World Academy of Science, Engineering and Technology
International Journal of Computer and Systems Engineering
Vol:2, No:10, 2008

Interoperability in Component Based Software Development

M. Madiajagan, and B. Vijayakumar

***Abstract*—**The ability of information systems to operate in conjunction with each other encompassing communication protocols, hardware, software, application, and data compatibility layers. There has been considerable work in industry on the development of component interoperability models, such as CORBA, (D)COM and JavaBeans. These models are intended to reduce the complexity of software development and to facilitate reuse of off-the-shelf components. The focus of these models is syntactic interface specification, component packaging, inter-component communications, and bindings to a runtime environment. What these models lack is a consideration of architectural concerns – specifying systems of communicating components, explicitly representing loci of component interaction, and exploiting architectural styles that provide well-understood global design solutions. The development of complex business applications is now focused on an assembly of components available on a local area network or on the net. These components must be localized and identified in terms of available services and communication protocol before any request. The first part of the article introduces the base concepts of components and middleware while the following sections describe the different up-to-date models of communication and interaction and the last section shows how different models can communicate among themselves.

***Keywords*—**Interoperability, component packaging, communication technology, heterogeneous platform, component interface, middleware.

I. Introduction

COMPONENT-BASED software development is gaining recognition as the key technology for the construction of high quality, evolvable, large software systems in timely and affordable manner. In this new setting, interoperability is one of the essential issues, since it enables the composition of reusable heterogeneous components developed by different people, at different times, and possibly with different uses in mind. Currently most object and component platforms, such as Common Request Broker Architecture (CORBA), Distributed Component Object Model (DCOM), or Enterprise Java Beans (EJB) already provide the basic infrastructure for component interoperability at the lower levels, i.e., they sort out most of the "plumbing" issues. However, interoperability goes far beyond that; it also involves behavioral compatibility, protocol compliance and agreements on the business rules. Information systems largely involve networking these days. The development of complex business applications is now focused on an assembly of components available on a local area network or on the net. These components must be localized and identified in terms of available services and communication protocol before any request. This paper presents the most common technologies that allow heterogeneous and distributed software systems to collaborate. The first part of the paper introduces the base concepts of components and middleware while the following sections describe the different up-to-date models of communication and interaction and their use in industrial applications. The last section shows how different models can communicate among themselves. This paper deals with the basic concepts related to component interoperability, with special emphasis in the syntactic, protocol and operational specifications of components. The main goal is to point out the existing problems, survey the current solutions and to point out how they address those problems, and to draw attention towards some of the still open issues and challenges in this interesting research area.

M. Madiajagan is with the BITS, Pilani – Dubai Campus, Knowledge Village, Dubai, P.O.Box: 500022, United Arab Emirates (phone: 00971-04-3664575, fax: 00971-04-3664580; e-mail: jagan@bitsdubai.com).

B. Vijayakumar is with the BITS, Pilani – Dubai Campus, Knowledge Village, Dubai, P.O.Box: 500022, United Arab Emirates. Tel:00971-04-3664575, fax:00971-04-3664580, vijay@bitsdubai.com).

II. Related Work

Component-Based Software Development is an emergent discipline that is generating tremendous interest due to the development of plug-and-play reusable software, which has led to the concept of '*commercial off-the-shelf*' (COTS) components [1]. In lieu of coding components, many CBS developers assume they can just "plug in" COTS products. They assume using COTS products will shorten their programming and testing effort, with little other lifecycle process effect.

COTS product has the following features:

- sold, leased, or licensed to the general public;
- offered by a vendor trying to obtain profit from it;
- supported and evolved by the vendor, who retains the intellectual property rights;
- available in multiple, identical copies; and
- used without source code modification.

New process drivers flow both from this COTS product definition and from the consequences of assembling systems from COTS products. These new process drivers are:

- CBS development is an act of composition.
- The realities of the COTS marketplace shape CBS development.
- CBS development occurs through simultaneous definition and trade-off of the COTS marketplace, system architecture, and system requirements.

COTS-based system development involves composition and reconciliation, whereas custom system development is an act of creation. Custom development starts with the system requirements and creates a system that meets them; the engineers are producers. However, COTS-based system development starts with a general set of requirements and then explores the marketplace's offerings to see how closely they match the needs; the engineers are consumers, who then integrate the products they buy into a system that meets the need. The nature, timing, and order of activities performed and the processes used differ accordingly. The approach to system development for COTS-based systems requires a fundamental change, as shown in Fig. 1.

