Web-based Legacy System Migration and Integration*

Ying Zou and Kostas A. Kontogiannis

Dept. of Electrical & Computer Engineering
University of Waterloo
Waterloo, ON, N2L 3G1, Canada

ABSTRACT

With the exponential growth of the Internet and the multi-tier distributed system architectures, there is an urgent demand to develop Web-based and component-based applications to reduce the time to the market and to leverage existing software. This paper presents an approach to migrate the existing legacy applications to a Web-enabled Network-Centric environment. The migration process focuses on specifying the identified legacy components in XML, consequently wrapping them by CORBA objects, and finally deploying the distributed component into the application server. A scripting language that is encoded in an XML format can be used for allowing thin clients to communicate with legacy components.

Keywords: Multi-Tier Distributed Architecture, Web-Based, Wrapping, Legacy Software, Network-Centric Environment, Integration, Migration.

1. INTRODUCTION

With the growth of the Internet, there is an urgent need to develop Web-based applications. Initially the Web was presented as a giant URL-based file server for publishing static information in a hypertext electronic form. With the incorporation of client/server architectures, CGI scripts, and Java applets, the Web is evolved into an interactive medium that provides dynamic information services. However, these tools are slow and mostly support stateless transactions. Moreover, downloading "fat" Java applets imposes many limitations due to low bandwidth Internet access. Today, new requirements for Web-enabled applications are emerging.

Organizations would like to take advantage of the Web in its various forms (Internet, Intranet, and, Extranets), universal access and 24-hours open for their enterprise computing. This requires the IT departments to port their legacy software assets to a distributed Web-enabled environment in order to leverage the functionality of existing legacy systems without having to rebuild these systems [1].

The largest problem faced by most large enterprises is the integration of their existing applications with the new applications. It is of no surprise that most of the new applications are developed by integrating existing components. In addition, legacy tools and legacy application libraries constitute significant assets for IT organizations [11]. So the integration of heterogeneous applications and systems is dictated by market requirements due to the plethora of operating systems, languages, networking protocols, and data representations. With its universal access across heterogeneous hardware and software platforms, the Web is the best choice to address this challenge to make a loosely coupled, but coordinated application [9].

Component-based software engineering has gained significant attention in recent years. Borrowing the idea from the "plug-and-play" hardware components, such software components, with well-tested code, can increase the quality of a software product, shorten the development time, reuse, and extend the lifecycle of the large-scale pre-packaged software. Business interest has been driven by competitive pressures to deliver agile applications more rapidly and economically.

Moreover, there is a constant shift towards thin-clients in multi-tier architectures away from the "fat-client" paradigm that is predominant in the traditional client/server topologies. Thin client architectures, which communicate with servers by means of the HTML and HTTP protocols, shift the focus from the client to the server, by leaving as little code as possible on the client side. This results in faster applet downloading and less client RAM requirements. Thin-clients require the servers to process the data and to provide complete services, such as transactional service, security service, and naming service.

Finally, the portability of code and applications is a critical issue, as more and more types of devices and embedded systems (i.e. handheld devices with wireless IP), become integral part of distributed applications.

It is apparent that the current HTTP/CGI paradigm cannot meet these requirements. The next generation of the Web, the so-called Object-Web, applies distributed object computing technologies to multi-tier architectures, and is gradually adopted as the development and deployment platform for distributed applications [5].

This paper is focusing on methodologies for the migration of legacy systems to a Web-enabled environment in the form of distributed software components and sketches the architecture for the legacy system integration using the Web as the medium.

The remainder of the paper is organized as follows. Section 2 introduces the enabling technologies. Section 3 sketches the overall architecture and major steps for the migration and integration of legacy system. The concrete methodologies for

*This work is funded by IBM Canada Ltd. through the Toronto Laboratory, Center for Advanced Studies.

each step are explored in Section 4, Section 5, Section 6, and, Section 7 respectively. Section 8 concludes the paper and outlines areas of future work.

2. ENABLE TECHNOLOGY

2.1 Distributed Software Component

A complete definition for the software component is given in [12] as:

"a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties."

