Establis a Common Software Service Center by Using Functional Integration Technique

Sheng-Chin Lee（李勝欽）[1] Jim-Min Lin（林志敏）[1] Hewijin C. Jiau（焦惠津）[2]
Chi-Tai Lee（李啟泰）[1]
asuka@ultra2.iecs.fcu.edu.tw, jimmy@fcu.edu.tw, jiauhjc@ee.ncku.edu.tw,
cltee@plum.iecs.fcu.edu.tw
[1] Department of Information Engineering, Feng Chia University, Taichung City, Taiwan
[2] Department of Electrica Engineering, National Cheng Kung University, Tainan City, Taiwan

Abstract

Providing a high quality and low cost software service is an important term of computerizing. This demand will increase when there're more groups or customers trying to computerize their systems. This paper offers a feasible approach named Functional Integration Technique (FIT) to establish a Common Software Service Center (CSSC). Any client could acquire software service from CSSC when they need. This way is especially suitable for the workgroups or communities.

Key words. *Software Service, Function Integration, COTS, Software Component, CORBA.*

1. Introduction

Modern software product plays a more important role in the computer world. It not only helps the end users to manage their computer hardware resources but also with some specific problems. Nowadays, software designers tend to design more complex and more powerful software product. As a result, a high quality software service is an essential term in a modern computer system. Always people need to spend a considerable cost to achieve this for two reasons. One is that since users' needs have been increasing, software products will be more complex. This results in the cost of software development and maintenance growing exponentiall . The other is that the end users must pay more cash to buy these software products, and spend more time to learn and manage them. This is a big burden for the customers.

In this paper, a new approach to cope with this problem is proposed . Here, we collect several practical COTS software applications and retrieve their original functions to construct reusable software components. By integrating these software components, new applications will be built down soon. Finally, a Common Software Service Center (CSSC) will manage these software components and provide the mechanism for the customers to request it for software service. This will benefit people working in a workgroup or living in a communit . They can acquire software service from CSSC and need not to buy the software products individually. Moreover, this a pproach will help people who are unfamiliar with operating computers to derive software service from CSSC.

The rest of this paper is organized as follow. In next section, the related work of our system such as FIT, CORBA software component will be overviewed. Section 3 describes the detailed architecture and specification of CSSC, and expounds how to build a CSSC. In section 4, we'll discuss the advantages and confronted problems of our approach, and its influence for software development. The final section is the conclusion and future works.

2. Related Work

In this section, we will illustrate the related techniques and terminologies of CSSC particularl

2.1 Functional Integration Technique (FIT)

The basic concept of FIT is that retrieving each independent application's function as a library or methods of an object. By sending command streams from a control program to the application, the application will execute those functions that the client needs (Fig.1). The way of doing this is wrapping this applicati on with a small program named wrapper. It is in charge of the interception and redirection of the application's I/O for towing the data streams to the appropriate destination. As a result, what we are interested in are those functions provided by this application and the format of its acceptable commands [5]. An example named parking management system could be found in literature [15].

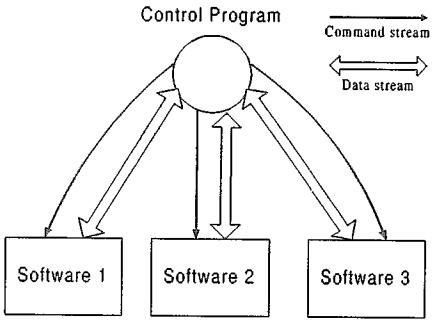

Figure 1. FIT semantic figure

2.2 Software Component

Software components are binary units of independent production, acquisition, and deployment that interact to form a functioning system [2]. In programming disciplines, a component is an identifiable part of a larger program or construction. Usually, a component provides a particular function or group of related functions. In programming design, a system is divided into components that in turn are made up of modules [14]. In this paper, software components play the same role as illustrate above. The only difference here is the implementation. In the CSSC system, a software component is a unit consists of a COTS software application and a wrapper (Fig.2). The wrapper sends command streams to the wrapped application to achieve its function and gets its output results as the software component's output. Any program or sub-routine can invoke the software component with the API provided by the wrapper.