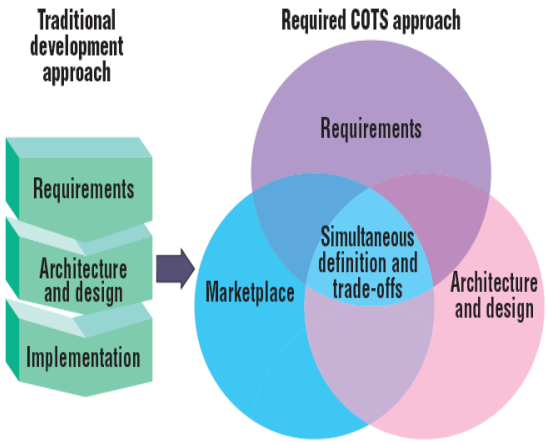

Fig. 1 COTS-based Systems

Enterprise software systems are becoming more and more complex. Applications have changed from simple stand-alone programs in a homogeneous environment to *highly integrated and distributed systems in heterogeneous environments*. Over the last ten years, the need for *web-enabled systems* has led to additional requirements. In 1990's there was a need for connecting the whole software system of an enterprise. Now the web environment results in the requirement to *interconnect the software systems of enterprises* to take advantage of the great business opportunities from the Net. The resulting Information software System (IS) plays a key role in enterprise; it is based on three fundamental parts: a set of data, a set of software processing and, a set of end-user presentation channels. The main goal of component based software is to reduce development costs and efforts, while improving the flexibility, reliability, and reusability of the final application due to the (re)use of software components already tested and validated.

This approach moves organizations from application *development* to application *assembly*. Therefore, one of the key issues of building applications from reusable components is *interoperability*. Interoperability can be defined as the ability of two or more entities to communicate and cooperate regardless of differences in the implementation language, the execution environment, or the model abstraction. Traditionally, two main levels of interoperability have been distinguished: the *signature level* (names and signatures of operations), and the *semantic level* (the "meaning" of operations). In the first place, syntactic interoperability is now well defined and understood, and middleware architects and vendors are trying to establish different interoperation standards at this level. They have given birth to the existing commercial component models and platforms such as CORBA, EJB, or DCOM. However, all parties are starting to recognize that this sort of interoperability is not sufficient for ensuring the correct development of large component based applications in open systems. On the other hand, the existing proposals at the *semantic level* provide component interfaces with information about their *behavior*. Behavior indicates how software responds to external events. The formal specification of the system can be expressed in term of model-based specification. Model-based techniques model the system using mathematical constructs such as sets and functions. The operations in a model-based specification are defined using pre- and post-conditions on the system state. Although much more powerful than mere signature descriptions, dealing with the behavioral semantics of components introduces serious difficulties when applied to large applications. In fact, most of the formal notations have been around for several years already, and they recurrently appear whenever there is a new prototype, which can be Abstract Data Types (ADTs), objects, or components. The use of formal methods for proving "*semantic correctness"* of components in complex applications remains an active area of research. This paper also considers interoperability at the *protocol* level, which deals just with the partial order between the components' exchanged methods, and the blocking conditions that rule the availability of the components' services. This level, firstly identified by Yellin and Strom, provides more powerful interoperability checks than those offered by the basic signature level. Of course it does not cover all the semantic aspects of components, but then it is not weighed down with the heavy burden of semantic checks. At the protocol level, the problems can be more easily identified and managed, and practical (tool-based) solutions can be proposed to solve them. CAPE-OPEN (CO) is a key technology for interoperability and integration of process engineering software components. IBM Glossary (2004) defines interoperability as the *capability to communicate, execute programs, or transfer data among various software units*. This paper analyzes software interoperability for Component based software applications.

3. Software Architecture and Technologies:

The different steps of software system development require one to view the system with respect to several individual perspectives such as those of end-users, analysts, designers, developers, etc. The software architecture is a good target as a candidate to manage these different points of view along the system lifecycle [2]. UML authors also recommend making use of a development process, which is *architecture-centric.* According to Booch *et al.* (1998), software architecture encompasses the set of *significant decisions about the organization of a software system* such as:

- selection of the structural elements and their interfaces by which a system is composed, behavior as specified in collaborations among those elements;
- Composition of these structural and behavioral elements into a larger subsystem and architectural style that guides this organization.