With the advances in networking, the software components are moving from the desktop paradigm towards enterprise-wide, distributed, network-centric environments. A distributed component possesses the following characteristics [19]:

- An explicit and well-defined interface defining the services it provides, and the structures of the passing parameters it uses;
- An explicit specification for describing the behavior and usage of the component, the required execution environments, and the location of the component;
- Software independence from its clients by encapsulating the detailed implementation;
- Communication with binders for registering with the clients via the Internet, Intranets or Extranets;
- Means for remote access and remote invocation of services;
- Provision for run-time dynamic component configuration.

The major middleware technologies, including CORBA, RMI, Enterprise JavaBeans, and DCOM, meet the above requirements and provide the integration environment for the distributed software components. These are discussed in more detail in the following sections.

2.2 Major Middleware Technologies

CORBA

CORBA connects and composes distributed objects with client applications, including legacy applications and databases, beyond the boundary of a single programming language.

OMG CORBA has several advantages over other middleware, including platform, language, and vendor independence. CORBA ORBs are available on almost all operating systems, such as Windows/NT, Unix, AIX, and Linux. The CORBA Interface Definition Language (IDL) separates interfaces from implementations. IDL as the uniform interface bridges different programming languages, including Java, C++, C, Smalltalk, and COBOL. It hides the implementation details from the client code and integrates heterogeneous languages. CORBA ORBs serve as the communication buses. They intercept any requests that the client makes, and find an object that can implement that request, then pass the required parameters to the server to invoke remote methods, and then return the results to the client. The client does not need to be aware of the physical location of the remote object, its implementation language, and the underlying operating system. CORBA IIOP over TCP/IP is an open industry standard, which allows ORBs from different ORB vendors to seamlessly inter-operate. IIOP is supported by most web browsers, which enable the Internet access to CORBA objects.

RMI

RMI is a Java-only solution for the distributed computing. Since it is implemented in Java, it is platform-independent. Similar to CORBA, it generates the stub and skeleton classes for the client and server. RMI makes use of two other protocols: Java Object Serialization and HTTP. The Object Serialization protocol is used to marshal call and return data. It enables RMI passing the full objects as RMI operation parameters and return values, whereas, CORBA allows to pass only the object reference back and forth. The HTTP protocol is used to "POST" a remote method invocation and obtain return data when circumstances warrant.

Enterprise JavaBeans

Enterprise JavaBeans is a Java-based, server-side component architecture for developing, deploying, and managing enterprise-level applications. It deals with the "infrastructure" code for deploying distributed object applications. It allows the developers to focus on the business logic and provides the standard component infrastructure, which enables the enterprise developers to quickly assemble distributed components.

CORBA and Enterprise JavaBeans have been developed independently. But they are complementary and can be seamless integrated. JavaBeans provides CORBA/IIOP as the transport mechanism for the pure CORBA client and server. An EJB server can be built on top of an ORB by using CORBA Naming Service and transactional services, etc.

DCOM

DCOM supports Windows platforms. DCOM-based applications can take full advantage of existing Microsoft services such as security and transactions. Some vendors are migrating DCOM to other platforms such as UNIX. DCOM may eventually become a truly cross-platform environment.

2.3 Distributing Components on the Web

XML (eXtensible Markup Language), is a metamodel for structured document exchange based on the Standard General Markup Language (SGML). It can be transferred via HTTP, and provides the machine understandable and human readable syntax, which allows the developers to define their own tags in different domains. For example, XML tags can be used to define the configuration information for the distributed software component. By contrast, HTML limits the expressiveness of a Web document by its pre-defined set of tags. XML will be one of the key technologies for the future Web application. Currently, XML is widely used as the distributed data standard format for data representation, data integration, data storage, message exchanging, or as a scripting language. XML is especially suited to handle a large volume of human-readable data [13, 14].

CORBA IDL (Interface Definition Language) is a standard format to describe the distributed component on the Web via the IIOP. It supports the basic specification for the distributed component, such as the operations and attributes provided by the component. However, IDL can not describe all of the information needed by the client to make use of the distributed component, for example, the configuration of the component and requirements for execution.

The combination of XML and CORBA IDL can facilitate to distribute the software component over the Web.

