INTEROPERABILITY IN COMPONENT BASED SOFTWARE ENGINEERING

MS. SHIWANI [1], MR. SOMESH [2]

[1] Assistant Professor in Computer Science, Sanatan Dharma College, Ambala Cantt

[2] Assistant Professor in Computer Science, Sanatan Dharma College, Ambala Cantt

ABSTRACT

In component-based software system development, it's necessary to confirm ability between elements supported their unambiguous linguistics descriptions, so as to get a viable system. A body of recent work has explored once a code fails, a confusing and sophisticated liability downside ensues for all parties that have contributed computer code practicality (whether COTS or custom) to the system. Potential contributors to the system failure include: (1) defective software elements, (2) issues with interfaces between elements, (3) issues with assumptions (contractual requirements) between elements, and (4) hidden interfaces and non-functional part behaviors that can't be detected at the part level. in this paper, our goal is to specialize in the ability issues created by defective COTS software system elements, this paper deals with the fundamental ideas associated with part ability, with special stress within the grammar, protocol and operational specifications of elements. the most goal is to indicate the prevailing issues, survey the present solutions and to indicate however they address those issues, and to draw attention towards a number of the still open problems and challenges during this fascinating space.

Introduction

COMPONENT-BASED computer Software development is gaining recognition because the key technology for the development of top quality, evolvable, giant computer code systems in timely and reasonable manner. ability is one amongst the essential problems, since it permits the composition of reusable heterogeneous elements developed by completely different folks, at completely different times, and probably with completely different uses in mind. presently most object and part platforms, like Common Request Broker design (CORBA),

Distributed part Object Model (DCOM) or Enterprise Java Beans (EJB) already provides the fundamental infrastructure for part ability at the lower levels, i.e., they delineated most of the “plumbing” problems. However, ability goes so much on the far side that; it additionally involves activity compatibility, protocol compliance and agreements on the business rules.

Component interoperability

The main goal of part primarily based software system is to cut back development prices and efforts, whereas rising the flexibleness, responsibleness, and reusability of the ultimate application because of the (re)use of computer code elements already tested and valid. This approach moves organizations from application development to application assembly.

Therefore, one amongst the key problems with building applications from reusable elements ability. ability is outlined because the ability of 2 or additional entities to speak and work in spite of variations within the implementation language, the

execution environment or the model abstraction.

In component-based computer code development, it's necessary to confirm ability between elements supported their unambiguous linguistics descriptions, so as to get a viable system. it's the power of 2 or additional elements to work despite their variations in practical side like interface signature or linguistics variations [1, 2, 4]. Non-functional ability like security and performance ability is recognized as necessary [5, 6]. historically, 2 main levels of ability are distinguished:

Signature level (names and signatures of operations), and also the linguistics level (the “meaning” of operations). within the initial place, grammar ability is currently well outlined and understood, and middleware architects and vendors are

ISBN: 978-81-954645-5-5

attempting to determine completely different interoperation standards at this level. they need born to the prevailing business part models and platforms like CORBA, EJB, or DCOM. However, all parties square measure beginning to acknowledge that this kind of ability isn't spare for making certain the proper development of huge part primarily based applications in open systems. On the opposite hand, the prevailing proposals at the linguistics level give part interfaces with info regarding their behavior. Behavior indicates however computer code responds to external events. The formal specification of the system is expressed in term of model-based specification. Model-based techniques model the system using mathematical constructs like sets and functions. The operations in an exceedingly model-based specification square measure outlined victimisation pre- and post-conditions on the system state. though {much additional|far more|rather more|way more} powerful than more signature descriptions, addressing the activity linguistics of elements introduces serious difficulties once applied to giant applications. In fact, most of the formal notations are around for many years

Components class and Interfaces:

An interface, a key part for middleware technology, may be a assortment of attainable functions wont to specify through its operations the services of a computer code unit. looking on the chosen middleware technology, interfaces square measure developed with a selected definition artificial language like CORBA, Microsoft IDL for COM, Java interface for RMI and WSDL for net services. a category is associate object orientating thought. It describes a collection of shared objects and belongs to the implementation step. associate object is associate instance of a category. associate object satisfies associate interface if it is such as because the target object in every potential request delineated by the interface. It belongs to the implementation step. but this object is distinct from the opposite usual objects since it collects the remote calls. the event of distributed computer code doesn't imply the selection of associate actual object-oriented language (commonly C++, VB and Java) since middleware's like COM and CORBA introduce the notion of pseudo objects. The part may be a computer code unit that encapsulates the implementation of business method logic. Sessions R. stresses the distinction between part and object technology, the previous being a packaging and distribution technology, that specialize in what a part will do, whereas the latter is associate implementation technology, that specialize in however a part works [3]. Objects and elements square measure computer code entities; objects square measure usually fine-grained units, act within the same computing method whereas elements square measure rather universal grained units, and square measure obtainable outside their own method with relation to interface definitions. they're issued from completely different computer code style. The distinction between these 2 states is clearly known within the CAPE-OPEN customary from CAPE-OPEN-Laboratory Network. A CAPEOPEN accommodating part may be a piece of computer code that features the provider proprietary codes—objects or no that acknowledge or/and use CAPE-OPEN interfaces. The communication between CAPE-OPEN part instances is outlined unambiguously by the CAPE- OPEN interfaces [7]. during this case, the middleware technologies square measure CORBA and COM.

Middleware interoperability protocol

Component based applications consist of several pieces of software, which are executed independently and reside on the same host or on remote hosts over a network such as intra and Internet. There is a need for *application integration* and so for *component communication* through well-defined interfaces. Middleware is a *set of software that allows and organizes communication and information exchange* between client component and server component. **Fig1** shows this technology as a universal communication bus, the “glue” of any IS, for integrating the different enterprise applications. It relies on a basic *client-server communication model* adding a key element; the *interface* defined in terms of Interface Definition Language (IDL). A middleware solution provides mechanisms for interface definition and communication as

well as additional services easing the use and implementation of component based software. The middleware interoperability protocol defines how the components communicate which each other. It defines marshalling process, how the data structures (integer, real, string.) can be translated into network messages. There are three kinds of middleware technology: *Message Oriented Middleware* (MOM), *Remote Procedure Call* (RPC) such as SOAP and *Object-Oriented* (OO). The interface design of OO middleware follows the object-oriented paradigm. At present the OO middleware solutions are (DCOM and .Net Remoting from Microsoft, CORBA from OMG and RMI from Java/SUN. COM, CORBA and SOAP. All communication between software components is handled by middleware technology. Let us see the different alternatives for inter-system communication technologies e.g. how our system could process requests between software components. As a first approach, we distinguish two ways for exchanging information: the *data model and Application Programming Interface* (API. With the data model, we can use point-to-point software

integration and file format/database integration. However, this static asynchronous communication is not appropriate to systems that use intensive integrated calculations. Indeed the performance penalty of managing physical files can be high and can prevent this approach being effective for exchanging information. Therefore, interoperability can be achieved by API for carrying out inter-communication processes. We can identify two kinds of API technologies that are commonly used in any IS project, *tightly coupled* and *loosely coupled* middleware **[8]**

