Dynamic Architecture Reconstruction with Java 2, Enterprise Edition Platform

Keunpyo Lee

School of Computer Science

Carnegie Mellon University

2006

Submitted to School of Computer Science

in fulfillment of the requirements

for Undergraduate Research Program

ACKNOWLEDGEMENT

I am indebted to Professor David Garlan for his giving me an opportunity to start the first research experience, and to Dr. Bradley Schmerl for his help, supervision and feedback throughout the project. I would like to thank the authors of the tools in ABLE project group also.

ABSTRACT

Successful software system development depends on making good architectural decisions early in the design process, which are difficult to change and most critical to get right. In this architecture-based development, one of the challenging problems is to ensure that the implementation conforms to the architecture. In order to determine or enforce consistency between the architecture and its implementation, this research describes a technique that examines and monitors a system's dynamic runtime behavior. From the observation, in principle, we can infer its dynamic architecture which involves changes in the overall system configuration or the progress of the computation. Furthermore, by introducing a technique that maps low-level system events to high-level architectural representation using Colored Petri Nets, we can have a more accurate image of what is going on in the real system. This research covers dynamic architecture reconstruction for Java 2, Enterprise Edition platform based software systems to demonstrate the practical applicability of the techniques.

TABLE OF CONTENTS

1.1 MOTIVATION 1

1.2 PREVIOUS WORK 3

1.3 PROJECT AIMS 3

1.3.1 Monitoring runtime behavior of a distributed multi-tier system 3

1.3.2 Mapping runtime behavior in terms of architecturally meaningful event .. 4

1.3.3 Representing the resulting architecture in high-level C&C view type 4

2.1 DISCOTECT OVERVIEW 5

2.2 ACME 6

2.3 JAVA 2 ENTERPRISE EDITION PLTFORM 7

2.3.1 Overview of J2EE Architecture 7

2.3.1.1 Distributed Multi-tiered Applications 7

2.3.1.2 J2EE Containers 8

2.3.2 Enterprise JavaBeans (EJB 2.0) Introduction 9

2.3.2.1 Session Beans 9

2.3.2.2 Entity Beans 10

2.4 JBOSS APPLICATION SERVER 11

3.1.1 Duke's bank application 13

3.1.2 Probe Design 14

3.1.3 Linking libraries in interest with the Probe 15

3.1.4 Packaging and Deploying to JBoss Application Server 15

3.1.5 DiscoSTEP Specification 16

3.1.6 The Discovered Architecture 17

4.1 ACHIEVEMENT 20

4.2 HINDSIGHTS AND FUTURE WORKS 22

BIBLIOGRAPHY 28

LIST OF FIGURES

Figure 1. The DiscoTect Architecture 6

Figure 2. Multi-tiered Application 8

Figure 3. J2EE Server and Container 9

Figure 4. Life Cycle of a Session Bean 10

Figure 5. Life Cycle of a Entity Bean 11

Figure 6. Location of Application Server in the context of multi-tiered System 12

Figure 7. Documented Architectural View of Duke's Bank Application 14

Figure 8. EJB Package 16

Figure 9. CP-net Places, Arcs and Node functions translated from inputs and outputs 17

Figure 10. Discovered architecture of Duke's Bank for Web Clients 18

Figure 11. Discovered Architecture of Duke's Bank for Application Clients 19

Figure 12. Relation between EJB home proxy and EJB deployment 21

Figure 13. Path from a caller to callee inside EJB Container 22

1.0 INTRODUCTION

This chapter provides background information related to the project. The motivation behind the project is explained. A brief overview of previous work is given. The conclusion describes the aims of the project.

1.1 MOTIVATION

The software architecture of a software system is the structure or structure of the system, which comprise software elements, the externally visible properties of those elements, and the relationship among them [BCK03]. Recently as the size of a software system increases, the software architecture has received a considerable attention and emerged as a crucial sub-discipline of the design process. Even though the software architecture provides an abstract representation of a system, from the fact that a large system is inevitably partitioned in a number of different components and externally visible properties, the architecture's ability to provide easy understanding of all of the system's present and foreseeable components in cleanly bounded manner gives the direction to increase many necessary and desirable capabilities such compatibility, extensibility, reliability, and usability. Briefly, the software architecture is principles and guidelines governing their design and evolution over time [Garlan 95].

However, progressive changes to a software system result in a drift of the architecture of the system inevitably from the original architecture while implementation or maintenance without any effort to maintain the architectural documentation. The continuing changes and increasing complexity will cause declining quality and satisfactory in use unless the architecture is rigorously adapted and reviewed to take into account for changes in the operational environment.

Even though considerable research has designed and developed of more descriptive notations, tools and methods to support more concrete architecture design process [BCK03], inconsistent implementation with the architectural documentation produced by many well disciplined software engineers implies that the existing tools and notation do not serve well enough to ensure strong relationship between a system's software architecture and its implementation.