Software system history can be distinguished as follows: centralized architecture in 70's, decentralized architecture in 80's, distributed architecture in 90's and web architecture in 2000's. The use of web technologies has allowed more complex functionalities to be offered on the net; from *information publishing to heterogeneous application integration*. Web (enabled/distributed) architecture is based on multi-tier architecture that separates the *presentation, business logic* and *data*. We use the architectural vision to identify and place the different Information Technologies to select for building the system [3]. This approach concerns not only the physical view but also the logical view (*e.g.* code organization, application design.). The basis for web architecture is shown in Fig. 2.

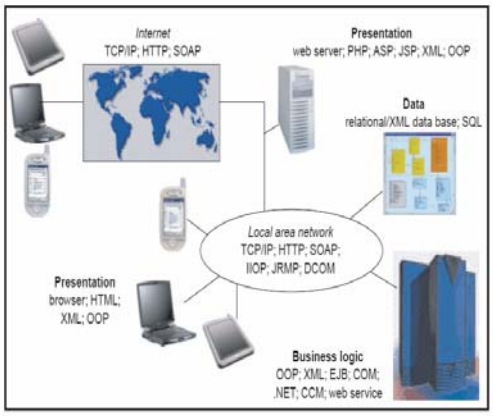

Fig. 2 Web Architecture Basis

The *presentation tier* allows the graphical interaction with the end-users over the network using a *thin client (*the browser) or a *rich client* (a dedicated GUI). The thin client presentation is performed using web browser-HTML (with script languages and XML if any). The communication with the business logic tier is based on HTTP-TCP/IP. Web dynamic technologies such as PHP, Microsoft ASP and Java JSP do not differ from this principle since these pages are compiled and executed on the web server side to generate just-in-time HTML pages displaying the graphical interface of e-business or other applications. On the other side, the rich client presentation is developed with usual Object-Oriented Programming (OOP) languages such as Java, VB, C++, C#, Delphi, and the communication is carried out by protocols from middleware technology such as CORBA-IIOP, (D)COM, .NET Remoting, Java RMI-JRMP, XML-HTTP and SOAP according to component based programming. The middleware technology is the basic mechanism by which software components transparently make requests to and receive responses from each other on the same machine or across a network.

The business logic tier encloses the application logic representing the enterprise know-how and rules. Usually this side is developed according to a component-based approach with Unified Modeling Language. This approach is based on advanced component models such as EJB from SUN / Java community, COM, DCOM and .NET from Microsoft, CORBA/CCM from OMG or Web services from W#C. These components are mainly implemented following an OO Programming and component intercommunication is performed by middleware inner protocols. The components run within a software framework called application server that provides a set of technical services such as transaction, identification, load balancing, security, data access, and persistence.

The data tier has the data persistence service with relational, XML and object databases generally using SQL to manage data. In addition to usual data techniques, work is being done on XML-enabled and distributed data management (SQLXML). Table I categories web architecture into six groups.

TABLE I
WEB ARCHITECTURE CATEGORIZATION

Technology	Protocol / Languages	Functions
Modeling Technology	Internet Protocol: TCP/IP,HTTP; XML (data model)	Modeling for object based systems
Communication technology	Internet Protocol: TCP/IP,HTTP; Internet Inter -ORB Protocol (IIOP) for CORBA; DCOM, .NET Remoting, Java RMI-JRMP (Java Remote Method Protocol), SOAP(Simple Object Access Protocol) , Specific definition programming language such as OMG IDL for CORBA, Microsoft IDL for COM, Java interface for RMI and WSDL for web services.	Data transmission over Internet
Implementation Technology	Internet Protocol: TCP/IP, HTTP; Internet Inter -ORB Protocol (IIOP) for CORBA; DCOM, .NET Remoting, Java RMI-JRMP (Java Remote Method Protocol), SOAP(Simple Object Access Protocol); Object-oriented programming (Java, C++, Eiffel, C#), Web Programming (HTML, XML, ASP, JSP, PHP, PERL).	Coding using object-oriented web programming languages.
Packaging Technology	Internet Protocol: TCP/IP, HTTP; Internet Inter -ORB Protocol (IIOP) for CORBA;	This deals with the creation, management and

	DCOM, .NET Remoting, Java RMI-JRMP (Java Remote Method Protocol), SOAP (Simple Object Access Protocol); EJB, (D)COM, .NET-Programming languages.	destruction of the business component. with this technology, the component developer no longer needs to write "technical"code that handles transactional behavior, security, database connection pooling.
Bridging Technology	Internet Inter -ORB Protocol (IIOP) for CORBA; DCOM, .NET Remoting, Java RMI-JRMP (Java Remote Method Protocol), SOAP(Simple Object Access Protocol); COM-Java RMI, EJB-.NET	Bridging technology allows one to extend the aptitude of a system to interoperate between outer technologies.
Memory Technology	Internet Protocol: TCP/IP, HTTP; relational, object, XML data base, SQL	Provide shared controlled access to schema information.