3. PROPOSED ARCHITECTURE

Businesses, agencies, and software houses have invested a lot of money on their legacy applications. At this point, no one can afford to spend the same money, effort, and risk again by re-implementing the same systems for another platform or programming language. One way to minimize risk and cost is to translate the legacy code from one language to another by using translator programs [7,15]. Even though this approach solves the problems of porting a legacy application form one platform to another, it does not solve the integration issues. Another approach is to re-architect and to translate the legacy system into an Object Oriented way and language (i.e. C++ or Java) [2, 16, 17]. Each identified object along with its associated methods is supposed to implement a specific function of the original system. Wrappers can be applied to

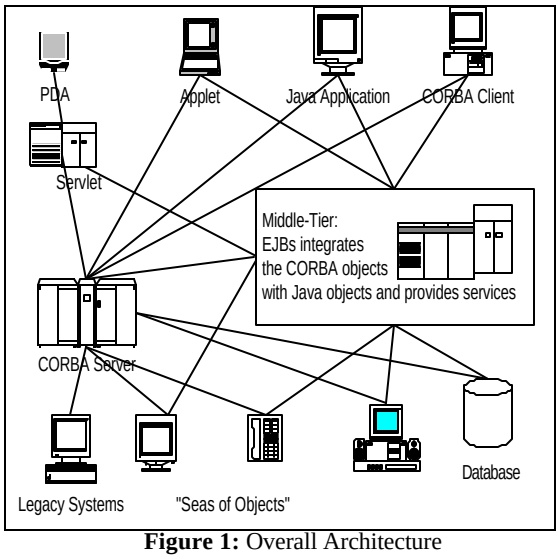

Figure 1: Overall Architecture

encapsulate each C++ or Java object obtained from the original system and make it available to a distributed Network-Centric environment. Even though this approach is promising, it causes a lot of network traffic due to the large number of objects involved.

The proposed approach focuses on the use of Web as an infrastructure medium, the utilization of Reverse Engineering to identify coarse granularity, self-contained components, and the use of Distributed Object Technologies to address issues that stem from distribution (message passing, concurrency control, interaction semantics). In brief, the proposed approach consists of the following steps:

1. Use reverse engineering techniques and restructuring to identify and generate a decomposition of the legacy system into modules.
2. Analyze the interfaces of the selected legacy components and store their signatures in a component repository using XML format.
3. Generate the CORBA/IDL and CORBA wrappers from the component repository.
4. Use EJBs (Enterprise JavaBeans), or Servlets to integrate the CORBA wrappers and to provide the services to the Web-based applications.
5. Define a scripting language using XML, to allow for the utilization of Web technologies for legacy components to be invoked and computational tasks to be specified.

The three-tier architecture is fundamental to deploy the legacy systems into the web-based application, shown in Figure 1. In the back-end, the legacy systems are wrapped by the CORBA as distributed objects. They serve as the object repository and provide the building blocks for the other distributed objects. The middle-tier such as application server integrates the distributed objects and builds the available services for the front-end clients. The front-end clients can be the fully functioned desktop, the diskless PDA, or the network computer.

4. IDENTIFICATION OF COMPONENTS

A software component can be considered as “an independently-deliverable software package which offers services through interfaces”[1]. It offers specific functionality and can be easily integrated with other COTS or custom made software components.

So far in the field of reverse engineering, most software analysis efforts have been focused on clustering and decomposition techniques based on cohesion, coupling, and other source code features. In most cases, legacy system decomposition has become a clustering problem. Clustering is a successful technique on decomposing a legacy system into subsystems. Nevertheless, these identified subsystems in most cases are not good candidates for distributed software components. The reason is that clustering heavily depends on static code features (coupling, cohesion) as opposed to dynamic requirements that are imposed by the architecture of the target, distributed application. From the legacy procedural code, it is especially difficult to recover clear interfaces via clustering due to prolonged maintenance and extended modifications the legacy system has undergone. A possible solution is to make the whole legacy system a reusable unit by wrapping it as a black box and provide the signature as an API obtained by the original legacy interface. However, the larger the software component, the less flexible it will be.