Normally, multi-tiered server-centric application has been considered difficult to develop because of many intricate features involved in the complex low-level details such as transaction, multi-threading, resource pooling and other enterprise-class services. However, over the past decade, considerable research has gone into development of many frameworks for developing complex multi-tier solutions, resulting in framework that can be rapidly deployed and easily enhanced [J2EE]. However, from the architectural perspective, the level of inconsistency between a system's software architecture and its implementation may increase in a distributed multi-tiered, server-centric dynamic software system for which composition of interacting components changes during run time [Garlan 97].

The validity of any analysis on architectural artifacts must be based on the conformance between architecture-as-built and architecture-as-designed. Some progress on this problem has been made to extract architecture in system's code level to ensure conformance. [Aldrich02, Jackson99, Kazman99, Murphy95].

These researches are based on static models derived from specifications like code level information to elicit architectural views. These techniques are adequate for steady-state behavior of the un-reconfigurable system. However, we need some adequate ways to capture and understand the reality of dynamic, reconfigurable software systems. The aim of this project is to provide a tool for collecting, disseminating and analyzing such dynamic aspects of architecture. In addition, the architecture and current status of the tool are presented and the system's use is illustrated through an example application in a distributed multi-tiered application system domain.

1.2 PREVIOUS WORK

The research is primarily related to discovering architectures for dynamic analysis of a distributed multi-tier software system. [AGS 04] provides an extensive overview of research area of determining the architecture of a system by examining its runtime behavior. The approach has a number of technical challenges Details concerning and various approaches to the challenges is given in Chapter 2 and only some general ideas about the application of the technique in the context of a large class of systems is given here.

There exist many low level event extraction techniques. [KC 99, Balzer99, Wells01, Dias03, Reiss03, Walker98, Walker00, and Zeller01] However, those previous works do not provide a way of specifying a series of correlated low-level events to architectural events. In principal, we can infer the runtime layout of a system by monitoring runtime behavior and relating this runtime observation to a high-level C&C (Component and Connector) view type architectural representation. [AGS 04] applies the techniques to several case studies to demonstrate mapping low-level runtime observation to architectural events, constructing C&C architectural view of the current implementation and hooking into existing architectural design tools to present, analyze and compare the runtime architecture of the system.

1.3 PROJECT AIMS

1.3.1 Monitoring runtime behavior of a distributed multi-tier system

The first general aim, as described in proposal, is to design and implement a probe that monitors a system's behavior and extracts low-level system event information, such as object creation, method invocation and others, in order to generate architectural events on the fly. Using the existing monitoring technology, AspectJ[Kieczales01], we illustrate how to inject code fragments into the target system to extract monitoring events.

1.3.2 Mapping runtime behavior in terms of architecturally meaningful event

The low-level system events from the probe are formatted as XML (Extensible Markup Language) string and placed on a JMS (Java Messaging Service) event bus to be consumed by the runtime mapping engine. However, finding mechanisms to bridge gaps between low-level system observations and architectural construct is a crucial capability needed to discover architecture of a system. By designing a collection of temporal state machines and allowing concurrent intermediate states using Colored PetriNets [Jensen94], then we can perform pattern matching against runtime events and recognize interleaved patterns of them.

1.3.3 Representing the resulting architecture in high-level C&C view type

A set of architectural events are outputted from the runtime mapping engine. The architecture builder (herein ACME) takes the events and dynamically constructs an architecture. The resulted architecture can be displayed to the user or processed to other analysis tools.

2.0 PREPARATION

This chapter presents and details a tool called DiscoTect (**Disco**vering Archi**Tect**ures) that addresses the concerns mentioned in constructing architecture dynamically from running systems. The information about the overview of DiscoTect is primarily adopted and extracted from [ASG 94].

2.1 DISCOTECT OVERVIEW

The tool adopted the approach illustrated in Figure 1. The steps the tool is applied are

- The probe is injected into a running system in forms of aspects or debuggers. The monitored events from the probe are firstly filtered by a tracing engine to select subset of the system events including method calls, object creation, value changes and so forth.
- The filtered low-level events are formatted as XML (Extensible Markup Language)[XML] streams and fed to a JMS (Java Messaging System)[J2EE] bus to be consumed by DiscoTect Engine.
- DiscoTect Engine is instructed how to interpret the low-level system events to corresponding architectural events. Such mapping specification is provided by DiscoSTEP (Discovering Structure Through Event Processing).
- The semantics of DiscoSTEP mapping is defined in Colored Petri Nets (CPN) which as a modeling language provides graphical structure of a system as a directed bipartite graph with annotations [Jensen94]. The full treatment of the formal semantics of DiscoSTEP mappings is not defined in this, but we refer to the reader to [AGS 04] for the full definition.

