Migrating Legacy Software Systems to CORBA based Distributed Environments through an Automatic Wrapper Generation Technique

Hyeon Soo Kim
School of Comp. Eng. and Software Eng., Kum Oh National University of Technology
188 Shin Pyung Dong, Kumi, Kyung Buk 730-701, South Korea

James M. Bieman
Computer Science Department, Colorado State University
601 S. Howes St., Ft Collins, CO80523, USA

Abstract

One of the strategies for migrating legacy systems to distributed object-oriented environments is wrapping. Wrapping is a method of encapsulation that provides well-known interfaces for accessing legacy systems. The advantage of wrapping is that legacy systems become part of the new generation of applications without discarding the value of the legacy applications. There, however, are many styles for interfacing with legacy systems. Application developers who want to migrate legacy systems to new environments and to use them have the burden of understanding and implementing various interfacing techniques. To solve this problem, we construct the extensible wrapping template classes for various interfacing styles and present an automatic wrapper generation method based on them.

Keywords: Legacy Software, Distributed Environments, CORBA, Wrapper

1. Introduction[*]

The innovative changes of internet environments and the introduction of CORBA eases the transition from mainframe based centralized legacy systems to more flexible object-oriented distributed systems. Here the *legacy software systems* refer to system software and application software designed and implemented with old technology (that is, without distributed concepts and object technology). Although legacy systems were implemented with old technology, they still provide value by performing crucial work for their organizations, and usually they represent a significant investment and years of accumulated experience and knowledge [2].

According to Sneed[6], there may be three strategies for introducing distributed object concepts into existing legacy software systems. One strategy is to start from scratch and redevelop all of the business application with the distributed object concepts. This approach frees the developers from any consideration of the existing systems. But, every function must be reimplemented and tested in a new language in a new environment, which is expensive and time consuming.

Another strategy is a reengineering approach. Engineers convert the programs of existing systems to object-oriented programs and distribute objects appropriately. This approach is a promising method since it is not necessary to reimplement functions whose functionalities are the same as the functionalities of the older systems. However, code conversion is not easy. Few tools and methods are available.

The third strategy is to wrap components of the existing systems and to invoke them from the object-oriented distributed environment. Wrapping is a method of encapsulation that provides clients with well-known interfaces for accessing server applications or components. The advantage of wrapping is that legacy systems become part of the new generation of applications without discarding the value of the legacy applications. Wrapping is a compromise approach. The construction of object-oriented distributed systems with first or second approach is maybe the ultimate goal. But the explosive increasing of demand for applications based on internet technology and object-oriented distributed environments does not permit the necessary time delay. Wrapping is a realistic approach, since it is accomplished easily and rapidly with current technology.

To use wrapping, application developers must understand and implement the interfacing techniques to legacy systems, and there are many styles of interfaces.

[*]This work was supported by postdoctoral fellowships program from Korea Science & Engineering Foundation (KOSEF).

The goal of this research is to cope with the difficulties. We investigate problems that should be solved in order to wrap legacy systems and suggest an effective wrapping method. We construct extensible wrapping template classes for various interfacing styles and present an automatic wrapper generator based on them to alleviate application developers' burden. By using the method, we are able to extend the usefulness of legacy applications by facilitating their migration to CORBA based distributed environments with minimum efforts.

The rest of the paper is organized as follows. In section 2, we briefly introduce CORBA and object wrapping techniques. In section 3, we explain the strategy for migrating legacy systems to CORBA based distributed environments. In section 4, we describe the wrapper generation method. In section 5, we present an example to demonstrate the effectiveness of wrapper application. Section 6 describes related work. And finally section 7 presents concluding remarks.

2. Backgrounds

2.1 CORBA

CORBA (Common Object Request Broker Architecture) was developed by the Object Management Group, a consortium aiming to achieve interoperability standards on all levels of an open market for object technology. The goal behind CORBA is to enable open interconnection of a wide variety of languages, implementations, and platforms [5]. Figure 1 gives a detailed view of CORBA.

Software developments in CORBA environments are performed at two sides: client-side and server-side. Server-side application developer implements **object implementations** for services, and then describes the interfaces to the provided services with IDL (Interface Definition Language). A client-side application developer makes reference to the IDL written by a server-side developer, and implements **client applications**.

