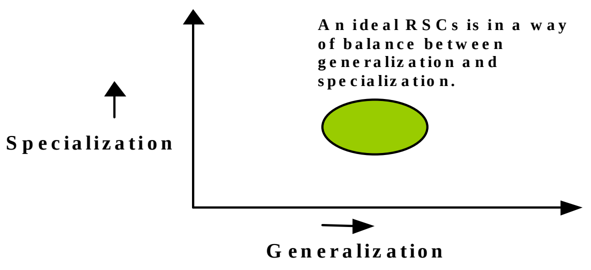
An ideal RSCs is in a way of balance between generalization and specialization.
Specialization
Generalization

Figure 1. An ideal Reusable Software Component

According to Fig 1, assume that an RSC is designed specifically for a certain application, making it quite adaptable for that particular application. However, such an RSC cannot be applied to other applications. On the other hand, if an RSC is designed too general, theoretically, it can be extensively applied to various applications. However, such a design would be difficult to tailor for a specific application. Thus, how to design and implement an RSC so as to create a balance between generalization and specialization is a challenging task.

Figure 2 depicts a client and server model to demonstrate the role of reusable software components, allowing us to more thoroughly describe the preliminary design and implementation of RSCs. In addition, a component can be viewed as a server capable of providing services for its potential clients through the interfaces. A client, an application program, only needs to know the interface specifications but does not need to know how the server implements these services. A server can accept various kinds of requests submitted from clients. If the interface constraints are severely restricted, the servers can obviously assert that although the requests for service from clients conform to their interfacing condition, many potential clients may also be lost whose service requests are just statistically incompatible. On the other hand, if the interface constraints are relaxed, the servers accept various requests of service from clients. This circumstance is a tradeoff between interface compatibility and flexibility. Notably, a component can be a client, a server or a client and server.

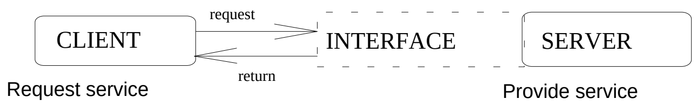

Figure 2. A Client and Server model

The object-oriented approach allows us to construct RSCs. The generalization is achieved by using the multiple polymorphism (i.e. the use of dynamic binding for implementation) and specialization is achieved by using the refinement (i.e. the use of inheritance for implementation) while designing RSCs. Closely examining how to design and implement RSCs allows us to infer that an effective reusable software components possesses the following features: 1) ease of generalization, 2) ease of refinement (specialization), 3) clear interface specification, 4) complete encapsulation, and 5) complete testing. While designed on the basis of the above principles, the reusable software components are the ideal RSCs for accumulating the future reuse.

3.1.2 Categories of software components

Five kinds of software components for software construction can generally be considered as reusable software components, starting from high level down to the lowest level:

1). Requirement specification components

In addition to belonging to a segment of the user requirement, this kind of component can be considered as the analysis level components (reusable analysis components). It has the highest reusability with the lowest applicability property.

2). Design patterns

A design pattern is an abstract design concept that focuses on the particular problem which solutions reuse. According to Erich Gamma's definition, *"Design patterns are descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context"* [13]. Erich Gamma also addressed twenty-three design patterns based on object-oriented design issues. In general, a programming language cannot represent a design pattern itself. It is more abstract than the design framework.

3). Design frameworks

This kind of components is the reuse of design attributed to particular analysis results and it is called reusable software design framework (RSDF) in [1]. The framework is commonly defined as *"a reusable design of all or a part of a system that is represented by a set of abstract classes and the way their instances interact."* Grady Booch defined a framework as *"a collection of classes that provide a set of services for a particular domain"* [14]. Another definition by [15] is *"a framework is the skeleton of an application that can be customized by an application developer".* These definitions obviously reveal that frameworks can be components plus patterns for a particular application domain [15]. A design framework is a design realization, as represented by an object-oriented programming language. Frameworks are more customizable than most components, and have more complex interfaces as well. Programmers must learn these interfaces before they can use them. In general, learning a new framework is difficult.

Frameworks differ from components primarily in that 1). Frameworks provide a reusable context for components. Framework not only implements a design pattern, but also provides standard interfaces for components composition that enables existing components to be reused; 2). Frameworks and components function collaboratively so that frameworks can more easily develop new components; and 3). Frameworks provide the specifications for new components and a template for implementing them.