- DiscoSTEP specification fed into the DiscoTect Engine interprets the events and performs pattern matching to produce architectural events such as component and connector creation, attaching components, etc.
- Because many intermediate events such as object creation, lookup, library calls, etc. might be interleaved. The DiscoTect Engine constructs a state machine using Colored PetriNets[Jesen94] designed to recognize interleaved pattern of runtime events

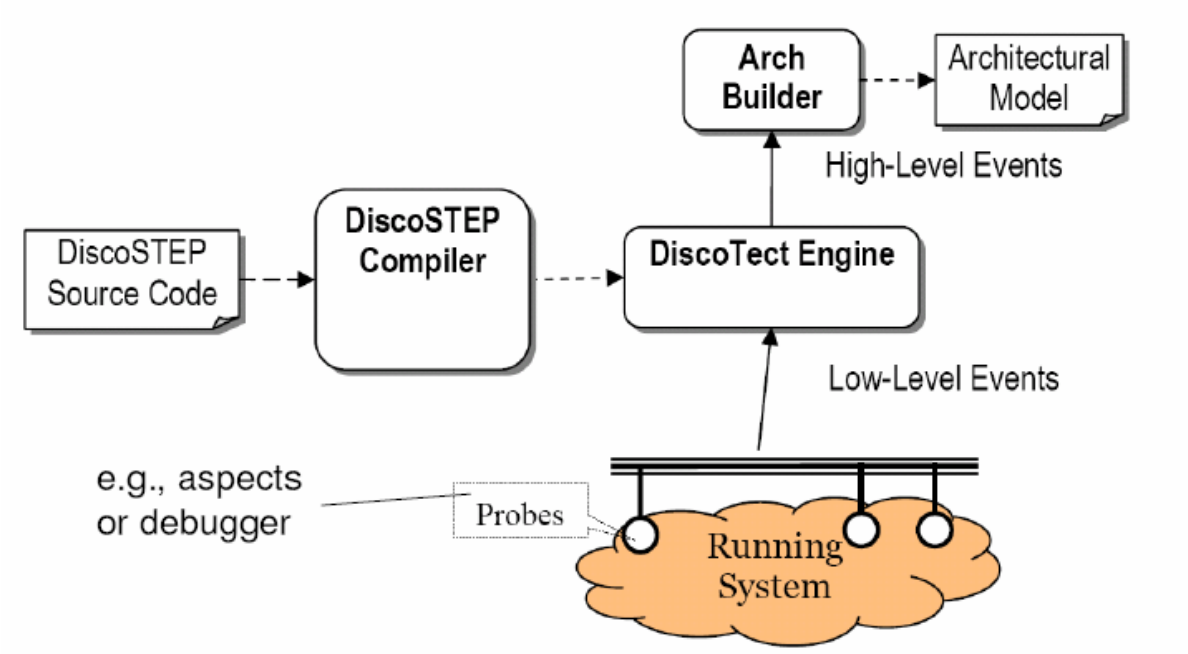


Figure 1. The DiscoTect Architecture

2.2 ACME

During initial consideration before writing the proposal, I was concerned that I had no prior practical experience with architectural description language (ADL). When I decided to attempt to use an ADL tool, I need some familiarity with the language, understanding of C&C view, and availability of connection between DiscoTect and the ADL tool. AcmeStudio [Schmerl04] is a simple, generic software architecture description language (ADL) that can be used as a common interchange format for architecture design tools. The ADL tool provides capability to display

runtime layout of a system relating runtime observation to a high-level C&C (Component and Connector) view type architectural representation, which may not be observable in the static artifact. It is implemented as a plug-in to the Eclipse [Eclipse05] environment, a framework for developing integrated development environments [Garlan00]. The architectural events produced by DiscoTect Engine are forwarded as XML format stream to AcmeStudio Remote Control plug-in communicating over Java RMI. The AcmeStudio [Schmerl04] incrementally constructs the corresponding architecture and display it for the user to analyze and check the architecture with respect to the expected architectural styles.

2.3 JAVA 2 ENTERPRISE EDITION PLTFORM

2.3.1 Overview of J2EE Architecture

Java 2 Platform, Enterprise Edition is a programming platform that provides environments in which we can easily develop and deploy distributed multi-tiered systems [J2EE]. For example, a system that uses middleware to service data request between a client and a database employs the multi-tier architecture. J2EE provides many types of interfaces and services including JDBC, RPC, CORBA, EJB, Servlets, JSP and several web services. J2EE also defines specification to coordinate and integrate the technologies with each other as well as with other legacy systems [BCK03]. From the dynamic architectural perspective, the J2EE™ is a desirable platform that the dynamic re-configuration behavior of architecture can be observed. By practicing the reverse engineering techniques we are concerning on the platform, we can have more dynamic architectural information.