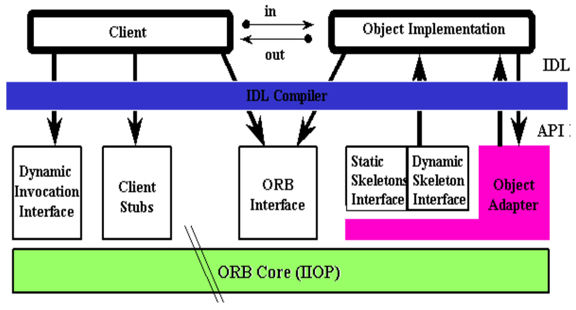


Figure 1. CORBA

2.2 Object wrapper techniques

Wrapping techniques provide a natural way of integrating legacy systems with each other and with new software. Object wrapping is related to new systems integration concepts, including layering, migration, reengineering, reverse engineering, and forward engineering. Mowbray et al. [4] introduce some of these concepts and describe how they can be applied to object wrapping.

1. Layering

Layering is a mapping from one form of application program interface (API) to another such that it maps one set of operations onto a completely different set. Layering can be done without modifying the underlying API design. An application example is the layering required for OpenDoc parts to interoperate with OLE2 components.

2. Data Migration

It involves moving the data used at legacy systems to another data model, for example, relational, extended relational, or object-oriented database systems. Sometimes it is used to map between database systems made by various vendors. Wrapping involves adding layering code to provide access to the other database. This approach focuses on reuse data rather than the functions of legacy systems.

3. Reengineering Applications

Usually reengineering is performed to reduce costs, to increase performance, and to enhance maintainability of the system. The process of reengineering comprises that a system is first analyzed, then changes are made to the system at the abstract level and the system is reimplemented.

For example, consider a situation where a legacy system is reengineered into an object-oriented system. The system model maps the application domain modeled as objects and associations into elements of the existing system, and an analysis model is created. With this analysis model, the work is performed to determine which parts of the existing system will be reimplemented using object-oriented techniques. After a new subsystem is designed using object-oriented techniques, it should interface to the remaining old system components. At this time, wrapping allows for the replacement of the old system components with object-oriented components.

4. Middleware

Middleware can be divided into two categories: distributed processing middleware and database and user interface middleware. CORBA is an example of distributed processing middleware. Database middleware acts as a mediator between various database products and provides a common access mechanism. The wrapped database middleware can become a data brokering service and provides format conversion between different formats and query conversion between query language dialects.

5. Encapsulation

Encapsulation is the most general form of object wrapping, it separates interface from implementation. CORBA encapsulations hide differences in programming language, location, operating system, algorithm, and data structure with the IDL. Encapsulation can be used with the legacy system when the system cannot be modified, because its code is inaccessible. In this case, all access can be provided via the wrapper. Encapsulation can also be used to partition and componentize a legacy system. Each component can be encapsulated separately, and then the system can be reintegrated using object-based communications.

3. Migrating Legacy Systems to CORBA Environments

Figure 2 shows the migration of legacy systems to CORBA environments. The client application in Figure 2 corresponds to **client** in Figure 1. The wrapper objects, depicted by ovals, correspond to **object implementations** in Figure 1. Using CORBA, a server-side developer should implement each wrapper object. This work burdens him with understanding both interfacing techniques and functions of legacy system. Actually, it is very difficult to understand and implement various interfacing techniques. Our approach is to alleviate this burden using an automatic wrapper generation method. Because, using our approach, wrappers include the implementation details for various interfaces to legacy systems. Developers only need to understand services of legacy systems and to describe them in IDL.

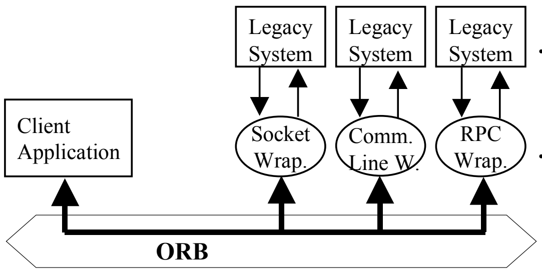


Figure 2. Migrating legacy systems to CORBA

4. Approach to Automated Wrapper Generation

4.1 Problems

To migrate legacy systems to CORBA environments, we solve the following problems:

- **Variety of the interfaces to legacy systems**

There are many interfacing styles in legacy systems, they have different implementations each other and are also dedicated. Thus, it is difficult for server-side application developers to implement wrapper objects for legacy systems, even though he understands some of the interfaces to the legacy systems. To cope with this problem, we propose WTC (Wrapper Template Classes) and an automatic wrapper generator based on WTC. The automatic wrapper generator sets developers free from needing to understand and implement various interfacing techniques.