The proposed approach is to transform the procedural code to object oriented classes and to identify the fine-grained software components for maximizing reusability and flexibility. Since object oriented languages provide mechanisms for information hiding and encapsulation, it is easier to identify the loosely coupled, small-scale, reusable software components. The first step aims at the meta-level description of the software component, in order to support automatic generation of the CORBA IDL and CORBA wrappers that move the stand-alone software components into a distributed platform.

```
public studentInfo{
   private:
          int       ID;
          char      LastName[20];
          char      FirstName[20];
  public:
          char*     get_LastName();
          void      set_LastName(char *lastName);
          char*     get_FirstName();
          void      set_FirstName(char *firstName);
          int       get_ID();
          void      set_ID(int id);
}
```

Figure 2: studentInfo Class Definition

With meta-level, self-descriptive information, the distributed components can be published in the Web-enabled distributed environment. The proposed specification of the distributed components includes the following information [18]:

- General compile-time and runtime properties, including keywords for describing its functionality, execution environment, where to find it and how to activate it, for example, the URL links, the path name.
- Classes that specify the composition of the components.
- External references that point to the external specifications that describe other components.
- Descriptions of public attributes, and methods.
- Attributes and methods that characterize the external interfaces and disclose the source classes.
- Return types and parameters that specify the parameters for method inputs and outputs.

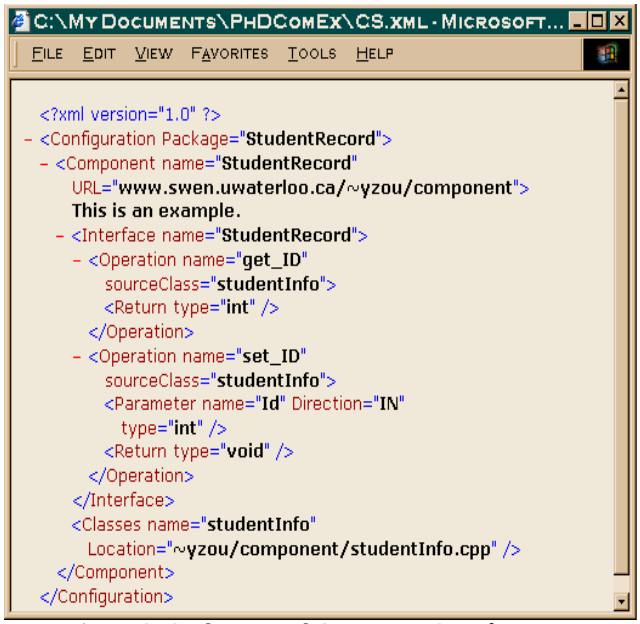

Figure 3: StudentRecord Component Specification

In this environment, it is important to choose a standard format to represent the specifications, so that, the identified legacy software components can be easily understood by the developers and integrated with other applications. XML provides a natural way to represent configuration information for components.

For example, suppose that the component consists of one class as illustrated in Figure 2. and the developer only wants to make the get_ID and set_ID methods available to remote clients. An example specification script for this example is illustrated in Figure 3. Such specifications supply rich information for creating, assembling and deploying component-based software. Moreover, they provide the possibility to dynamically bind the component and execute the component over the Web by extracting the URL of the component location. For example, Internet Explorer 5.0 can interpret and display XML documents, so that, the syntax of the specification can checked right away, and the component catalog and configuration can be searched and viewed on the Web.

5. WRAPPING IDENTIFIED COMPONENTS

CORBA is the appropriate infrastructure to integrate the software components in a heavily heterogeneous distributed environment. The mapping from C++ to CORBA IDL is fixed by the CORBA specification. So the interfaces can be directly generated from the component specification as the following steps:

```
interface StudentRecord{
          long      get_ID();
          void      set_ID(int long id);
}
```

Figure 4: StudentRecord Interface Definition

First of all, a single IDL interface is generated, by denoting the <interface> tag in the XML specification. Secondly, externally visible operation identifiers are extracted from the <operation> and registered in the CORBA IDL, shown in the Figure 4. Finally, the IDL to C++ compiler generates the client-side stub and the server-side skeleton classes.

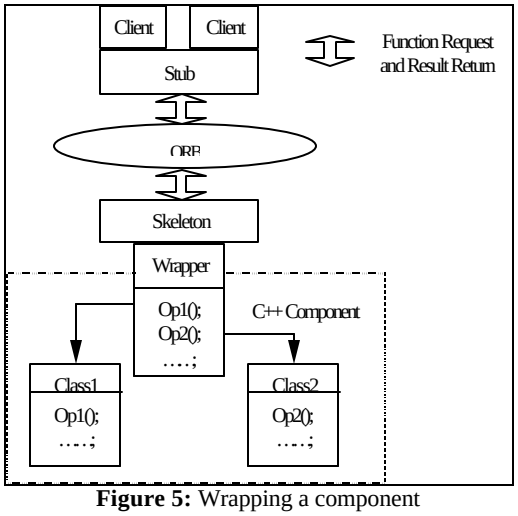

Figure 5: Wrapping a component

The CORBA wrapper inherits from the server-side skeleton classes and encapsulates the C++ classes. The operations in the interface are the public operations in the