In order to develop such modern software applications and systems, technology selection involves many criteria. One main issue is to know if the technology is an *(open) standard technology or proprietary technology.* Open standard technologies are developed by software engineering from IT/software companies who collaborate within "neutral" organizations such as W3C, OASIS and OMG in accordance with a *standardization process.* It is worth noting that this trend, issued from web philosophy, is also present in process and software engineering field. Open standard technologies are freely distributed data models or software interfaces. They provide a basis for communication and common approaches and enable consistency [4], resulting in improvements in developments, investment levels and maintenance. Clearly, the common effort to develop IT standard or domain oriented standard and its worldwide adoption by a community can be a source of cost reduction because not only is the development cost shared but also the investment is expected to be more future-proof. Open computing promises many benefits: flexibility/liveliness, integration capability, software editor independence, low development cost and adoption of technological modernization. It is also possible to build Information System from enterprise software especially for enterprise management. Enterprise Resource Planning (ERP), Enterprise Application Integration (EAI) and portal applications can provide build-in solutions that need to be tailored to enterprise framework and requirements. These solutions, open source or commercial, involve standard technologies and yielding web architecture. The following sections deal with *component based development technologies for software interoperability* unambiguously *communication, packaging* and *bridging* technologies.

III. Overview of Communication Technology

Components Class and Interfaces:

An *interface*, a key element for middleware technology, is a collection of possible functions used to specify through its operations the *services of a software unit*. Depending on the selected middleware technology, interfaces are developed with a specific definition programming language such as OMG IDL for CORBA, Microsoft IDL for COM, Java interface for RMI and WSDL for web services. A class is an object-oriented concept. It describes a set of shared objects and belongs to the *implementation step*. An *object* is an instance of a class. An object satisfies an interface if it can be specified as the target object in each potential request described by the interface. It belongs to the implementation step. However this object is distinct from the other usual objects since it collects the remote calls. The development of distributed software does not imply the choice of an actual object-oriented language (commonly C++, VB and Java) since middleware's such as COM and CORBA introduce the notion of pseudo-objects. The *component* is a software unit that encapsulates the implementation of business process logic. Sessions R. stresses the difference between component and object technology, the former being a *packaging and distribution technology,* focusing on what a component can do, while the latter is an implementation technology, focusing on how a component works [5]. Objects and components are software entities; objects are typically *fine-grained units, interact in the same computing process while components are rather universal grained units,* and are available outside their own process with respect to interface definitions. They are issued from different software design. The difference between these two states is clearly identified in the CAPE-OPEN standard from *CAPE-OPEN-Laboratory Network.* A CAPE-OPEN accommodating component is a piece of software that includes the supplier proprietary codes—objects or not—which recognize or/and use CAPE-OPEN interfaces. The communication between CAPE-OPEN component instances is defined unambiguously by the CAPE- OPEN interfaces [6]. In this case, the middleware technologies are CORBA and COM.

Middleware Principles:

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. Figure 3 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 (D)COM and .Net Remoting from Microsoft, CORBA from OMG and RMI from Java/SUN. COM, CORBA and SOAP are detailed in following sections. 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). Normally, these methods are used in Information Systems based on open computing architecture. 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 [7].

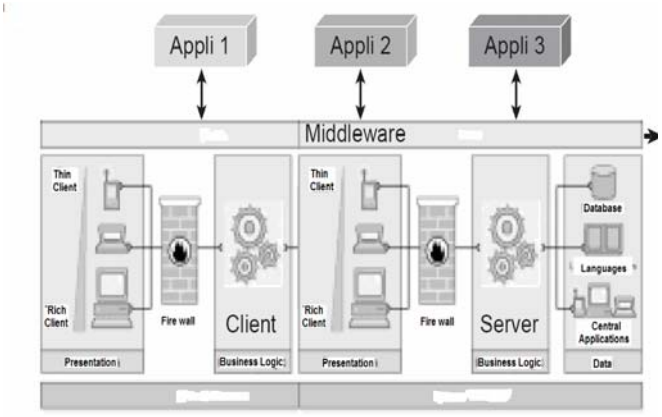

Fig. 3 Middleware and client/server model

– *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 (little endian, big endian), 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).