- **Representation of interfaces to legacy systems**

To generate wrappers automatically, a server-side developer should submit interfacing information for legacy systems to an automatic wrapper generator. Thus, some representations are required to describe easily the interfaces to legacy systems. For this, we propose EIDL (Extended IDL). In any case, a server-side developer should describe the services of legacy systems with IDL for clients, in other words, he has to describe the interfaces to wrapper objects. The EIDL provides one way of describing the services of legacy systems for clients and another way of describing actual interfaces to legacy systems for a wrapper generator.

4.2 EIDL (Extended IDL)

The EIDL is an extended version of IDL. The EIDL is composed of IDL parts and LI (legacy interface) parts. Figure 3 shows an example EIDL definition. Here, the commented part, i.e. a line with leading double slashes, is LI part and the others are IDL parts. Each operation in IDL parts has a corresponding LI part.

```
module Wrapper {
    interface C_lang {
        string compile (in string name, in string inFile);
            // <-CMDLN : execute "cc $0.c -o $0" ->
        ….
    };
    ….
};
```

Figure 3. An example of EIDL definition

1. IDL parts

This conforms to IDL style of CORBA. The client-side applications are able to use the services of legacy systems only via the interfaces declared in IDL parts of EIDL. In fact, these interfaces are interfaces to the wrapper objects wrapping legacy systems. Thus a client-side developer is able to use abstractly the services of legacy systems without detailed knowledge of actual interfaces.

2. LI (Legacy Interface) parts

The actual interfaces to legacy systems are described in the LI parts. They consist of the legacy system interface type, the name of actual service of the legacy system, and arguments transferred to the legacy system. A server-side developer writes this part with appropriate commands as if he uses services of the legacy system in a stand-alone environment, except that he adds an interface type. For example, to use the C language compiler, he just writes, "cc $0.c –o $0". Here, $0 means the first parameter of an IDL operation. It maps name in this example.

4.3 WTC (Wrapper Template Classes)

In this research, we categorized interfacing styles of legacy systems as follows: Command line, Socket, File, Signal, RPC, FIFO. For this work, we referred to reference [4] entitled 'Essential CORBA'. We implemented wrapper template classes corresponding to the identified interfacing styles, respectively. The class hierarchy for WTC is shown in Figure 4. Here, the lowest classes are template classes. This figure shows each wrapper object, depicted by an oval, which is instantiated from the template classes. Of course, the above categories for interfaces may not cover all the interfacing styles of legacy systems. Because, however, the currently developed classes may cover many interfacing styles of the existing legacy systems, it will be not difficult to actually apply these classes.

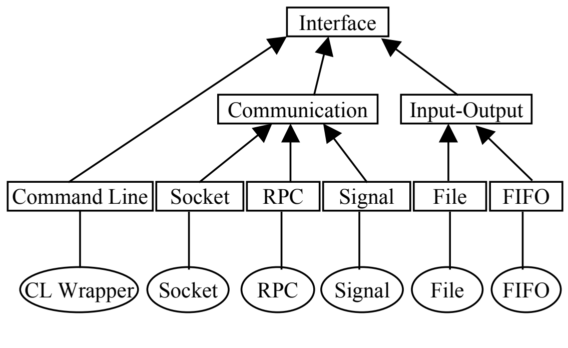


Figure 4. Wrapper Template Classes Hierarchy

4.4 Automatic wrapper generation steps

1. parsing step

The wrapper generator accepts an EIDL file as input, and then compiles it. At this step, the wrapper generator finds out the interface types and operations of legacy systems. The arguments of each operation of legacy systems are also processed. That is, the arguments (i.e. $0, $1, etc.) in LI parts are replaced by the corresponding parameters of the operation in IDL parts appropriately. For example, '*$0*' is replaced by '*name*' in the case of an example in Figure 3.

2. inheriting step

After the interface types of legacy systems are determined, the wrapper generator searches WTC to select appropriate classes. Each interface type inherits one concrete class from its template class. The arguments of each operation within concrete classes are substituted by the concrete parameters according to the relationships identified in step (1).