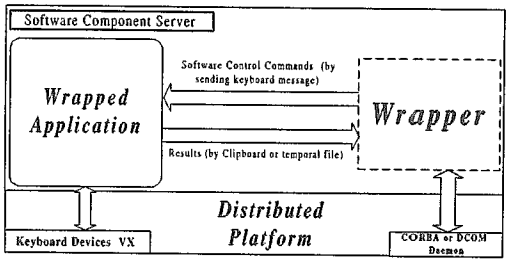

Figure 2. The software component in CSSC

2.3 Wrapper

Here we give a formal definition for the *wrapper* in our domain. A wrapper is a program substituting for the user programs to operate the wrapped application [14]. Wrapper also has the ability to solve some problems that mayb occur when it wraps the application, and it provides standard API so that any program or sub -routine could invoke it easily. The detailed specification of the wrapper could be found in our previous research literatures [5,7,9,10,11,15].

3. Common Software Service Center

Common Software Service Center is an open system and designed for people in a workgro up or communit to acquire software services (Fig.3). The main essence of CSSC is to provide software services from those practical applications for the customers automaticall . Here, we only define the framework of CSSC but regulated the detailed implementation specification. Anyone can establish his own product in his style; compiling the CSSC architecture is the only requirement. In this section, we will introduce the architecture of CSSC and one method we proposed to establish the CSSC.

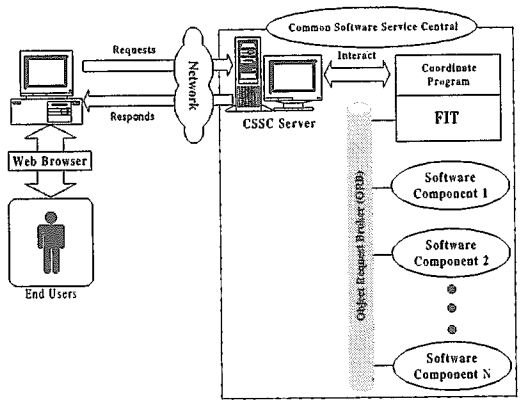

Figure 3. The semantic structure of the Common Software Service Center

3.1 CSSC Architecture

The CSS contains five parts (Fig. 4). We commentate them one by one and illustrate the interaction among these parts as follow.

- **CSSC kernel.** CSSC kernel is the core of the whole system and decides how to serve the customers' requests. It will find out th ose appropriate software components and integrates them into new applications to satisfy the customers' software requirement. In short, it is in char ge of the management and operation of CSSC.
- **Software Component Services.** Software components are responsible to provide the software services. Each component has its unique function. They are stored in the CSSC library. When needed, the CSSC kernel will in tegrate them with the integration mechanism.
- **Integration Mechanism.** It is necessary for the CSSC kernel to have a mechanism to integrate the software components. In our system, we use FIT as the integration mechanism.
- **ORB.** Those software components and CS C kernel itself may work on heterogeneous platform. The CSSC needs an ORB to deal with the underlying network communication and information interchange issues. In our system, we use the CORBA ORB to take this job.
- **Requirement Description.** The requirement description provides the mechanism for the customers to describe their requirements formally. It may be implemented in various forms. For example, a formal scripting language will be one choice.

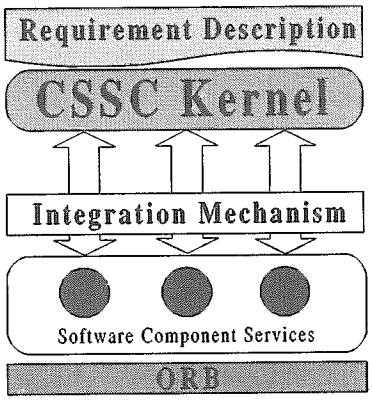

Figure 4. The CSSC architecture

The operation flow of CSSC is shown in figure 5. It follows the steps below.

Step 1: The customer sends his request to the CSSC kernel.

Step 2: The CSSC kernel analyzes the customer's request and classifies them into two types. One is provisional request, and the other is long-term service. If the customer's request would be executed only a few times, such as provisional calculation, data query, or on-line shopping...etc., it will be classified as the provisional request. On the other hand, the requirement wo uld be invoked many times, such as dail accounting, CAI, or multimedia applications...etc., will be classified as the long-term service.

Step 3: The CSSC kernel implements provisional request as a temporal service object, which provides the functions to deal with the requests.

Step 4: The CSSC kernel implements long-term service as an isolated application, which will achieve the software services.

Step 5: The service object or the application will execute to

achieve its services.

Step 6: After the service object or application executed, CSSC kernel will return the final results to the customers.

Step 7: If the service is a service object, it will be released after it finishes the task.

Step 8: If the service is an application, the CSSC kernel will store this application into its library after it had finished its task. When the customers request for service next time, it could serve the requests by invoking this application again.