2.3.1.1 Distributed Multi-tiered Applications

The J2EE platform employs distributed multi-tiered application model for class of enterprise applications. Application logic is divided into components according to function. The various application components that make up a J2EE application are installed on different machines

depending on the tier in the multi-tiered J2EE environment which the components belong to. Figure 2 is an example depicting two different types of multi-tiered application.

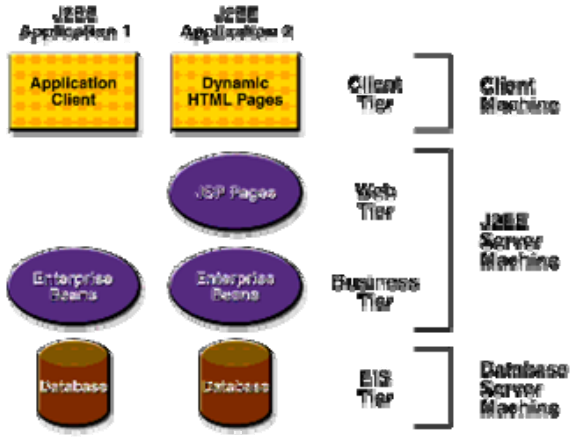


Figure 2. Multi-tiered Application

The J2EE architecture is composed of several components including

- Client-tier components run on the client machine
- Web-tier components run on the J2EE server
- Business-tier components run on the J2EE server
- Enterprise Information System (EIS) – tier software run on the EIS server

2.3.1.2 J2EE Containers

J2EE server provides many complex underlying low-level service details in the form of a container including transaction, state management, multi-threading, naming service, security, resource pooling, etc. This allows J2EE applications easy to write and maintain. J2EE container is the interface between a component and the low-level platform specific functionality that supports the component. Components are packaged into a J2EE module and deployed into its container. The Figure 3 illustrates container types.

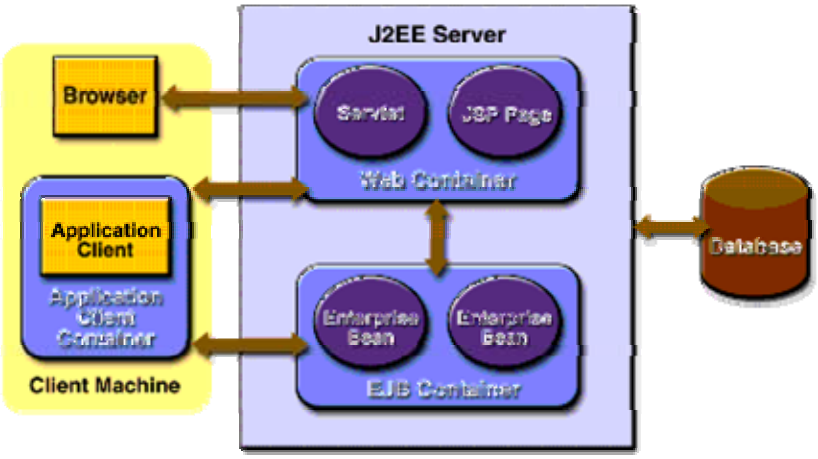


Figure 3. J2EE Server and Container

The details of each container are described and explained in [EJB]. Here we merely cover the containers used for the research.

- **J2EE Server** – runtime portion of a J2EE product, providing EJB and Web container
- **Enterprise JavaBeans (EJB) container** – manages the execution of enterprise beans for J2EE applications running on the J2EE server.

2.3.2 Enterprise JavaBeans (EJB 2.0) Introduction

J2EE platform employs JavaBeans component architecture to manage the data flow between an client tier and the components running on the server tier, or between server components and a database. JavaBeans component are not considered part of J2EE component by the J2EE specification. [EJB] outlines specification that JavaBeans components should conform to.

2.3.2.1 Session Beans

A session bean represents a single client inside the J2EE server. A client access the methods defined in the session bean to access an application deployed on the server. The session bean work on behalf of its client, abstracting the client from any complex details by executing necessary tasks inside the server. A session bean is not shareable with any other clients. When a client terminates, its session bean appears to terminate and is no longer associated with the client. There are two types of session beans, one is a stateless session bean and the other type is a stateful session bean. A stateless session beans does not maintain a conversational state for the

client and is persistent only for the duration of the client's invocation. The client's state is no longer retained. However, a stateful session bean represents a unique state of a client. Unlike a stateless session bean, the client's state is retained for the duration of the client-bean session.