3. composing step

In CORBA, interfaces of server applications should map to the interfaces of IDL. In our case, however, there exists a situation that an operation name in IDL parts is not the same as the name of the corresponding operation in the wrapper classes. For example, see 'compile' and 'execute' in Figure 3. As a solution, we use an example servant generated from the IDL compiler (see Figure 6). It has the perfect structure of the server object except it is in skeleton form. That is, it has the same interfaces corresponding to IDL interfaces, but there are no implementation parts for operations. This step composes a complete wrapper object with an example servant and a concrete class derived in step (2). Here, the former provides the object structure and interfaces of operations, and the latter provides implementation codes for operations.

```
….
public  java.lang.String  compile (
        java.lang.String  name,
        java.lang.String  inFile
)
        ….
        File.save(name,  inFile);
        String  cmd  =  "cc " + name + ".c  -o " + name;
        CommandLine.execute(cmd);
        ….
}
```

Figure 5. Generated wrapper object

For example, a wrapper object in Figure 5 is generated from an EIDL in Figure 3 using the wrapper generator.

Figure 6 shows an automatic wrapper generator and surroundings. A server-side developer writes EIDL to describe the services provided to client and compiles it using the IDL compiler. At compile time, only IDL parts are compiled because LI parts are treated as comments. The products of IDL compiler are a server skeleton, a client stub and an example servant, which is a language specific example class for server applications. Again the EIDL is submitted to a wrapper generator. As stated before, a server-side developer's role is merely to write an EIDL file. He does not have to implement the interfacing techniques to legacy systems any longer.

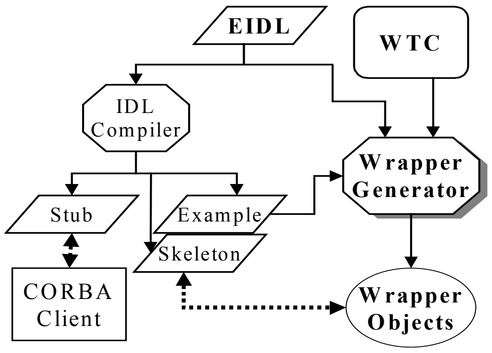


Figure 6. A Wrapper Generator and Surroundings

5. Application Example

This section describes a simple distributed C language IDE that simulates an integrated development environment usually provided in a stand-alone system. We assume the situation that Host A has no available C compiler and debugger but Host B has them, and a client-side developer is in Host A. Figure 7 shows the C IDE structure in CORBA based distributed environments. This system uses a wrapper generated by the automatic wrapper generator. C IDE accesses the wrapper in Host B via ORB and passes the messages to the wrapper. And then the wrapper interacts with C compiler and debugger and passes the results to C IDE via ORB inversely.

In this example, the C compiler and the debugger exist in the same host. Of course, they can be in the different hosts, respectively. Moreover, the client application developer does not have to know the locations where they are located. He can find them using the CORBA naming service.

Figure 8 shows a screen snapshot in Host A after compiling a C program in Host B. The compiled results are shown at the lower window.

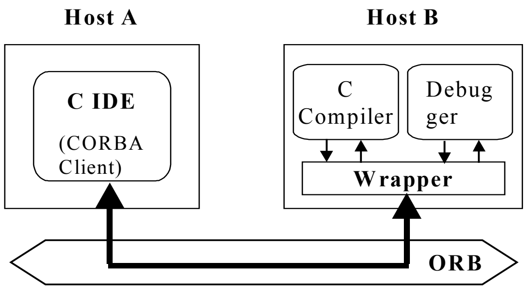
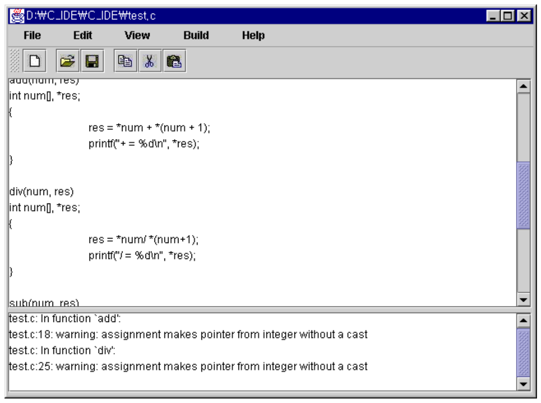


Figure 7. A simple distributed C IDE



```
add(num, res)
int num[], *res;
{
                res = *num + *(num + 1);
                printf("+ = %d\n", *res);
}

div(num, res)
int num[], *res;
{
                res = *num/ *(num+1);
                printf("/ = %d\n", *res);
}

sub(num, res)
```

```
test.c: In function `add':
test.c:18: warning: assignment makes pointer from integer without a cast
test.c: In function `div':
test.c:25: warning: assignment makes pointer from integer without a cast
```