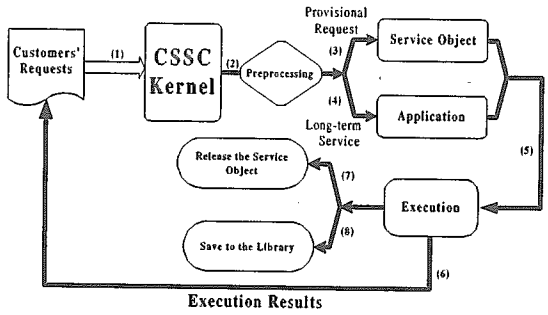

Figure 5. CSSC system operation flow

3.2 Establish the CSSC

We hav notified that the CSSC is an open system. As long as the designer follows the architecture of CSSC, anyon can build up the CSSC in his own style. Depending on the need, the designer can replace any part of the CSSC. Here, we will introduce one method to establish the CSSC.

3.2.1 Construct Software Component Services

Software components provide the actual functions for software services. The reader can find the definition of software component of the CSSC system in section 2.2. By using FIT, we construct these software components by wrapping COTS software applications as the following steps.

Step 1: Identify the software component, such as its function or interface.

Step 2: Find out an appropriate COTS software application that provides the actual functions the component needs.

Step 3: Analyze this application to induce its correct command stream to wrap it.

Step 4: Design the corresponding wrapper program.

Step 5: Combine the wrapper and the wrapped application into the software component.

The detailed implementation could be found in our previous papers [7,9,11,15].

3.2.2 The Integration Mechanism and ORB

Synthesizing our pre-definition above, we use CORBA as the underlying component communication mechanism. As a result, each component is regarded as a CORBA object. The CSSC kernel will creates a coordination program as the CORBA object container to integrate these CORBA with the FIT policy. Then it will achieve its task.

3.2.3 Requirement Description

Requirement description is the mecha nism providing the customers a formal method to describe their requirements. For better and easier design, it will be divided into two parts named *customers' user interface* and *description mechanism.*

Customers can only interact with the CSSC by customers' user interface, so it is important to design a user-friendly interface. The web browser is used to take this job since its ease and common. From customers' view, the CSSC kernel is like a web server. It provides some web pages that contain Java applets to communicate with the coordination program. Java applet s on the web page can communicate with other CORB objects, such as coordination program, with the *OrbixWeb ORB*. As a result, the customers can send their requests to the CSSC kernel with the web browser easil (Fig.6).

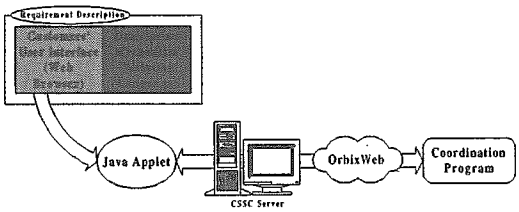

Figure 6. Requirement description semantic

Description mechanism defines the specification of the description. The customers have to write down their requirements with the formal description rules. The CSSC specification doesn't formulate the standard content of description mechanism, but we suggest the designers using a formal script technique such as a simple script language as its implementation.

In our example, we define a Component Integration Language (CIL) to be the requirements script language. CIL is a simple script language. It defines an instruction set and related syntax to glue the software components into new applications. As we mentioned before, the customers' requests could be classified into two types named provisional request and long-term service. In CIL, it also provides instructions to support this classification. Currently, we have finished the prototype of CIL, and the full edition will be our future work.

The operation flow of the custome rs writing their requests in CIL is shown in figure 7. The process goes as the following.

Step 1: The customers write down their requests in CIL.

Step 2: The CIL compiler compiles the CIL program. If the CIL program is correct, the CIL compiler will produce the CIL command code and send it to the CSSC kernel.

Step 3: After the CSSC kernel gets the CIL command code; it will execute it with the CIL VM. The CIL command cod will invoke some software components to achieve its functions.

Step 4: Final, the CSS kernel passes the execution result to the customers.

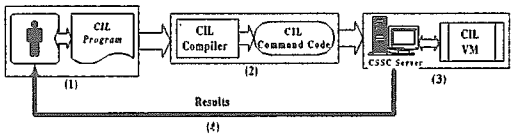

Figure 7. The operation flow of the customers writing their requests in CIL

3.2.4 The CSSC kernel

By definition, the CSSC kernel is in charge of the management and the operation of the CSSC. It is hard to describe in every detail what the CSSC kernel will be because there's no completely adequate definition of the CSSC kernel. But roughly we can define here is that all of the management policy and physical service routine are brought together into a system, and that is the CSSC kernel. Here, one example is provided to illustrate how to define and establish the CSSC kernel. The structure of the CSSC kernel is shown in figure 8.