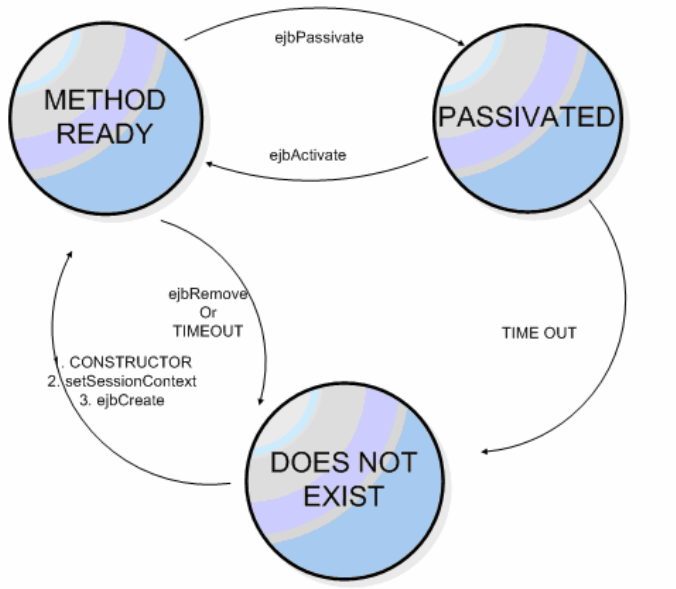


Figure 4. Life Cycle of a Session Bean

The life cycle of a session bean, which is illustrated in Figure 4, is very important in that the object instantiation must be monitored.

2.3.2.2 Entity Beans

An entity bean represents a business object in a persistent storage mechanism. Each instance of an entity bean corresponds to a row in the underlying table associated with the bean. Entity beans are different from session beans in that the entity beans are persistent, allow sharable access, have primary keys, and participate in relationship with other entity beans. Such persistence is supported in two ways, one is by the Container-Manager Persistence (CMP) and the other way is by the bean developer. For Bean-Managed Persistence, the bean developers themselves must implement logistics to provide the persistence.

The life cycle of entity beans are different from that of session beans in that they are not created or assigned to each client, but managed in an instance pool by the container. Unlike a session bean, ejbCreate() doesn't instantiate a new JAVA object but return a bean from the pool. Thus,

entity beans do not necessarily provide creation infrastructure. So the way to monitor how a entity bean is created must be treated differently from a session bean.

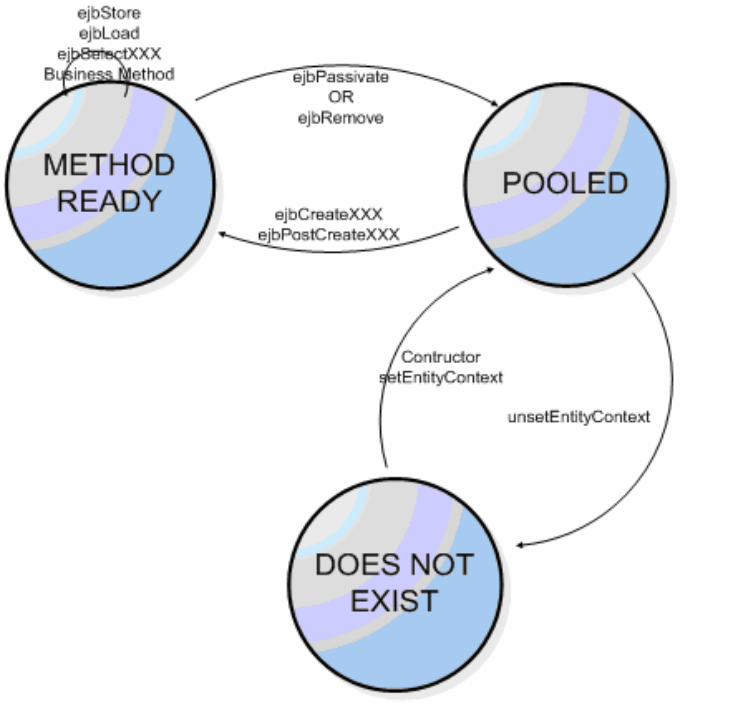


Figure 5. Life Cycle of a Entity Bean

2.4 JBOSS APPLICATION SERVER

An application server often refers to a component-based product that resides in the middle-tier of a server centric architecture. Main responsibilities of an application server are to provide middleware services for security and state maintenance, along with data access and persistence. The Figure 6 illustrates where application server resides on in the context of three-tiered architecture.