Implementation of Components:

The development of software that involves components is known as *component based software engineering* [8]. This is special case of object-oriented software engineering, and development techniques and methodologies apply here. Object oriented software development methodologies exist and the implementation of software components benefit from the use of these methodologies e.g. unified process [9] discusses unified process applied to COTS, iterative models for components integration [10]. In this paper, we address the particular problem of the implementation of software components. The main idea in components is to identify *business objects* that correspond to specialized activities, such as modeling, thermodynamics, numerical resolution, control and advanced control. Specialized engineers are developing new algorithms inside specialized components that implement interfaces dedicated to the corresponding domain. Each component can evolve independently and remain passive as long as it preserves the behavior of interfaces. The use of UML (Booch *et al.,* 1998) as a modeling language is the central point of *software engineering tools.* Some processes for software development employ standard use of UML and tools to develop software of high quality with an iterative lifecycle. UML use cases are good examples to describe project requirements.

The analysis is done using class diagrams with definition of high-level classes, packages and interfaces for components. The developer has a global view of the classes and packages. Classes can implement different levels of complexity and heterogeneous environment. The development can be iterative due to the use of components. Application team (like database search, computational problems.) develops application classes

and IT team for communication, management of middleware data types and memory allocation, develops technical classes. This separation is natural with UML class diagram. The use of middleware types and structures is very complex and has to be developed by a specialist IT team in specific technical classes. UML CASE (Computer Aided Software Engineering) tools enhance the capacity for changes in *round-trip engineering.* This is always important to have an up-to-date UML model. Generation of analysis and design documents can be done from this model and the class source code. The round-trip functions are essential for traceability quality requirements. The framework of a modern CASE tool is able to handle links with tools for software development:

Editor of source code;

– Wizard of code environment, such as wizard to generate classes for Microsoft COM interfaces;

– Configuration management tool.

Thus component based software development and UML modeling allow efficient cooperation of applicative and IT teams in an iterative lifecycle process.

IV. Communication Technologies by Data Model

Inter-application communication by data model requires the definition of a standard data format, because the effective representation of the data is the heart of this model. In order to be adopted by major actors of application development, such a format should be standard, robust and open (*i.e.* can be easily tailored to specific business needs). The W3C consortium released the XML specification to address this problem.

The XML Language:

EXtensible Markup Language (XML) is a simple, very flexible text format derived from Standard Generalized Markup Language (SGML). Originally designed to meet the challenges of large-scale electronic publishing, XML from W3C is also playing an increasingly important role in the exchange of a wide variety of data on the web and elsewhere. XML is an extensible file format because it is not a fixed format like HTML (a single, predefined markup language). Instead, XML is actually a meta-language—languages for describing other languages—, which can be specialized to meet the needs of a particular domain, like scientific or business domains [11]. XML targets web development, due to its syntax being close to the syntax of HTML, thus providing natural transformations to produce web documents, but also targets other computer areas, such technical data handling, knowledge management. XML files are based UNICODE, which provides consistent representation whatever the language of the writer of the file and its reader.

A DTD (Document Type Definition) is a formal description in XML declaration syntax of a particular type of document. It sets out what names are to be used for the different types of element, where they may occur, and how they all fit together. DTD can be inline directly in an XML file, as well as being referenced as an external resource, that may be shared by different files. Given a DTD, the XML parser that reads the document, thus avoiding additional verifications of the document, can directly enforce the validity of a XML document. An XML schema is another language that expresses syntax constraints on XML files, but instead of DTD, this language itself is based on a XML-like syntax. XML files contain data along with enclosing tags describing the semantics of the data; these files can be processed in order to transform the structure of the file. XSL (eXtensible Stylesheet Language) aims at easy transformation of XML files into files of different format, XML compliant or not. XSL is a functional language that associates to the elements of the input XML files a set of transformations of the data contained in the input file. One of the main use of XSL language is to transform XML files containing technical data, for example a list of data, into a more user friendly presentations, for example a table containing exactly one data per line, with associated color set depending on the nature of the data, and displayed on a standard internet browser. The XSL Transformation Architecture is shown in Fig. 4.