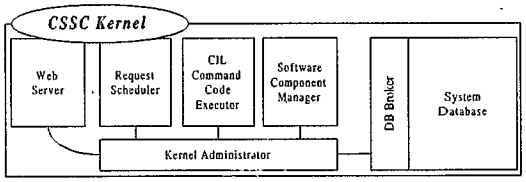

Figure 8. The structure of the CSSC kernel

- **Kernel Administrator** Kernel administrator is the console of the CSSC kernel. It is in charge of managing and coordinating other routines of the CSSC kernel accordin to its management policy. If there is a customer asking for servi ce, it will dispatch a suitable service routine to serve the request. For example, if a CIL command code is sent to CSSC kernel, kernel administrator will inform the CIL command code executor which invokes the CIL VM to interpret the CIL command code.
- **Request Scheduler.** Request scheduler schedules requests from the customers. Customers request CSSC with CIL command code. To identify each request from different customer in different time,

CIL compiler adds a tag to each CIL command code. The tag is encoded with customer's identification and the request time. The encoding polic guarantees the uniqueness of the tags. Request scheduler will acknowledge kernel administrator of the scheduling result after completing the task.

- **CIL Command Code Executor.** The CIL comman code executor invokes the CIL VM to interpret CIL command code and pass execution results to the place assigned by kernel administrator.
- **Software Component Manager.** Software component manager maintains the information of each software component in th e system database. When kernel administrator or other programs need to invoke the software components, it serves the requests.
- **DB Broker.** DB broker is the only one can communicate with the system database directly. It provides a set of functions for other routines to access the system database. Also, it is in charge of maintaining the system database
- **System Database.** System database is a comprehensive database. It contains wrappers, information of the software components, records of the customers, system daily l s, and others.
- **Web Server.** As customers view the CSSC as a web server, the CSSC kernel has to provide a web server in the front end. The web server is the same with the other web servers, but controlled by the CSSC kernel. In our system, we use the Microsoft IIS set up the web server on the Windows NT platform.

The designers can regard CSSC kernel as the operating system of the CSSC system. Any resource, operation management policy, and corresponding physical service routines can be added into the C SSC kernel. Moreover, designers can add other elements for necessar .

4. Discussion

CSSC indeed brings some advantages for designers to develop their software services. Comparativel , the designers will confront some problems when they develop their own software services by using this approach. We'll discuss these possible issues in this section.

4.1 Reducing the development cost

Traditionall , the designers need to develop their software products from scratch. The development process includes drawing up the software specification; designing the detailed algorithms and codes; implementing the software products, and testing and maintenance. Each phase will consume lots of manpower, recourse, money, and time. When the requirement becomes more complex, the development cost will increase exponentiall . The major reason causes this problem is that the designers have to take care ever detail of the development process such as how to design the user interface, how many object need to be created in the program...etc . All of these issues will increase the designers' burden.

On the contrary in the CSSC system, the designers just only need to collect some practical applications and wrap them into software component services. When a customer request for software service, the CSSC service mechanism will find out the appropriate software components to serve the request automatically. Besides the physical software services supported by the software components, other elements in CSSC such as web server or ORB are all standard services that can be set up by using COTS tools, like IIS or *Orbix*. As a result, instead of developing everything themselves, the designers can finish most of the development tasks by reusing off-the-shelf software products. Then the development cost will be reduced substantiall .

4.2 Increasing the software service reliability

The designers can develop most of the software services of

CSSC by using off-the-shelf software products. The most obvious advantage of doing this is increas ing the reliability. Because those off-the-shelf software products are verified in the market, software service reliability is better than software developed from scratch.

4.3 Ease for use and maintenance

For customers, as long as they describe their requirement based on the requirement description rules, they can get the services conveniently. If the customer is unfamiliar with operating software application, he can ask the CSSC for software service and doesn't have to operate this application himself. Moreover, it is not neces sary for the customers to organize and manage those software applications. The CSSC system will handle the rest and get the work done.

For CSSC administrators, the most important thing is to maintain the CSSC kernel to make sure the CSSC system works correctly. These tasks include manage the system database, update the web server, add new software component services, and manage the customers' records, etc. They don't have to maintain and debug the source code of the software components exactly.

4.4 Applicability

The CSSC system is designed for acquiring low cost and high quality software services. For sharing recourse, the customers can add their own software components on the CSSC and let other customers to access them. The maximum benefit of doing this is that the customers need not to pay much cash to buy all of the software applications they need, and they don't have to concern the organization and management of these various applications. The CSSC system will help the customers to deal with this probl em. We suggest that people work in a workgroup or live in a communit can build up their own CSSC to share the software services.