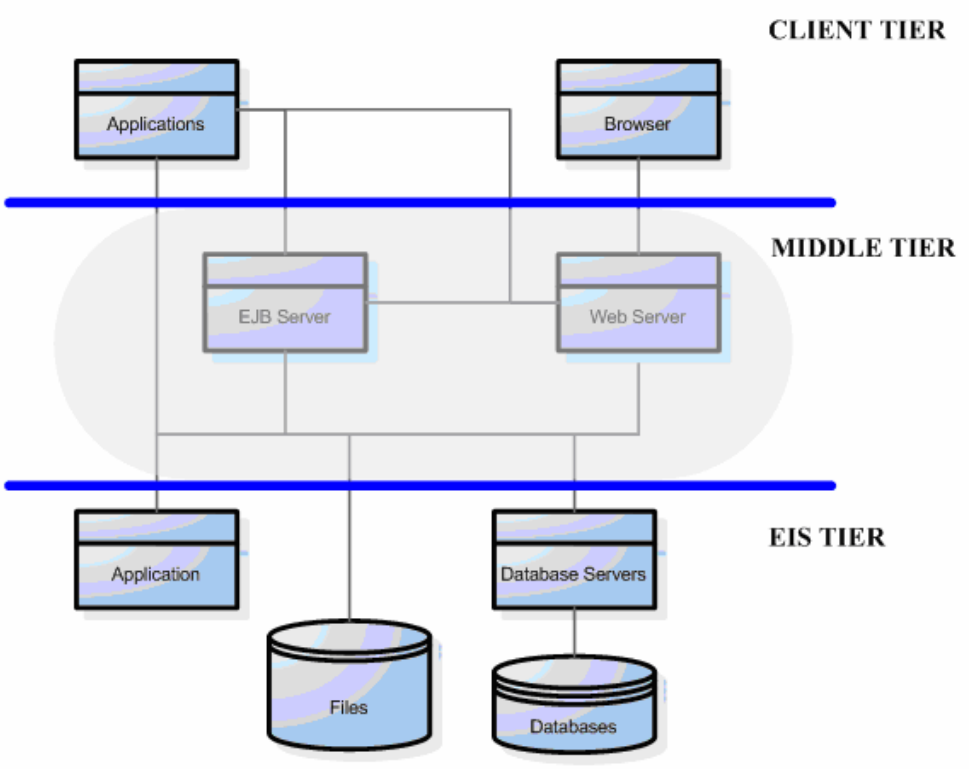


Figure 6. Location of Application Server in the context of multi-tiered System

The JBoss Application Server is a open source Java 2 Enterprise Edition (J2EE) application server. It is fully compliant to the J2EE 1.4 specification. We can notice that JBoss Application Server is modularly developed and implemented using component-based plug-ins. The modularization is supported by the Java Management Extension (JMX) API. The effort allows the server components and applications deployed on it easily. We refer the reader to [JBOSS] for more introduction and explanation about JMX industry-standard interfaces.

3.0 IMPLEMENTATION AND APPLICATION

This chapter details the application of architecture reconstruction techniques introduced in the previous chapter and outlines how the concepts were employed to apply them to reconstruct a Java 2, Enterprise Edition Platform based application from Sun Microsystems, Duke's Bank Application, implemented mainly in Java. The goal of the reconstruction effort is to implement and apply the reconstruction techniques to a system implemented in Java to depict the system's dynamic architecture providing what's actually going on. We used EJB architectural style that distinguishes participating components as entity beans, session beans, EJB container and database. The reconstruction process is consisted of the following steps

- Produce an aspect that monitors and filters system level events. The low level events we are interested in are the correlated interleaved events to be mapped to an architecturally significant event, for example, object interaction and creation through procedure calls and object instantiation.
- Write DiscoSTEP specification that is compiled and fed into the DiscoTect Engine for it to use in transforming a series of low level events to architectural events.
- Use AcmeStudio to describe EJB architecture style in C&C viewtype by creating relevant family and system.
- Open JMS connection from Acme Studio Remote module to accept architectural events in order to process the JMS message and use them to represent architecture on the system editor display.

3.1.1 Duke's bank application

Duke's Bank demonstrates a selection of J2EE technologies working together to implement a simple on-line banking application. It uses EJBs and web components (JSPs and servlets) and

uses a database to store the information. The persistence is bean-managed, with the entity beans containing the SQL statements which are used to manipulate the data. Upon the deployment on the application server, there are two types of clients that access the application,

- Application client used by administrators to manage and customers and account
- Web client used by customers to access account histories and perform transaction

Enterprise beans provide a way for the clients to access the customer, account and transaction information stored in database. The application is amalgamation of many of the component technologies including enterprise-beans, application clients, and web components. Figure 7 gives the layout of the architectural model how the components interact.