4). Code components

This component is the primary reusable part in our proposed construction approach. A code component can be considered as a class, from object-oriented paradigm, which includes a complete set of member functions and data structure. Code component differs from the conventional (or functional) library in that it has a well-defined and well-designed interface with data abstraction and information hiding the semantic meaning. Inheritance mechanism in code components provides flexible usage of components. These object-oriented features not only encourage components reuse, but also improve reusability.

5). Data components

Computer data include numerical data and multimedia data. In computer application domain, multimedia data include factors such as text, image, animation, video, and voice, which can be manipulated by the computer. Based on the different kinds of data, a data component can be classified into six types: 1) numerical data such as an electronic constant, 2) text data such as a document, 3) image, 4) animation: 2-D and 3-D animation, 5) video, and 6) voice.

The above types of data can be packaged as reusable data components. These components can be referred to as passive data component since the component is simply a group of constant data definition.

3.2 Multimedia reusable components (MRCs)

Among the control operations on multimedia data include capture, manipulate, edit, store, and display. These multimedia data along with its control operations can be encapsulated as a component. Such a component is called *Multimedia Reusable Component (MRC)*. Based on the features of each type of multimedia data, the data itself and related operations to a component can be composed. With the assistance of the object-oriented paradigm, the message mechanism can be embedded into the component, and then, the component acts similar to an active object.

3.2.1 Features of multimedia data

Compiling multimedia data into MRC hinges on understanding the features of multimedia data. In the following, we thoroughly discuss the multimedia data presented in [8]. The following points can be summarized as follows:

1) Text always uses the standard text format such as ASCII code,

2) Image is a single picture. Images must always be aggregated with another MRC. By doing so, an image’s background must be set to transparent. Most of the image format follows the Microsoft Windows' standard, bitmap (BMP), DIB (Device Independent Bitmap), JPEG and GIF,

3) Animation is a sequence of pictures. Like the image, the background of animation also has to be set to transparent. The data formats include bitmap, FLI/FLC, MOV and AVI in computer,

4) Video is a digitized data from analog video signal that can be controlled as animation. The data formats include MOV, AVI and MPEG,

5) Voice (or Sound) is a digitized data from analog voice signal. The data format includes WAV and MIDI.

Table 1 summarizes the features of multimedia data. In actual applications, the above-mentioned multimedia are transformed into another format due to the compression and security considerations.

Table 1. Summary of multimedia data

Media type	Format	Characteristics	Related Information
Text	ASCII	1. can be aggregated with other types of data	File size
Image	BMP, DIB JPEG, GIF	1. background transparency 2. can be aggregated with other types of data 3. time independent	Format, Resolution and # of colors, Height, Width, File size
Animation	BMP, AVI, FLI/FLC, MOV, MPEG	1. background transparency 2. can be aggregated with other types of data 3. time dependent 4. needing synchronous with voice.	Format, Resolution and # of colors, Height, Width, Total frame, Frame rate, File size
Video	MOV, AVI, MPEG	1. can't be aggregated with other types of data 2. time dependent 3. voice is a part of the data	Format, Height, Width, Total display time, Frame rate, File size
Voice Or Sound	WAV, MIDI	1. can be aggregated with other types of data	Format, Quality(bits per sampling), Sampling rate, Total time, File size

3.2.2 Object-oriented paradigm and reusable multimedia components

Object-oriented paradigm is a natural model for computer animation [17]. In general, object-oriented paradigm is quite appropriate for computer multimedia. Consider a multimedia presentation system. An animation object and a voice object are presented simultaneously, and should be synchronized as well. Restated, these two objects start and end at the same time point. By mapping the situation to real world model, these two objects are "alive" and can communicate with each other. In an object-oriented system, objects communicate with each other via a message mechanism. The message mechanism is also used to perform entities, execute concurrently and synchronize between entities. Based on the object-oriented paradigm and technology, multimedia reusable component is defined herein as a composition of media data, possible operations, and message mechanism.