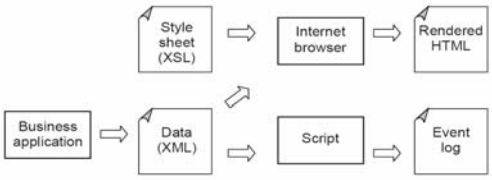

Fig. 4 XSL Transformation

The business application produces an XML compliant data file, which can be used by other, domain specific, applications or by a generic tool (like a browser) which can produce a user friendly view of the data contained in the input file, with the help of business standard style sheet.

XML Specializations:

Since XML is a meta-language, it gave birth to other languages, which are dedicated to particular business domains. Amongst the many specializations of XML, we can enumerate:

– SVG is an XML sub language that is used nowadays to specify vector graphics and render them in commercial browsers.
  - XMI from OMG is an XML language that can be used to describe UML entities, such as Classes, diagrams, relationship, etc.
  - MathML from W3C is an XML language for mathematical expressions.

V. Packaging Technology

Middleware components run within a controlled runtime environment provided by the server editor. This packaging technology deals with the creation, management and destruction of the business component. With a technology, the component developer no longer needs to write "technical" code that handles transactional behavior, security, database

connection pooling, because the architecture delegates this task to the server supplier. These techniques are now widely used in network solutions and this section describes the different technologies.

EJB, Java Community Technology:

Enterprise JavaBeans:

Enterprise Java Bean (EJB) proposes a high-level approach for building distributed systems. It allows application developers to concentrate on programming only the *business logic,* while removing the need to write all the common code required for any multi-tier application development scenario [12]. For example, the EJB developer no longer needs to write code that handles transactional behavior, security, connection pooling or threading. In essence, EJB is a *server component model* for Java and is a specification for creating server-side, scalable, transactional, multi-user, and secure enterprise-level applications. EJB can be deployed on top of existing transaction processing systems including traditional transaction processing monitors, web servers, database servers, application servers.

Benefits of Using EJB:

In multi-tier architecture, it does not matter where the business logic is. With EJB, business logic can reside on any server, while adding additional tiers if necessary. The EJB's containing the business logic are platform-independent and can be moved to a different, more scalable platform if necessary. An EJB can move from one platform to the other without "any" change in the business-logic code. A major highlight of the EJB specification is the support for ready-made components. This enables one to "plug and work" with *off the- shelf* EJB's without having to develop or test them or to have any knowledge of their inner workings.

EJB Communication:

RMI is used as communication protocol for EJB clients. However, EJB may provide CORBA/IIOP protocol for a robust transport mechanism and pure CORBA clients can access EJB as EJB clients. Currently, a highlight of OMG's CORBA Services is the wide range of features they provide to an enterprise application developer. In the future, rather than trying to rewrite these services, EJB server vendors may simply wrap them with a simplified API, so EJB developers can use them without being CORBA experts.

Java 2 Platform, Enterprise Edition (J2EE):

The Java 2 Platform, Enterprise Edition (J2EE) is a set of coordinated specifications and practices that together enable solutions for developing, deploying, and managing multi-tier server-centric applications. Building on the Java 2 Platform, Standard Edition (J2SE) the J2EE platform adds the capabilities necessary to provide a complete, stable, secure, and fast Java platform to the enterprise level. It significantly reduces the cost and complexity of developing and deploying multi-tier solutions. Enterprise Java Beans are part of the J2EE platform but the platform provides also many other key technologies, for example complete web services support.

.NET, Microsoft Technology:

The *Microsoft* response to J2EE is called .NET, and provides a set of standard components, languages, etc., aimed at the development of business applications [15]. The different elements are:

– The CLR (Common Language Run-time) is the mechanism that allows every compliant language to interoperate closely (objects defined in one language can be used in another one). Languages such as C#, VB.NET, can be compiled "on the fly" into *Intermediate Language* (IL). The resulting code is called managed code because the IL provides services and concepts that help the execution of such code (for instance, garbage collecting to prevent memory leaks, sandboxing to prevent malicious code being executed, etc.). On the other hand, CLR also enables the execution of *unmanaged code,* letting the global security policy of the virtual machine decide if it can be allowed. The CLR is the equivalent of *Java virtual machine,* the IL of the *Java byte code.*

.NET common classes are a set of common classes that are provided by the framework and that ease the enterprise application development. These classes are dedicated to management of files and can be considered as a replacement for *Microsoft* foundation classes.