CORBA wrapper. According to the "sourceClass" attribute (Fig. 3) value in the XML <operation> tag, the wrapper re-directs the invocation of its public operations to corresponding methods in the encapsulated C++ class, shown in Figure 5. Essentially, the wrapper acts as a façade: it offers clients a single, simple interface to the underlying objects. It glues together the CORBA distributed capabilities and the standalone components. Moreover, new functionality can be introduced in the wrapper, for example, emitting the run-time information, getting the location of component on the fly. It is notable that the object wrappers are housed within the CORBA server infrastructure, providing the back-end services.

6. INTEGRATION OF CORBA OBJECTS

Application servers occupy the middle tier in multi-tier distributed applications. They have been widely adopted as the runtime environment of choice for integrating heterogeneous applications. Different application servers are implemented on different technologies. Enterprise Java Beans (EJBs) provide means to facilitate the invocation of distributed objects in a more standardized way than CORBA. CORBA is suitable for the lower level aspects of distributed applications, leaving the task of defining how components should inter-operate to the programmers, while EJBs handle in a more standardized way issues related to communication, security, and concurrency control. EJBs are considered as a workbench for "plug and play" components, and for developing application servers. The application server assembles the individual CORBA wrappers, integrates them with other remote objects, and makes services available to a front-end Web-enabled application (i.e. a Web browser). EJBs utilize the CORBA naming services to locate the CORBA objects, and to dispatch the request to the CORBA objects over the CORBA IIOP. They provide a link between Web-based applications and the back-end diversity software applications, databases, transactional systems.

The EJBs provide a higher level solution for integration. However, they add another layer on the top of the distributed objects and therefore contribute to degradation in performance time.

Another way for integration is to utilize the CORBA IIOP, which is the OMG solution for the distributed objects to communicate over the Internet. Netscape browser supports IIOP. Web applications can take the form of Java applets for directly integrating distributed objects via IIOP. Initially, Java applets are downloaded via HTTP. Once an applet is fully downloaded, it takes over communication with remote object using IIOP. This approach is suitable for the "fat-client".

Based on the IIOP, another web integration scenario takes advantage of servlet, which enables the "thin-client" in HTML. Servlet becomes the integrator for the remote objects, and hosts in the web server instead of downloading object to the local machine. The use of IIOP as the directly communication protocol will improve the performance and transparency to the end users.

7. SCRIPTING LANGUAGE FOR THIN CLIENTS

In a thin-client environment, applications are downloaded to the clients on-request. Clients can issue individual requests to specific servers using CORBA or any other message passing mechanism. However, given the infrastructure that the IIOP, http, CORBA and mark-up languages now offer, it is much more efficient to allow thin-clients issue multiple requests to many servers. These requests can be building blocks of aggregated tasks that are requested by a thin-client. In addition, the requests can be implemented in a more sophisticated paradigm, such as the Event-Condition-Action paradigm, where specific requests and actions are carried out only after specific events have been intercepted and specific conditions have been fulfilled. In order to allow for thin-clients to poses such functionality, we must define a scripting language that allows for the composition of services available in the application servers, and the formation of aggregating on-demand tasks. The scripting language can be implemented using XML and sent to servers by the standard http protocol. Servlets can be used to intercept and interpret the transmitted scripts from the thin-client. Therefore, the client can take advantage of a Web browser to compose scripts (i.e. Web forms), and does not need a special compiler, or script interpreter.

In such a scenario, XML becomes the primary candidate as the implementation vehicle for the scripting language. An XML-CORBA or XML-EJB bridges are servlets. Once the script is sent to a servlet, the servlet resolves the script, calls the corresponding components in the application server, and returns the result to the front-end client. The servlet translates the scripts into CORBA requests or EJBs requests, and vice versa. From the client's point of view, the scripting language is directly executable.

To interpret the scripts, the bridges should maintain the repository for the mapping between the script keywords and the available services in order to be directly runnable by the servlet. The repository stores the component specifications, which provide the keywords, which describe the services, and the location of the component. Following the keyword, the servlet can easily find the appropriate service and extract the URL location from the repository. The XML-based component specification makes it possible to dynamically register new services to the repository.

