International Journal of Computer Applications (0975 – 8887)
Volume 57– No.18, November 2012

Survey of Integrating Testing for Component-based System

M. K. Pawar
Assistant Professor,
UIT, RGPV, Bhopal

Ravindra Patel, PhD.
Associate Professor
UIT, RGPV, Bhopal

N. S. Chaudhari, PhD.
Professor, Deptt. Of CSE
IIT, Indore

ABSTRACT

Today's software larger in size, design complex and time consuming to implement them, for this we need a prominent solution to overcome these problems. Component-based software development (CBSD) has emerged as an Object Oriented (OO) Software Engineering approach that forced rapid software development. Using CBSE approach we can eliminate these problems largely. To build the application using CBSE approach we can develop the software with lowest price, reduced in size and we can reduce the time also. The component-based application may be implemented in house or by different vendors, integrate them in a different environment is still challenging. Because the unit level testing of each component, which can be good but when we Integrate them with different framework is a considerable problem. The nature of the component is Heterogeneous, so the integration is a bit complicated. Through this paper we are highlighting features and drawback of each methods of various Integration testing. And we are also giving a potential method which would be suitable for Integration testing.

General Terms

Component integration testing.

Keywords

Component-based system, Component integration testing, interface testing.

1. INTRODUCTION

Traditional software systems become larger in size, more complex and uneasily controlled and high maintenance cost, resulting in high development cost, slower productivity, compromised in software quality and high risk to move to new technology [1]. Consequently, there is an increasing demand of searching for a new, efficient, and cost-effective and time to market software development standard. One of the most potential solutions today is the component-based software development approach. The idea that software should be componentized, [11] built from existing *components* by gluing prefabricated components together much like in the field of electronics or Mechanical. Concerns and objectives are similar in many other engineering disciplines such as in Civil engineering: house prefabrication, Chemical engineering: proteins, Electronic engineering: circuit and Industrial engineering: cars, in above engineering domain especially in Industrial and civil engineering successfully develop components because of Standards and Rules. These engineering branches are quite old so the rules and standards are well defined. The rapid changes in the software and implementation point view it is quite complex so the concept of component in the software will take time in the Rules and Standard. Since CBS development process is fast, less time required to assemble and easy maintenance. Quality wise it is also good, because testing of each and every component is performed at unit level. But when we integrate the component, then the unexpected result can occur at different levels. Therefore, substantial testing and an appropriate method require at the integration level.

When we talk to Component integration that our first focus is the interface, because the interface connects two or more components and that the application is ready to use. Interfaces are the interaction points of components, through which a client component can use the services declared in the interface as provided and required interface. Each interface is identified by an interface name and a unique identification. Every interface can include numerous operations, where each operation performs one particular task. Software components can be incorporated in a system as units as shown in Figure: 1, A component based application on a hotel reservation system where each component is tested well and integrate them in a new environment to perform the task of an application. The component to be connected to one another, not only realize it when we actually do in a new environment indeed have their effort to Integrate. Component Integration is one of the main phase which is done by the user If we are not connected component properly interfacing is the problem or connect the properly and even expected results [2] are not received then fully testing is required on the level of integration.

A simple example of two components expressed in UML 2.0 as shown in Figure: 2 the other component, responsible for facilitating the customer's order, *requires* the order processing component to charge the customer's credit/debit card, the functionality, which the latter *provides*.