4.5 Concurrence for multiple requests

CSSC system works in a multitasking and distributed environment. It means there will be multiple transactions requesting for services. In CSSC system, the designer invokes the COTS software applications as the software component services. Nevertheless, most of these COTS software applications are designed for single-user so that they can't be shared. Then the critical section and synchronization problem will occur. Sometimes it also involves the license and IP (ntellectual Property) problems. A mechanism must be provided to handle these problems. In our previous paper [10], we had proposed one scheme to deal with it. The designer can add a mature synchronization scheme such as timestamp -based protocols into the wrapper. It will cope with this issue when the customers request the software component services.

4.6 Efficiency

Efficiency is one key evaluation of a software service, and we are dissatisfied with this issue in the CSSC system. The reason is that only a few parts or functions of a COTS software application will be used for the software component services. To retrieve these functions, the software component has to involve the entire application. It will increase the CPU loading, and waste memory space. As a result, the efficiency of the software component services becomes not ideal. This situation will be worse when we add many components to CSSC system.

5. Conclusion And Future Work

In this paper, we propose the CSSC system for software designers to develop software services rapidly. Not only be beneficial to designers, but also decrease the heav burdens of customers. Our design philosophy is utilizing the off-the-shelf software products and techniques to establish new software services. Reducing the development and management cost is the principal goal. We believe this is a feasible approach of developing software system for

software designers.

There are still some open topics of the CSSC. First, we have to improve the efficiency and the stability of CSSC. Second, to generate the wrapper automatically. Final, although the requirement mechanism is formal, it is not pure user-friendl since it uses a script language as its description. To switch it into a graphics user-interface and tries to build up an expert system for the customers to query it for software services is need.

Acknowledgement

This research was supported by National Science Council, Taiwan under Grant *NSC88-2213-E-035-005*.

Reference

[1] Charles Petzold, "Programming Windows 95," Microsoft Press, 1996

[2] Clemens Szyperski, "Component Software," Addsion-Wesley Publishing Company, 1998.

[3] Her Gen, Liu Tin., Lai M.J., "Software Revolution Beyond the Millennium the 2nd Edition," Inforist Press Corp., 1996.

[4] IONA Technologies, Orbix Advanced Programmer's Guide. 1994.

[5] Jim-Min Lin, "Cross-Platform Software Reuse by Functional Integration Approach," Proceedings of COMPSAC 97, pp.402 -408, Washington, D.C., USA, Aug.11-15, 1997.

[6] Lan Sommerville, "Software Engineering," Addison-Wesley Publishing Company, 1995.

[7] R.C. Lin, C.T. Lee, and Jim-Min Lin, "Reengineering MS-Windows Software Applications into CORBA Services", Proceedings of 1998 Workshop on Distributed System Technologies & Applications, pp.201-208, Tainan, Taiwan, May 1998.

[8] Robert Orfali, Dan Harkey and Jeri Edwards, "Instant CORBA," John Wiley & Sons, Inc., 1997.

[9] Sheng-Chin Lee, Jim-Min Lin, Hewi Jin Jiau, and Winston Lo, "Reengineering MS-Windows Applications into Reusable Software Components with DLL Format," Proceedings of the 4th Workshop on Multimedia Technology and Applications Symposium (MTAS '99), Apr. 1999.

[10] Sheng-Chin Lee, Jim-Min Lin, and Hewi Jin Jiau, "Reengineering Single-User/Single-Tasking Applications into Shareable Software Components Under a Distributed Environment," Proceedings of 1999 Workshop on Distributed System Technologies & Applications, Tainan, Taiwan, May 1999.

[11] Ta-Hsin Wu, Jim-Min Lin, and Hewi Jin Jiau, "Reusing UNIX Software Applications Under CORBA Environment by Using Wrapper Technique," Proceedings of the 9th Workshop on Object-Oriented Technology and Applications, pp.1-6, Sep. 1998.

[12] Wen-Zhi Chen, "Microsoft IIS Web Techniques," Kifon Corp., Dec. 1997.

[13] Whatis.com, The Definition of Wrapper, URL:http://www.whatis.com/, 1998.

[14] Whatis.com, The Definition of Component, URL:http://www.whatis.com/, 1998.

[15] Yee-Chong Pan, Jim-Min Lin, and Win-Tsung Lo, "Wrapping Existing Software Tools as CORBA Objects," Proceedings of the 8th Workshop on Object-Oriented Technology and Applications, pp.187-193, Sep. 1997.