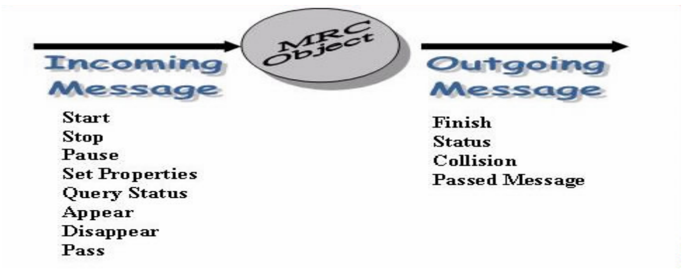

Figure 3. Possible messages for an MRC

An MRC object can receive messages and send messages, as indicated in Fig. 3. The most common incoming messages include start, stop, pause, set properties, query status, appear, and disappear. The MRC responds when a message arrives. For instance, the *start* message causes the MRC object to present its media data. The *disappear* message causes the MRC to hide its media data from display area regardless of whether or not the object is active. Notably, some messages force the MRC object to respond by a send out message, such as *query status* message. The outgoing messages include finish, object status; collision with other objects and, then, pass the incoming message. For instance, while the object completes its presentation, it post *finishes* a message to a system or other objects. System and MRC objects are communicated via such messages. In addition, the sequential objects presentation and parallel objects presentation can be performed as well.

MRCs have different features than reusable software components since multimedia data differ from other conventional computer data:

1) Multimedia data always require a larger storage space and the retrieval depends on the data content. The component storage and management problem must be considered as well;

2) Multimedia data are retrieved on the basis of the data content;

3) Each type of multimedia data has its own features. For instance, the display of text or image is time

independent but voice, animation and video are time dependent;

4) Several MRCs must be synchronized while they are displayed at the same time. For instance, to show an animation and voice simultaneously, they must start and end at the same time point;

5) MRCs not only include multimedia data itself, but also the operations, which can manipulate them; and

6) MRCs are implemented by the object-oriented technique. Multimedia data and related attributes are encapsulated and protected in an MRC. Media data cannot be manipulated from outside of the MRC. *Message mechanism* provides a feasible means of controlling and communicating with MRCs. An MRC is reacted according to what message has arrived. For instance, an MRC starts to appear while receiving a start message. MRCs can also be organized by a message chain, such as MRCs synchronization, serial presentation, and parallel presentation.

In addition, MRCs aggregate as a coding process, which produces a software program. The program can be executed as a multimedia film. If the program represents a software requirement, then the aggregation of MRCs is a process in writing a multimedia software requirement.

3.2.3 An MRC example

Figure 4 depicts an E-Cash MRC example. The animation is recorded as a bitmap picture that consists of several frames. The AnimMRC class is derived from the ActorMRC class. All the possible operations and attributes of logical actors are defined in the ActorMRC. Note that all physical control, such as display a frame on to screen, play a wave data by calling MCI, of a media data is defined in the *BaseMRC* class and the *Component* class. Parts of the class hierarchy definition are listed in table 2.

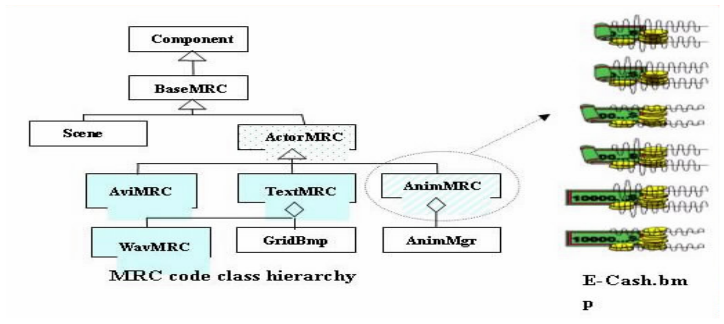

Figure 4 An MRC example: code component + media data

An E-Cash MRC is an animation, which consists of a bitmap picture (E-Cash.bmp), a code component (the AnimMRC class), and related attributes. Table 3 depicts the attributes of the E-Cash MRC, including object name, visible flag, total frames in the bitmap picture, moving path, presentation speed, etc. These attributes are recorded in a script file.