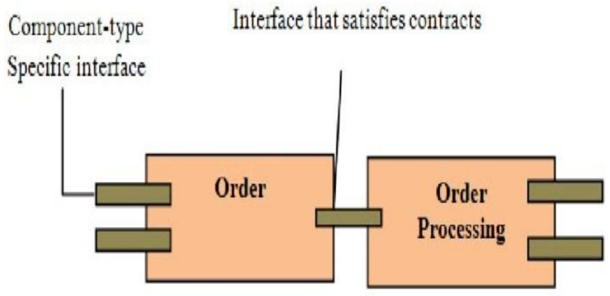

Figure: 2 Representation of Components in UML 2.0

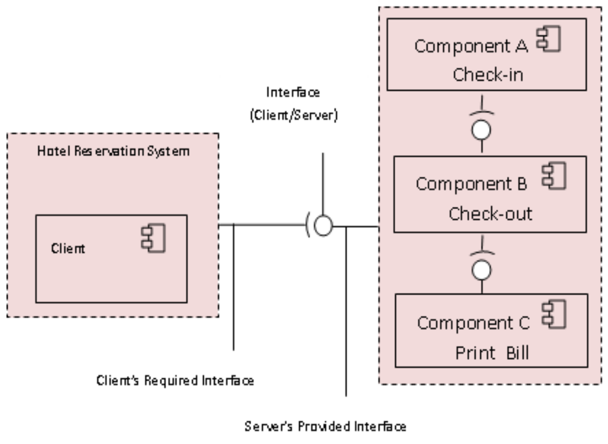

Fig: 1 Hotel Reservation Components with provided & Required Interface

2. COMPONENT BUILDING AND UNIT TESTING

Before we may know the various issues and methods of Component integrate, it is necessary to know the process of implementing the component and how to perform the unit level test. Implementation of the component is still challenging task because of the lack of standards and services to make interoperable various languages and platform. Here we summarize the building process of component and their unit testing:

2.1 Component implementation process

To implement the Component of any language and any platform we have to required particular language mapping IDL compilers. There are various IDL compiler tools were developed by different vendors and they successfully achieve the adaptive environment for most of the languages [9]. IDL compilers that support the CORBA [9] standard such as: IIOP.NET, interoperation between .NET, and CORBA or J2EE, Jacob, written in Java IDL-to-Java Compiler, R2CORBA, a CORBA implementation of the Ruby Programming Language, VBOrb, CORBA Visual Basic clients and servers, MICO, IDL to C++ mapping, ACE ORB (TAO), IDL to C++ mapping, omniORB, ORB with C++ and Python bindings, ORBit, C and Perl bindings, *idlj - The IDL-to-Java Compiler* etc.

Here we summarize that how to implement the component using the OmniORB IDL compiler for IDL to C++ language mapping. The steps are necessary to build the component and client - server programs using the *OmniORB* [10] compiler:

- Compile the IDL (*for IDL to C++ Mapping*)
- *IDL Compiler* generates the files (i.e. Stub, Skeleton and other files for CORBA and Network support).
- Implementation of the method, Server and Client program.
- Compile the implementation method, server, client and other program files using the C++ compiler
- Link the stub file to the client, skeleton file to the server and other files.
- Launch the naming service, server and client program using an appropriate port number.

2.2 Unit testing

A ***unit testing*** of Component is the validation process that is carried out to find the errors. Unit testing is performed according the specification of input and output parameters. Developers of an application programmer know that testing is necessary steps to build reliable application. In a component based application development process the component may be collection of in house or third party. So it is necessary to perform the unit test on different components. There are various unit testing frameworks being available to perform unit testing (for example Junit, Xunit etc.). The testing procedure uses verification and validation methods [15] which may vary from one framework to another framework but will all fundamentally test if a particular test unit is fit for use . Unit tests facilitate refactoring – When changes are occurring at the code level within a unit, tests are readily obtainable to check if the changes produce errors. Units of component can be checked at all times to make sure that functionality is upheld. Unit tests allow collective ownership because the implementation code is not available due to its black box nature, changes may be made by all relevant vendors. This is because unit tests protect the verification and validity of the code so that after changes are made the unit must be tested to certify that all functionality still remains as their specification.

3. ISSUES AND CHALLENGES IN COMPONENT-INTEGRATION TESTING

Over the years different researcher and many published papers are engaged in this effort how the software component should be implemented and how a reliable component-based application should designed. So far only a few research papers who drew attention to how the software component integration testing should be performed in an effective manner.