Figure 8. A screen snapshot after compiling

6. Related works

There is some prior work that addresses issues for generating wrappers with semi-automatic generation or with meta-wrapper style. Vidal et al.[8] suggested wrappers and mediators to access data from heterogeneous databases or legacy servers. To access multiple sources in a dynamic environment, wrappers are rewritten by the task of capability based rewriting (CBR), depending on the capability of each source. However, the capabilities of the available source and the representations appear very diverse since the rapid growth of the Internet, and the emergence of representations such as XML, to facilitate the exchange of data on the Web, has dramatically increased the number of available sources. Every time we access new resources, we would need to rewrite wrappers. Vidal et al. introduced a meta-wrapper component into the architecture and provided a single interface to the mediator using the meta-wrapper component. Meta-wrapper component, by providing a single meta-wrapper interface, reduces the complexity of the CBR task of the mediator and makes multiple sources transparent. This research is confined to database domains and/or information retrieval domains on the Web. Another limitation is that this approach is only used restrictedly for applications that access documents or data on the Web. It may be possible to provide a single interface using the meta-wrapper component. That is, although the treated resources have a number of types, the methods that treat them are roughly similar.

Ashish et al.[1] suggested information mediators for obtaining information from multiple Web sources. Using this approach, wrappers are built around individual information sources to translate between the mediator query language and the individual sources. The authors note that it is impractical to construct wrappers for each Web source by hand for several reasons: very large information sources, frequently added new sources, and frequent changes to the format of sources. To cope with this problem, the authors provided a semi-automatic wrapper generation facility. Three steps are involved in generating a wrapper for a Web source: structuring the source, building a parser for the source pages, and adding communication capabilities. This approach, like the approach of Vidal et al., is applicable to only the Web sources.

There is work of Souder and Mancoridis [7] employs wrappers for securely integrating legacy systems into a distributed environment. Such wrappers provide their own layer of security between the security domains of the host and the distributed object system to protect the application against malicious users and the host from malicious applications.

7. Concluding Remarks

In this paper, we present the wrapping technique that enables various legacy systems to be reused on CORBA based distributed environments without any changes to them. To mitigate the burden of application developer who wants to use legacy systems on CORBA based distributed environments, we suggest an automatic wrapper generation method based on extensible wrapping template classes. The benefits of this research are as follows:

- Enhancement of reusability

By using previously developed programs through wrapper objects, we can reuse them at the level of executable codes.

- Reduction of software development cost

The previously developed programs have high reliability since they have been used and tested for a long time. Rather than redesign and redevelop programs with the same functionality, wrapping legacy systems should reduce both development and testing cost.

- Location transparency of server applications

CORBA wrapper objects for server applications have a location transparency feature via the naming service, one of the CORBA common services. Thus, although wrapper objects can be located any location on the network, the developer who uses wrapper objects is able to construct client applications without any consideration about the locations of server applications.

8. References

[1] N. Ashish and C. A. Knoblock, "Semi-automatic Wrapper Generation for Internet Information Sources", in Proc. of 2nd Int'l Conf. on Cooperative Information Systems, 1997, pp.160-169.

[2] M. Battaglia, G. Savoia and J. Favaro., "RENAISSANCE: A Method to Migrate from Legacy to Immortal Software Systems", in Proc. of CSMR'98, IEEE Computer Society, 1998, pp.197-200.

[3] N. Ganti and W. Brayman, *The Transition of Legacy Systems to a Distributed Architecture*, John Wiley & Sons, 1995.

[4] T. J. Mowbray and R. Zahavi, *The Essential CORBA: System Integration Using Distributed Objects*, John Wiley & Sons, 1995.

[5] R. Orfali and D. Harkey, *Client/Server Programming with JAVA and CORBA*, John Wiley & Sons, 1998.

[6] H. M. Sneed and R. Majnar, "A Case Study in Software Wrapping", in Proc. of ICSM'98, IEEE Computer Society, 1998, pp. 86-93.

[7] T. Souder and S. Mancoridis, "A Tool for Securely Integrating Legacy Systems into a Distributed Environment", in Proc. of Sixth Working Conf. on Reverse Engineering, 1999, pp.47-55.

[8] M. E. Vidal, L. Raschid, and J. R. Gruser, "A Meta-Wrapper for Scaling up to Multiple Autonomous Distributed Information Sources", in Proc. of 3rd Int'l Conf. on Cooperative Information Systems, 1998, pp.148-157.