ASP .NET provides classes used during Active Server Pages (ASP) creation, enabling the execution of C# code within HTML pages.

.NET remoting is the .NET middleware technology that handles the deployment of distributed applications in NET [13, 14] compares it to web services.

The .NET and J2EE architectures are very similar, each having its advantages, and its respective defaults. The key technology is the programming language, *i.e.* C# for Microsoft .NET and Java for EJB. These object oriented languages are similar in scope (simplified OO languages), run on virtual machines and thus are naturally portable on different architectures (Windows CE, XP for .NET, and all UNIX flavors for Java).

CCM, OMG Technology:

CORBA Component Model (CCM) is a specification that focuses on the strength of CORBA as a server-side object model. It concentrates on issues that must be addressed to provide a complete server side middleware component model. *It can be described as a cross-platform, cross language superset of EJB.* The CCM gives developers the ability to quickly build web-enabled enterprise scale applications while leveraging the industrial strength of CORBA. Tight integration with EJB advantages CORBA's *cross platform* and *multiple-language* capabilities. The CCM is part of the CORBA 3.0 specification [16]. It extends the CORBA object model by defining features and services in a standard environment that enable application developers to implement, manage, configure and deploy components that integrate with

commonly used CORBA services. These server-side services include transactions, security, persistence and events.

Web Services, W3C Technology:

Web technologies are more and more used for application-to-application communication as explained in previous sections. At first, software suppliers and IT experts promised this interconnected world thanks to the technology of web services. Web services propose a new paradigm for distributed computing [17] and are one of today's most advanced application integration solutions [18]. They help business applications to contact a service broker, to find and to integrate the service from the selected service provider. However, even if the idea of web services has generated too many promises, web services should be viewed for now *as a part* of a global enterprise software solution and not as a global technical solution. In a project, web services can be used within a general architecture relying on Java EJB or on Microsoft's .NET framework [19].

VI. Bridging Technology

Above we dealt with interoperability features within a particular inner technology. Bridging technology allows one to extend the aptitude of a system to interoperate between outer technologies. Two bridges are illustrated, given that SOAP and web services, as noted previously, may play the role of connector. Enterprises are characterized by an organization networked through their information system in which all the elements have to interact. This results in an increasing dependence with regard to information technologies for interoperability. Corresponding multi-tier architecture information systems are today built over advanced EJB or .NET component frameworks, themselves relying on middleware technologies such as CORBA, RMI and (D)COM. Initially EJB technology is multi-system and mono-language (Java) while .NET technology is mono-system (Windows) and multi-language. CCM aims at proposing a multi-system and multi-language technology. Solutions exist to "unlock" EJB and .NET. For example Common Language Infrastructure (CLI) and C# programming language from .NET are now standardized by ISO. Implementations on *e.g.* Linux are available. In addition, web services and SOAP can give many benefits. Andrews predicts dramatic changes in the web services market for 2006, and announces a new class of business applications called "service-oriented business applications" [20]. The merging of web standards, IT and object/component technologies to form SOA and web services is announced as the next stage of evolution for e-business knowing that grid and autonomic computing should add their contributions too. There is no doubt that the scientific field will derive many benefits from this trend. The engineer already benefits from information technologies for interoperability, especially with XML, COM and CORBA.

VII. Practical Scenario

Interoperability is the ability of systems, units, or forces to provide services to and accept services from other systems, units, or forces and to use the services exchanged to enable them to operate effectively together. A number of mechanisms are required in order to achieve interoperability between the database management systems running on different platforms. These include:

- A mechanism for issuing requests
- A mechanism describing data (self-describing data strings are useful)
- A mechanism for performing data format translations
- A mechanism for performing joins
- Rules and a protocol for issuing requests and receiving responses in a structured manner.

In a multi-tier environment, remote data access is never seamless. The ability to provide transparent data access to heterogeneous databases is exactly what large Information System organizations desire most. Transparency implies that in the user's view (the external schema of the three-schema architecture), the physical location of the data (the internal schema) is completely hidden; the enterprise database appears homogeneous. The client and server components rely on the availability of Communication protocols, security, transaction management, replication, and management services within their respective environments. In order to achieve complete interoperability in a multi-tier environment, these services must be consistent and must be universally available across the heterogeneous platforms. For example consider a configuration of databases for student's data stored using POSTGRES under Linux Server, MYSQL under Linux Server, Oracle under Windows Server, and MYSQL under Windows Server. The user can frame a query over distributed relation stored at different servers located in different sites and the components at respective sites coordinate in semantic integrity check, query execution, and send the result to the initiating site.