The software components have been unit tested, many of these components have been deployed independently[15], however, Substantial Integrating testing is required, Which also be according to the quality. As we analyze, this task is even harder due to various reasons. Study and installation of various IDL compilers are challenging, which require a lot of configuration. The language, in which you want to develop the component and testing of that component. Therefore, appropriate efforts to encourage the reliability of the components are necessary. But as time moves on, more and more software components will be available from different vendors. On the other hand when the "perfectly matched" components are integrated, The following issues are still encountered during testing of component based software, Which then requires further testing effort.

3.1 Heterogeneity of Component

Since the nature of the component may be heterogeneous, it can be implemented using various programming language and runs on different platforms. Operating the component-based system in various platforms is one of the challenging issues. So it is necessary that an implementation of a component in any language should support various operating systems, database of various corporations and middleware through which the communication is performed [12]. For instance, the failure of the European Ariane 5 launch vehicle was due to a reused module in Ariane 4 failing to convert a 64-bit floating-point value to a 16-bit signed integer. There are various well developed components are exist and they are working well to build an application but the limitations of the component are that their domain specific nature for example COM, DCOM of Microsofts, EJB of Java etc. When we cross the boundaries of their domain they are incompatible to operate in different environments. For example Microsoft has developed .NET framework and incorporate a tool of an intermediate language: *Microsoft Intermediate Language (*MSIL). MISL supports the various packages of its own development and limited functionality of JAVA and COBOL to make interoparabe. It easily compiles and generate the intermediate files that support the Visual C++, Visual Basic, Visual J++, Java, and COBOL. With the .NET framework, components that are implemented using their own high level languages, JAVA and COBOL under different platforms can be easily combined without worrying about the interoperable issues.

3.2 Component Communication

Component communication is another issue that arise during component integration. Many people assume that the integration of software components has been just a process to connect the component in plug-in way. For many simple cases, it is true that software components can be viewed as an interface point. In other words, a software component application in which client sends the request to the server object and server object will reply back if and only if client request is per specification. For most complex software components, middleware communication, naming services and data models need to be taken into account. Middleware communication specifies how the different components interact. Their interactions can go through an RMI method [1], or their communications can go ORB or other middleware. Different data structure defines the contents and format of the interactions in the middleware communication. Usually, software components are developed by different vendors, who may make different assumptions of how components interact and what details are involved in their interactions. Furthermore, a component may expose multiple interfaces, which may have varied constraints and different types of relationships with each other. For instance, a Vending Machine component may represent Select Item, Coin Checker, Compute Change, and Dispenser interfaces. Among these interfaces, instead of allowing arbitrary invocation sequences, a Select Item interface has to be invoked at the very beginning of each transaction, and Dispenser has to be performed at the end of each transaction. Different invocation sequences may generate different results.

3.3 Distributed System Issues

As component-based systems are always built under a distributed operating environment, which will then come into all the issues of distributed systems, such as transaction controlling and deadlocks. These distributions related [8] issues can only be detected during the integration phase. Moreover, component-based may even introduce versioning issues, which is caused by the coexistence of two different versions of a component in the system.

4. INTEGRATION TESTING TECHNIQUES

There is growing demand of Component based system many researchers are beginning to focus the testing of Component based system at unit and system level. There are various published papers have discussed the improvement of testing in component based system. Many researchers have given the different component integration techniques; we are highlighting some major integration testing techniques:

4.1 UML based test model

The Unified Modeling Language (UML) is a general-purpose modeling language that is used to identify, visualize, construct and document the artifacts of a software system.UML Statechart can be used to describe the static and dynamic behavior of a component or object over time by modeling its lifetime. UML has many useful tools [4], such as interaction diagrams, statechart diagrams, and collaboration diagram, component diagrams, differentiate the activities of a component in various phases, and thus can be used in testing component-based systems. All the elements of UML are analyzed and apply the various testing criteria to test the component based system. In a Statechart diagram main elements are states, transitions, events, and actions. States and transitions define all possible states and changes of state an object during its lifetime. State changes occur as responses to events received from the object's interfaces. This test model uses the UML Sequence and Collaboration Diagrams to take out the faults existing between the component interfaces communicating with each other in the system. It incorporates the UML based development process to the test process. This technique of testing has not yet implemented. But this can be automatic testing technique.