8. CONCULSION

The emergence of electronic commence, mobile computing, and embedded wireless systems create new requirements for Web-enabled applications. A particular requirement is to port existing legacy systems into the Web.

The multi-tier architecture provides numerous advantages for legacy system migration and integration with other applications. The key step is to specify the component configuration in XML, to provide rich information for the consequent automatic wrapping, integration and searching for the services provided by the legacy component. Currently, we are using for our prototypes VisualAge C++, Refine, Rigi and PBS [3], for reverse engineering, program analysis, and legacy system decomposition. Moreover, we have selected the IBM Visualage Enterprise Edition to develop the Enterprise JavaBeans application server [8], the IBM Websphere as the EJBs deployment environment, and the Visibroker as a CORBA implementation.

On-going work, with IBM Canada Center for Advanced Studies, focuses on the exploration of the appropriate

framework for the integration and dynamic service registration of services with the application server.

9. ACKNOLEDGEMENT

We would like to thank Richard Gregory and Kelvin Cheung for the valuable comments they provided on the preparation of this paper.

10. REFERENCES

[1] Brown, A., Barn, B., “Enterprise-Scale CBD: Building Complex Computer Systems From Components”, In Proceedings of STEP’99, Pittsburgh, PA. September, 1999.

[2] G. Canfora, G., A. Cimitile, A. DeLuccia, ``Decomposing Legacy Programs:A First Step Towards Migrating to Client-Server Platforms", In Proceedings of IEEE IWPC'98, June 1998.

[3] P. Finnigan et.al. “The Software Bookshelf”, IBM Systems Journal, Vol. 36, No. 4, November 1997.

[4] Gamma, E., Helm, R., Johnson, R., Vlissides, J., “Design Patterns: Elements of Reusable Object-Oriented Software”, Addison-Wesley, 1994.

[5] Held, J., Susch, C., Golshan, A., “What Does the Future Hold for Distributed Object Computing”, StandardView Vol. 6, No.1, March/1998.

[6] Hoque, R., “CORBA 3”, IDG Books Worldwide, Inc., 1998

[7] Kontogiannis, K., Martin, J., Wong, K., Gregory, R., Muller, H. and Mylopoulos, J., “Code Migration Through Transformations: An Experience Report”, In Proceedings of CASCON’98, Toronto ON., November 1998.

[8] Picon, J., Edwards, C., Scenini, G., “Using VisualAge for Java Enterprise Version 2 to Develop CORBA and EJB Applications”, International Technical Support Organization, http://www.redbooks.ibm.com.

[9] S. Tilley, et.al. "On Using the Web as Infrastructure for Reengineering", In Proceedings of IWPC ’97: Dearborn, MI; May 1997.

[10] Vogel, A., Rangarao, M., “Programming with hEnterprise JavaBeans, JTS and OTS: Building Distributed Transactions with Java and C++”, John Wiley & Sons, Inc, 1999.

[11] David Curtis, “Java, RMI and CORBA”, http://www.omg.org, 1997.

[12] Clemens Szyperski, “Component Software: Beyond Object-Oriented Programming”, Addison-Wesley, 1998.

[13] “CORBA and XML; Conflict or Cooperation?”, http://www.omg.org, 1999.

[14] Mark Elenko, Mike Reinertsen, “XML & CORBA”, http://www.omg.org, 1999.

[15] De Lucia, G.A. Di Lucca, A.R. Fasolino, P. Guerra, S. Petruzzelli, “Migrating Legacy Systems toward Object-Oriented Platforms”, 1997, IEEE.

[16] Prashant Patil, Ying Zou, Kostas Kontogiannis and John Mylopoulos, “Migration of Procedural Systems to Network-Centric Platforms”, CASCON’99, Toronto, 1999.

[17] Kostas Kontogiannis, Prashant Patil, “Evidence Driven Object Identification in Procedural Code”, STEP’99, Pittsburgh, Pennsylvania, 1999.

[18] David Krieger, and Richard M. Adler, “The Emergence of Distributed Component Platforms”, Computer, IEEE, March 1998.

[19] Cynthia Della Torre Cicalease, Shmuel Rotenstreich, “Behavioral Specification of Distributed Software Component Interfaces”, Computer, July 1999 IEEE.