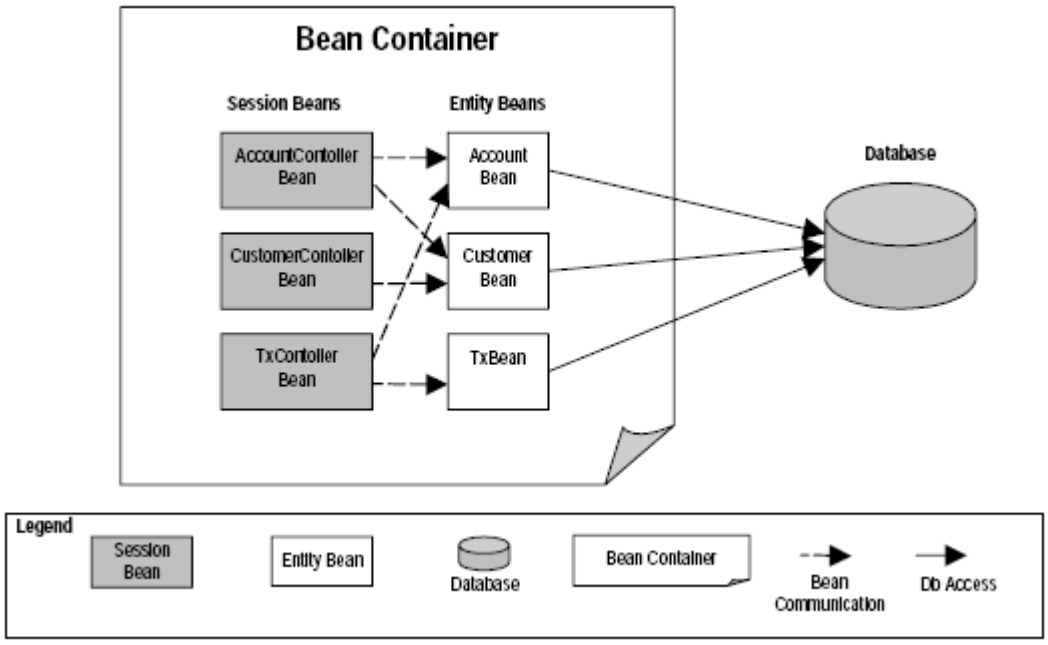


Figure 7. Documented Architectural View of Duke's Bank Application

The scope of the task is restricted to reconstruct structure and interrelationship among bean container, session beans, entity beans, and database.

3.1.2 Probe Design

One possible solution to trace all the method calls is to scatter logging function across the system implementation. But as discussed in the previous chapter, one possible result is that code is

tangled across a system and leads to quality and maintenance problem. Some aspects of system implementation such as profiling, logging, and feature variations are extremely difficult to implement in a modular way. [] We employ Aspect Oriented Programming paradigm to enable the clean modularization of the crosscutting concerns. AspectJ is Java platform compatible aspect-oriented extension. Before we discuss the design of the probe, some clarification of concepts that consist of AspectJ is necessary. We refer the reader to [Kieczales01] for the detail concept and understanding of Aspect-Oriented Programming and AspectJ.

- Join Point is a well defined point in the execution of the program
- Pointcut picks out certain joint points in the program flow
- Advice brings together a pointcut and a body of code to implement actual crosscutting concerns.

3.1.3 Linking libraries in interest with the Probe

After defining the aspect for probing a existing system, we need to weave the system with the probe. It's possible to weave at compile time by feeding provided jar files (e.g. jboss.jar, bank-ejb.jar, etc.) as inpath argument to the AspectJ compiler.

3.1.4 Packaging and Deploying to JBoss Application Server

Now that the we have the JBoss Application Server running, we have to prepare to deploy the online banking application on server. There are four steps

- Package JAR file that contains the compiled and weaved classes and deployment descriptors for the entity beans and associated controller session beans which the client interacts with.
- Package WAR file that contains web source (JSPs, images, etc.) to provide the front end to allow users to interact with the EJB module.
- Package Application Client for Java client for administering customers and accounts.
- Assemble the EAR file, which is the complete application containing the three modules packaged. The assembled EAR file is copied to JBoss server deployment directory to provide banking service.

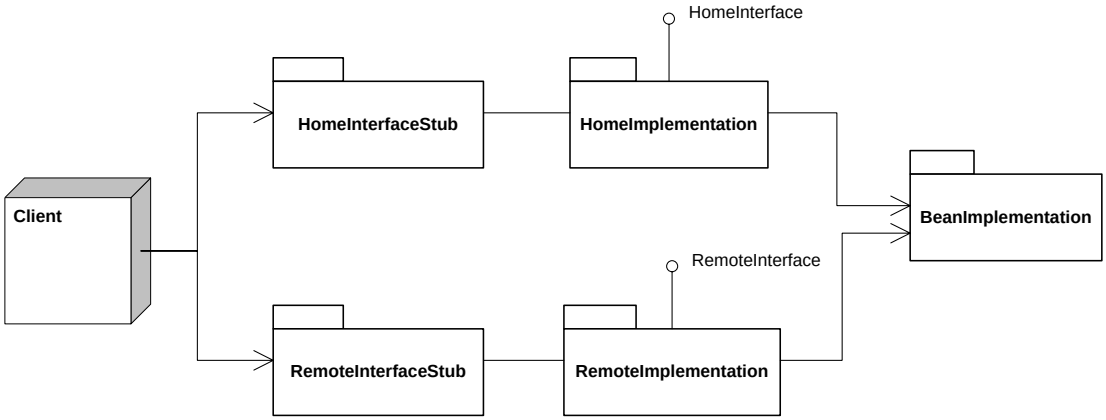


Figure 8. EJB Package

3.1.5 DiscoSTEP Specification

The Duke's Bank application DiscoSTEP program uses the following types: string, inti, call, holder, connection-holder, create_component, and create_connector. We derive the color sets for the CP-net as:

∑= {string, init, call, create_component, create_connector}

The following corresponding CP-net transitions are obtained from the rules.