Table 2. Partial implementation of class AnimMRC

```
/* class vFramesMgr */

class _export vFramesMgr {
public:
  vFramesMgr(int fCount, const TColor&transC,
        HPALETTE hPal=0, BOOL tmpPal=FALSE);
  virtual ~vFramesMgr();
  virtual BOOL PtIn(const TPoint&testPos)=0;
  virtual TBitmap*FramesBmp(int fStart,int fEnd)=0;
  TBitmap*FramesBmp();
  TDib*     FramesDib(int fStart,int fEnd);
  TDib*     FramesDib();
  const TColor TransC();
  HPALETTE Palette();
  TSize FrameSize();
  virtual void ToSelFrame()=0;;
  BOOL ToFirstFrame();
  BOOL ToLastFrame();
  BOOL ToPrevFrame();
  BOOL ToNextFrame();
  int operator==(const vFramesMgr&obj);
  void SelSeg(int id);
  void AddSeg(vExtent*pSeg,int id,BOOL sel=FALSE);
  int    SelSeg();
  int SelFrame();
  void SelFrame(int idx);
};

///////////////////////////////////////////////
/* class vBMPFramesMgr */
```

```
/* class vVisual */
class _export vVisual {
public:
  vVisual(){ m_visual=TRUE; m_rect=TRect(0,0,0,0); m_redrawR=TRect(0,0,10000,10000); }
  virtual ~vVisual(){}
  virtual BOOL PtIn(const TPoint&point) { return Visual()&& m_rect.Contains(point); }
  virtual void Draw(TDC&)=0; // before drawing, had better test if Visual() or not
  virtual void Rect(const TRect&newR) { m_rect=newR; } // set region of object
  const TRect&Rect() const { return m_rect; } // the region of the object
  // set the center point of the object ( usually used to move the object )
  void TopLeftPos(const TPoint&pos) { Rect(TRect(pos,Rect().Size())); }
  inline void CenterPos(const TPoint&pos);
  inline void CenterPos(const int x,const int y) { CenterPos(TPoint(x,y)); }
  inline TPoint CenterPos() const ; // return the center point of the object
  void Visual(BOOL visual) { m_visual=visual; }
  BOOL Visual() const { return m_visual; }
  int operator==(const vVisual&obj) { return this==&obj; }
  void NeedRedrawR(const TRect&r) { m_redrawR=r; }
};
```

```
class _export vBMPFramesMgr : public vFramesMgr {
public:

  virtual BOOL PtIn(const TPoint&testPos);
  void PtInType(int ptInType);
  virtual void ToSelFrame();
  virtual TBitmap*FramesBmp(int fStart,int fEnd);
  TBitmap *Bitmap();
  void Bitmap(TBitmap*pBmp,BOOL autoDel);
  TRect       RectFrame(int idx) const;
  TRect       RectSelFrame() const;
  int operator==(const vBMPFramesMgr&obj);
};
```

```
#include "framesmg.h"
#include "vvisual.h"

// class AnimMRC
class _export AnimMRC : public vBMPFramesMgr, public vVisual {
public:
  AnimMRC(…);
  virtual void Draw(TDC&dc);
  virtual BOOL PtIn(const TPoint&point);
  void DrawGray(BOOL drawGray)
{ m_drawGray=drawGray; }
};
```

Table 3. Attributes of E-Cash MRC

```
OBJ_NAME=E-Cash
VISIBLE=1
POSITION=0 0 0
TOTALSTEP=6
SIZE=160 101 100
ACTIVERGN=2
0   0 0 0 0   0 0   NULL -1 0 0   0 1 0
4   0 0 0 0   0 5   NULL   0 1 0   1 1 0
BMPFILE= E-Cash.bmp
ACTIONSPEED=6

[COMMENT]=0
```

3.3 Standardization of the component

Components without a corresponding organizational structure and format are extremely difficult for users to accumulate it and to reuse it for an application. In the following, we present the standardization of RSC and MRC.

3.3.1 Classification and naming

An effective organizational structure and a effective naming system are a prerequisite for reusable components. These facilitate the user in looking for the necessary components. Moreover, complicated organizational structures and naming systems create unnecessary difficulty for locating an appropriately designed RSC and MRC.

In the following, we describe the approach to classify RSCs and MRCs. Basically; a software program can be modeled as a composition of data structures and algorithms by using various tools and subsystems in a computer system (e.g., operating system, compilers and other useful software). Logically, the following hierarchy can be used to classify RSCs and MRCs, as illustrated in Fig. 5. Such a classification is made herein for the organization. The first level is a domain, and the second level is the component's type, i.e. structures, tools, and subsystem.