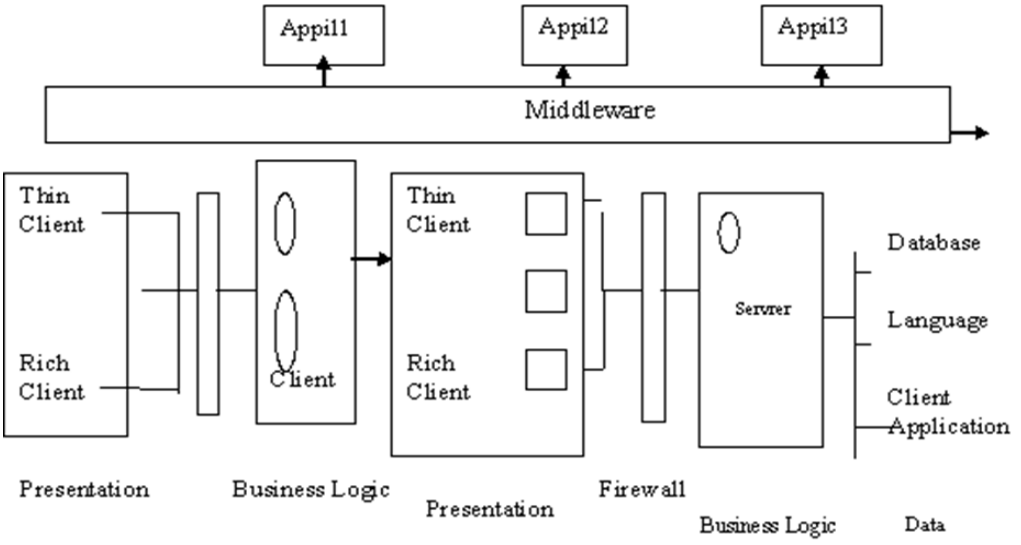

Fig 1. Middleware and client/server model

Tightly coupled middleware **technology**:

This technology requests that software components have a close relationship. Typically, this means that the components are built on identical middleware. OO middleware are typical examples. Here the components are closely linked by *implementation dependence.* For example, a COM component can interoperate with a COM component on Windows. However, non-trivial solutions exist to break this tightly coupling such as bridging technologies.

Loosely coupled middleware **technology:**

Software components do not need to share the common technology platform and they can be deployed easily over the web. The components are loosely coupled by *implementation independence.* The web is based on this kind of protocol with HTML/HTTP. In this field, the emerging industry standard for loosely coupled inter-communication process is SOAP.

Marshalling:

An important notion for the API model is *marshalling* as remarked previously. Because distributed systems are heterogeneous (*i.e.* non-uniform hardware, operating systems, etc.) the exchanges of data between different components must adhere to the same conventions with respect to the internal encoding of numeric data, to the encoding of data over a network (Unicode strings). Marshalling is the mechanism that ensures that the parameters of a method call are properly passed between the caller of the method and the callee (*i.e.* the code that implements the method).

OPEN PROBLEMS IN INTEROPERABILITY IN COMPONENT BASED SYSTEMS

Even in the simple case where the objective is to integrate new systems that are designed from the beginning to interoperate, the challenge is substantial. In particular, large software and/or hardware systems that are intended to interoperate are built from specifications, and these specifications are inherently incomplete. This leads to syntactic and semantic mismatches that are usually discovered during system integration and testing (but may not be discovered until long after the system is in production use).

Interoperability problems are of two kinds: interface mismatches and protocol mismatches. An "interface" describes a component's characteristics, e.g., its functionality, structure and performance. A "protocol" describes the connections the components use for communication, e.g., reliability (no lost messages, no duplicates), directionality, rules that govern

ISBN: 978-81-954645-5-5

the temporal ordering of messages and performance. If system A uses a different protocol than system B, then we have a protocol mismatch. If system A makes incorrect assumptions about the meaning or the format (grammar) of data that it receives from system B, then we have an interface mismatch.

Interfaces and protocols are contracts between interoperating systems. Their specifications define the requirements for communications between heterogeneous systems. They are thus the keys to interoperability. There is a problem when the contracts are violated, incomplete, inappropriate or not in place. More specifically, an interoperability problem arises between two components if their interfaces do not match, if their protocols do not match or if their interfaces and protocols do not match. If the protocols or interfaces are not clearly specified, it is difficult to determine whether an interoperability failure is truly a mismatch or whether one of the systems hasn't fully or correctly implemented the specified protocol or interface.

Interfaces and protocols may be open or closed, formal or informal. For the purposes of this report section, open roughly corresponds to "published" or otherwise openly available to anyone who wishes to use the protocol and interface. If open, the protocol and interface may be standard (i.e., developed and ratified by a technical committee that includes representation from many different suppliers and users) or not.