4.2 Component Interaction Graph (CIG)

Component Interaction Graph (CIG), [3] which shows the interactions and various dependencies among components. CIG detects the failures of interface encountered during integration of various components to build an application. The approach utilizes both the static and dynamic information to design test cases and determining test sufficiency. The example of component interaction graph shown in figure 3 to build an application. CFG is formed by the sets of provided and required interfaces, where each vertex represents a method of an interface. Edges are represented from the vertices corresponding to the required interfaces to the vertices of providing interfaces for component dependencies, and from the provided to the required interfaces for component dependency. The suggested approach can be applied on all types of components and it is based on black-box testing.

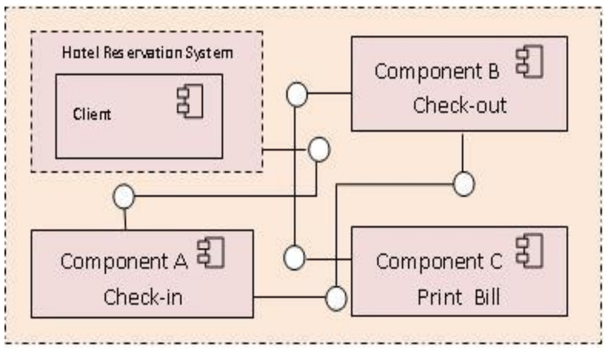

Fig:3 Component interaction graph with provided & Required Interface in Hotel Reservation

4.3 Certification of components

The certification of a component is a proper demonstration that the component complies with its specified requirements and is acceptable for use as specified. A good certification methodology [6] gives the component reliability that it deserves. The approach suggests a method for the certification of component. It covers the certification of the whole application as well as a component unit. The certification applies to the safety critical parts of the component that have been identified during the analyses. For certification of component we require black-box testing as component level, System-level fault booster technique, checking lack of operational level and implement measures to protect the building step.

4.4 Component Interaction Testing (CIT)

This approach [5] is based on the assumptions that how the components interact with one another. These suppositions are restricted as formal test requirements that specify the choice of test cases. The testing technique presently focuses on the control-flow interactions of events; other types of interaction are not encountered. So we cannot imagine it completely reliable testing technique. Scalability is an another issue with this technique. In theory the model can enough scale from unit testing to system testing but as the size of an application model increases, it becomes more difficult.

4.5 The Component Metadata way

In this approach [7] component metadata information is used to analyze and test components. When integrating a well-developed component to build an application, we may need to perform a set of tasks including, the requirement of third-party information about the certification component, analyses and testing of the system, and evaluation of some quality of the resulting based on pre and post condition. The metadata is based on different kinds of information depending upon the explicit context and needs. There is a unique arrangement and a unique label for each kind of metadata provided [7]. The source code for the component is generally not available, and so we have only a formal specification of the components. The component developer embeds this summary information in the software component. The metadata illustrates both the static and the dynamic aspects of the component. Metadata Increases the precision of the program analyses and change according to the particular functionality required by the component user. The idea behind the concept of metadata is to define an infrastructure that lets the component developer add the different types of data that are useful in a given context. Obviously, metadata [15] can also be produced for in-house-developed components, so that all the components that are used to build an application can be handled in a uniform way. This notion of providing metadata with software components is highly related to what mechanical engineers do with hardware components: just as building a car, so a software component needs to provide some information about itself to be used in different context [13]. The more metadata that is available from or about a component, the fewer will be the restrictions on tasks that can be performed by the component user, such as applicable program analysis techniques, model checking etc. In this sense, the availability of metadata for a component can be perceived as reliable by an application developer who is selecting the components to deploy in his/her system. So far, it has applied and tested with small applications.