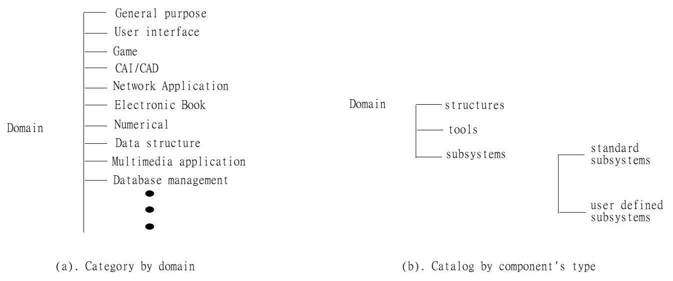

Figure 5. Component categories

Also, a naming system for the RSC and MRC is necessary. Component naming allows the user know the component function by examining the component's name. A relatively easy means of performing this task is to separate component name into three parts. By using the first letter to represent the application domain, the second letter to represent the catalog of the component, and the remaining one is the component function name. The illustrative example in Fig. 6 indicates that the component is a windows game component, which is used for playing a video.

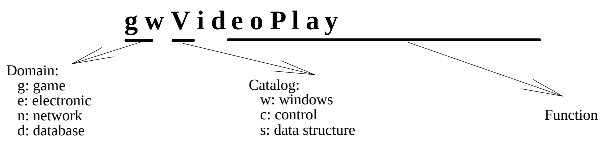

Figure 6. An illustrative example for the naming system

3.3.2 Components information

Component standardization facilitates the users in retrieving, managing and presenting the component. In the following, we present an architectural layout for the reusable software components [2]. Basically, it has three major parts: 1) reusable software component specification, 2) reusable software component body, and 3) component layout. Detailed information regarding these three parts are described thereafter.

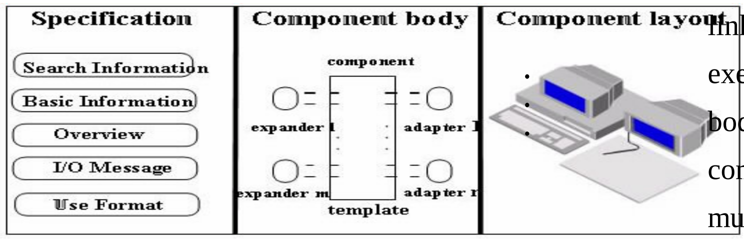

Figure 7. Component information

Specification

1). **Search information:** consists of the component name, author, taxonomy code, application domain, creation date, and modification history. Some of this information can be used as the keyword during component searching.

2). **Basic information:** consists of the component size, component type, version, and file name in system.

3). **Overview:** describes the basic functionality and features of the component. For an MRC, the component overview presents the media data.

4). **User interface:** describes the public interfaces (public member functions) supported in the component for user application and all operations on the media for MRC.

5). **Input/Output message:** describes the meaning and format of parameters that are passed onto the component through user interface. For an MRC, the I/O message also includes types of events, which the component can accept and submit.

6). **Use format:** consists of an example program segment that demonstrates how one can use the component.

Component body

Component body is the implementation part of the component. Following the compilation, the object code of the body is included into a library (the extension file name is lib). It is directly linked at linking phase with the main program to produce an execution code. In this manner, the implementation body can be hid from the users. For an MRC, the component body includes the code component and multimedia data.

Component layout

Component layout uses multimedia data to represent the component in visual form. This part of the component is useful in visual programming environment. The extension file depends on media data type, such as bmp, wav, and avi.

For MRC, each component should have the above-mentioned related file and some extra information for the use of MRC during implementation, which can be summarized as follows:

1). Script, each MRC has its own properties based on the media data. These properties are recorded in a separate file named script file. While compiling MRCs into a scene, this script file is included into the scene script. Chapter 4 elucidates the script design and the use of MRC. The extension file is spt;

2). Demo scene, how to use the MRC to represent a requirement scenario is demonstrated. More detail will be discussed in section 4;

3). Formal requirement specification, denote the possible requirement specification while using the MRC to represent a requirement segment. The extension file name is spc; and

4). Framework, denote the possible framework while using the MRC to represent a requirement segment. The extension file name is cpp.