Some other problems arising in component interoperability are:

1. COTS software is usually delivered as black box components with limited specification making it difficult to predict how the components behave under

different conditions.

2. There is a general lack of methods for mapping user requirements to component based architecture. Components are packaged and delivered in many different forms (example: function libraries, off-theshelf applications and frameworks).

4. Component framework offer varying features (example: component granularity, tailorability, platform support, distributed system support, interoperability).

5. Most component integration processes suffer from inflexibility by a lack of component evaluation schemes. This problem is often compounded by a lack of interoperability standards between component frameworks and adequate vendor support.

6. Most COTS software is generally not tailor able or “plug and play”. Significant effort may be required to build wrappers and the “glue” between components in order to evolve the applications or tailor components to new situations. As the system evolves these wrappers must be maintained.

Mechanisms for Resolving Mismatches

Three possible approaches to resolving or mitigating interface and protocol mismatch problems for legacy (and other) systems are:

- *Pair-wise approach:* For each pair of non-interoperable components, A and B, we need a pair of translation functions, one from A to B and one from B to A. Full connectivity among N components can require N(N-1) translation functions. The Information Access section calls this approach "point-to-point connectivity."
- *Common language approach:* For each component, A, we need a pair of translation functions, one from A to S and one from S to A, where S is a common language (i.e., protocol and interface) with which every component can communicate. S can be viewed as a single standard, e.g., RS232 or Ethernet, to which all players subscribe. Full connectivity among N components requires at most 2N translation functions. The Information Access section calls this approach "federated architectures" where the "facilitator" is the common language. This approach is also taken in the "hourglass" model of the Network Components and Protocols section; that is, the Internet Protocol (IP) is the common language used in the Internet.
- *Broker approach:* This is a hybrid approach. There are third parties ("brokers"), each of which understands some subset of the many protocols used by the components. If A wants to communicate with B, a broker acts as an intermediary. The broker figures out how to translate A's language so B can understand it and vice versa. Implementers of A and B do not have to provide translation functions. Instead, brokers supply them. One can view the union of the brokers as a set of standards. This approach is related to the Information Access section's

"mediator network." If this approach can be realized efficiently and without restricting functions, one advantage it offers for the NII is that it allows a set of acceptable standards rather than requiring that a single standard be agreed upon and adopted by government, industry, etc.

Conclusion

Interoperability is one of the major challenges, particularly within component based software development environments, an approach in which prefabricated reusable software components from independent sources are assembled together to build applications. There are many aspects related to component interoperability, including syntactic agreements on method names. In order for CBSE and CBD to flourish, technologies must exist that allow for the successful predictability as to how interoperable different software components are. Without predictability, interoperability cannot be known *a priori* until after a system is built. And it may be too late in the life cycle to financially recover if it is discovered that one or more of the components are not

Compatible. This paper has dealt in detail, the Interoperability issues pertaining to Component based software development.

References:

[1] P.Wegner. Interoperability. *ACM Computing Surveys*, 28(1):285{287, 1996.

[2] A. Vallecillo, J. Hernandez, and J. Troya. Component interoperability. Technical Report ITI-2000-37, University of Malaga, 2000.

[3] Sessions. R. Objects and Components, ObjectWatch newsletter number 28, June 2000

[4] S. Heiler. Semantic interoperability. *ACM Computing Surveys*, 27(2), 1995.

[5] P. Brereton and D. Budgen. Component-based sys- tems: A classification of issues. *IEEE Computer*, pages 54{62, Nov. 2000.

[6] M. Grechanik, D. Perry, and D. Batory. A security mechanism for component-based systems. In *Proc. 5th Intl. Conf. Commercial-of--the-Shelf COTS-Based Software Systems (ICCBSS06)*, 2006.

.[7] Beland, J. P. and Pos, M. Open Software Architecture Beland, J. P. and Pos, M. Open Software Architecture.

[8] Brown A. W., Component –Based Software Engineering: selected papers from the Software Engineering Institute, Wesley – IEEE Computer Society Press.