VIII. Open Problems in Interoperability in Component-Based Systems

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.

3. Components are packaged and delivered in many different forms (example: function libraries, off-the-shelf 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 tailorable 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.

IX. 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, behavioral specifications of components, service access protocols, business domain knowledge and shared ontology, negotiation of Quality of Service and other non-functional properties. XML allows the flexible development of user-defined document types. It provides a robust, non-proprietary, persistent, and verifiable file format for the storage and transmission of text and data both on and off the web [11]. It can be customized to meet the users' needs in many kinds of applications. Whereas CORBA, RMI, (D)COM and .NET Remoting try to adapt to the web, SOAP middleware ensures a native connectivity with it since it builds on HTTP, SMTP and FTP and exploits the XML web-friendly data format. Reasons noted for the success of SOAP are its native web architecture compliancy, its modular design, its "simplicity", its text-based model (in contrast to binary and not self-describing CORBA, RMI, (D)COM, .NET protocols), its error handling mechanism, its suitability for being the common message handling layer of web services, its standardization process and its support from major software editors. Enterprises are characterized by an organization networked through their information system in which all the elements have to interact. This results in an increasing dependence with regard to information technologies for interoperability. Corresponding multi-tier architecture information systems are today built over advanced EJB or .NET component frameworks, themselves relying on middleware technologies such as CORBA, RMI and (D)COM. Initially EJB technology is multi-system and mono-language (Java) while .NET technology is mono-system (Windows) and multi-language. CORBA Component Model aims at proposing a multi-system and multi-language technology. This paper has dealt in detail, the Interoperability issues pertaining to Component based software development.

References

[1] Gray T. Leavens and Murali Sitaraman, Foundations of Component-Based Systems, Cambridge University Press, 2000.
[2] Hofmeister, C., Nord, R. and Soni, D, Applied Software Architecture, Addison- Wesley Longman, 2000.
[3] Serain D. Enterprise Application Integration – L' architecture des solutions e-business, avril 2001.
[4] Fay, S. Standards and reuse. The Rational Edge, May 2003..
[5] Sessions. R. Objects and Components, ObjectWatch newsletter number 28, June 2000.
[6] Beland, J. P. and Pos, M. Open Software Architecture Beland, J. P. and Pos, M. Open Software Architecture.
[7] Wassermann A. I. Tools Integration in Software Engineering Environments, spinger-verlag, Berlin, 1998, pp.138-150.
[8] Brown A. W., Component –Based Software Engineering: selected papers from the Software Engineering Institute, Wesley – IEEE Computer Society Press.
[9] Chan, R. Adopting RUP in a COTs implementation Project. The Rational Edge, May 2003.
[10] Boehm B. Spiral Development: Experiences Principles, and Refinements. Spiral Development Workshop, Ed. Wilfred J. Hansen, February 2000.
[11] XMLFAQ(2004). The XML FAQ http://www.ucc.ie:8080/cocoon/xmlfaq.
[12] Peltzer D. (2003) .Net & J2EE Interoperability, November 2003.
[13] Browning, D. Integrate .NET Remoting into the Enterprise Windows Server System http://www.ftponline.com/wss/2002_11/magazine/features.
[14] Holloway, R. Compare >NET Remoting to Web Services, Visual Studio Magazine, Web services in the Enterprise, 12-11, 2002.
[15] TMC. The Petstore Revisited: J2EE Vs .Net Application Server Performance Benchmark, November 2002.
[16] Object Web Consortium 2004. www.objectweb.org.
[17] Bloomberg, J. Web Services: A New Paradigm for Distributed Computing, The Rational Edge September 2001.
[18] Linthium, D. S. Next Generation Application Integration from simple Information to Web Service, September 2003, Addison Wesley.
[19] Newcomer, E. Decide between J2EE and .NET Web Services, Windows Server System Magazine - 2,9, 2002.
[20] Andrews, W. Predicts 2004, Gartner's Prediction . http://w3.gartner.com/research/ spotlight/ assert_55117_895.jsp