Above information provides the basis for users to retrieve and to use the existing components for their application.

四. 計畫成果自評 (1/3)

Software reuse is an effective approach to improve software productivity and quality. Many reusable components have been designed and used for various applications. These reusable components include many commercialized application frameworks, design patterns, and code components (both in source code form and binary code form). In this study, we propose the concepts of RSCs and MRCs. A reusable software component (RSC), a software module, is implemented on the basis of object-oriented programming paradigm that can be repeatedly used in software construction. Five kinds of software components for software construction can be considered as reusable software components: requirement specification components, design patterns, design frameworks, code components, and data components. This year's study extends reusable software components to incorporate with multimedia data as Multimedia Reusable Components (MRCs). In other words, the MRCs include not only code and

documentation, but also voice narration and possible animation sequences. An MRC should handle the media data itself, such as start and stop presentation, and edit, and should communicate with other components via a message. With the employment of an object-oriented paradigm, MRC is defined herein as a composition of media data, possible operations, and message mechanism. An MRC can be viewed as an active object and can be mapped to real world model. MRCs can be aggregated into a multimedia film, which represent a software requirement. As planned, we have achieved the goals of the first year. And we will focus on the issues 2 and 3 for the next year project. Specifically, a script language to represent the visual representation is proposed and an authoring tool to create a multimedia representation will be designed and implemented.

五. 參考文獻

[1] D. J. Chen and David T. K. Chen, "An experimental study of using reusable software design frameworks to achieve software reuse", The Journal of Object-Oriented Programming, May 1994, pp. 56-68

[2] P. F. Chen, On the Study of Using Software IC Construction Approach to Achieve Software Reuse, Master Thesis of N.C.T.U. Taiwan, June 1991D. J. Chen and P. J. Lee, "On the Study of Software Reuse Using Reusable C++ Components", The Journal of System and Software, Vol. 20, No. 1, Jan 1993

[3] Grady Booch, Software Components with Ada, Benjamin/Cummings, California, 1987

[4] M. Lenz, H. A. Schmid and P. W. Wolf, "Software Reuse through Building Blocks", IEEE Software, July 1987, pp. 34-42

[5] Brad J. Cox, “Object Oriented Programming-An Evolutionary Approach”, Productivity Products International, Inc. 1986.

[6] Ruben Prieto-Diza and peter Freeman, “Classifying Software for Reusability”, IEEE Software, January 1987.

[7] Chung-Chien Hwang, The Design and Implementation of Multimedia Resource Components for Electronic Story Book, Master Thesis of N.C.T.U. Taiwan, 1995.

[8] M. D. Mcllroy, “Mass-produced Software Components”, in Software Engineering Concepts and Techniques (1968 NATO Conf. On Software Engineering), J. M. Buxon, P. Naur and B. Randell, 1976.

[9] Peter Freeman, “A perspective on Reusability”, The Computer Society of the IEEE, 1987, pp.2-8.

[10]Pascoe, Geoffery A., “Elements of Object-oriented Program.”, Byte, Vol. 11, No. 5, June 1992.

[11]Mitchell D. Lubbars, “Wide-Spectrum Support for software Reusability”, Proceedings of the Workshop on Software Reusability and Maintainability, October 1987.

[12] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, Design Patterns – Elements of Reusable Object-Oriented Software, Addison Wesley, October 1994

[13] Grady Booch, "Designing an Application Framework", Dr. Dobb's Journal, February 1994, pp. 24-31

[14] Ralph E. Johnson, "How frameworks compare to other object-oriented reuse techniques: Frameworks = Components + Patterns", Communications of the ACM, Vol. 40, No. 10, October 1997, pp. 39-42

[16] Dale Rogerson, Inside COM, Microsoft Press, 1996

[17] Breen David E., Getto Phillip H., Apocada Anthony A., Schmidt Daniel G., Sarachan Brion D., "The Clockworks: An Object-Oriented Computer Animation System", Proceedings of Eurographics, pp. 275-282, 1987

[18] W.C. Chen, "A Visual and Reuse-based Paradigm for Software Construction," Ph.D. dissertation, Computer Science and Information Engineering Dept., National Chiao Tung University, Taiwan. June 1998