5. PROPOSED METHODOLOGY

This approach is based a combination of Sequence Diagrams and UML State chart. Furthermore the semantic information is expressed in OCL. The process is built using standard notations, which are used widely. This overcomes the issue of learning new notations and languages to identify with the approach suggested. This approach assumes that all the components in the integration level have already been tested individually and thus considers them as black-boxes. We carry out the component integration testing in the following steps:

- Building of the UML test model based on the one flow of events. First the sequence diagram of normal flow of events is extracted. This information is then used to model component interaction states and transitions using UML State chart.
- The UML State chart is used to generate test sequences for normal and exceptional behaviour.
- Test Cases are selected by application of the test case selection criteria on the UML test model.
- The testing technique has potential for automation where test sequences can be generated automatically from the State chart.
- We intend to evaluate the effectiveness of the proposed approach on a set of components.

6. CONCLUSION

In this survey paper we revealed the various integrations testing techniques of component based systems; this paper is useful for early stage researcher.. We propose component integration testing based on UML and OCL which overcomes some of the limitations of existing model based integration testing. Most of the component integration testing can be automated. UML is a semi-final language for indicating, visualizing, building, and documenting the artifacts of software systems and at times is not capable to represent or model a perspective satisfactorily. We demonstrate and encourage the use of OCL to overcome this limitation of UML. Propose approach does not assume the availability of source code and also will be helpful on COTS. And therefore will also be useful for testing component assembly comprising of COTS components

7. REFERENCES

[1] Pour, G., “Software Component Technologies: JavaBeans and ActiveX,” Proceedings of Technology of Object-Oriented Languages and systems, 1999, pp. 398 – 398. Weyuker, E.J., Testing component-based software: A cautionary tale, IEEE Software, 15(5):54–59, September/October 1998.

[2] Wu, Y. Pan D., Chen, M. H., Testing Component Based Software, submitted to International Conference on Software Engineering, Toronto, 2001

[3] Ye Wu, Mei-Hwa Chen and Jeff Offutt “UML-based Integration Testing for Component-based Software”, The 2nd International Conference on COTS-Based Software Systems (ICCBSS), pages 251-260, Ottawa, Canada, February 2003

[4] Choi, Y. H., B., Jeon, J. O., A UML Based Test Model for Component Integration Test, Workshop on Software Architecture and component (WSAC), Japan, 1999.

[5] Liu, W. and Dasiewicz, P. Formal Test Requirements for Component Interactions, IEEE Canadian Conference on Electrical and Computer Engineering, 1999.

[6] Vaos, J. M., Certifying Off-the Shelf-Components, IEEE Computer, June 1998.

[7] Orso, A., Harrold, M. J., Rosenblum, D., Component Metadata for Software Engineering Tasks, In Proc. 2nd International Workshop on Engineering Distributed Objects, Davis, CA, November 2000.

[8] Harrold, M. J., Liang, D. Sinha S., An Approach To Analyzing and Testing Component Based Software, Proceedings of the First International ICSE Workshop on Testing Distributed Component-Based Systems, Los Angeles, CA, May 1999.

[9] Object Management Group; Object Management Architecture Guide, OMG Document Number 92.11.1, Revision 2.0, 1992

[10] The omniORB version 4.1, User’s Guide Duncan Grisby, Apasphere Ltd., Sai-Lai Lo, David Riddoch, AT&T Laboratories Cambridge, July 2009.

[11] Cheesman, J., and Daniels, J. UML components: A simple process for specifying component-based software. Addison-Wesley, 2001.

[12] Heineman, G. and Councill, W. Component-based software engineering: Putting the pieces together. Addison-Wesley, 2001.

[13] Orso, A., Harrold, M. J., Rosenblum, D., Rothermel, G., Soffa, M. L., and Doo, H. Using Component Metadata to support the regression testing of component-based software, In Proceedings of the International Conference on Software Maintenance (ICSM2001), pp 716-725, November, Florence, Italy, 2001.

[14] Liang, D., and Harrold, M. J. Reuse-driven inter-procedural slicing in the presence of pointers and recursion. In Proceedings IEEE International Conference on Software Maintenance, pp. 421-430, 1999.

[15] M.J Rahman, F Jabeen,, A. Bertolino “Testing Software Component for Integration: survey of issues and techniques”, 2006, DOI:10.1002/stvr.357, wiley InterScience 17:95-133.