T={CreateSystem, StartBean, CreateDB, CreateBean, RecordDBConnection, ConstructReadPort, ConstructWritePort, ContructDBPort, InvokeInterface, InvokeProxy, RecordConnection, AttachComponents}

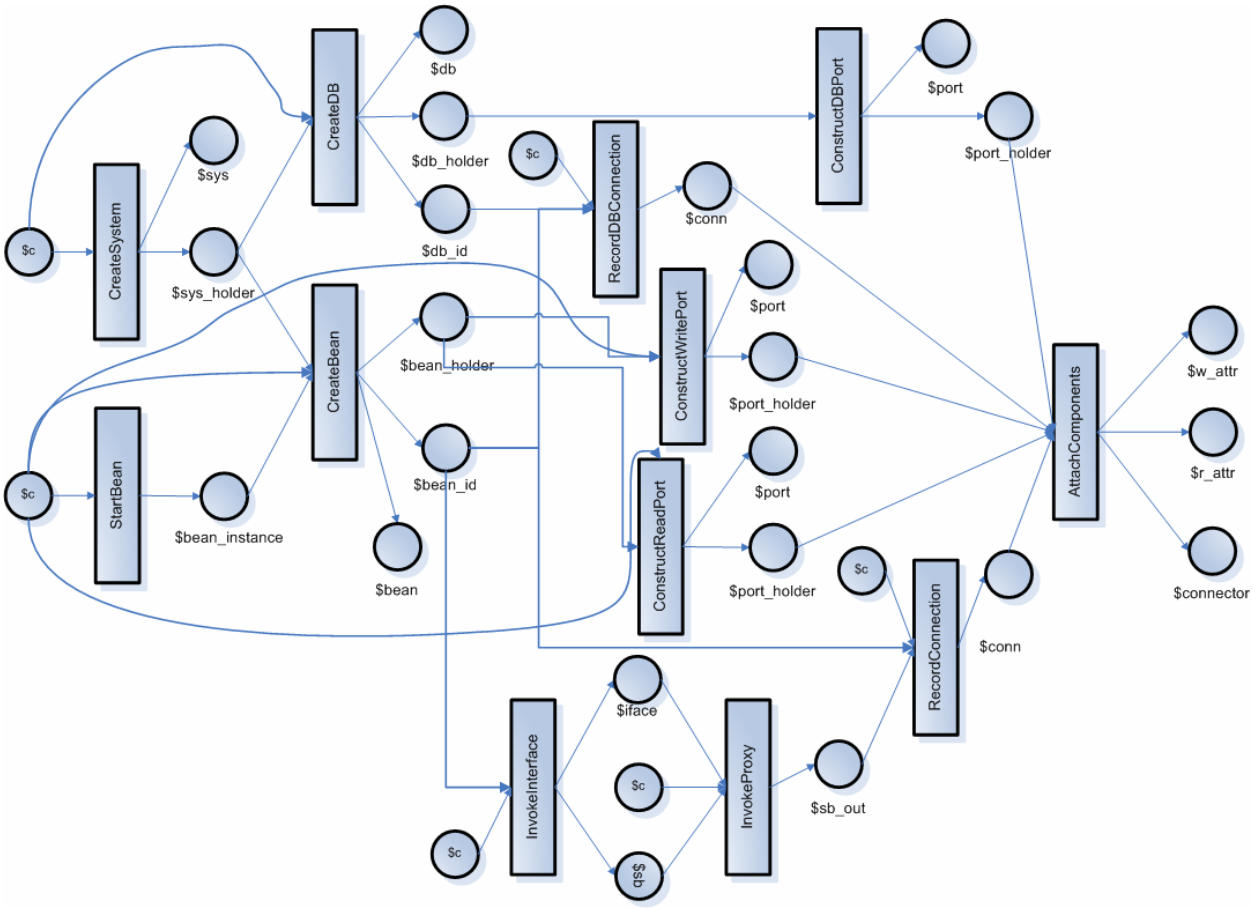


Figure 9. CP-net Places, Arcs and Node functions translated from inputs and outputs

3.1.6 The Discovered Architecture

Applying all the techniques introduced to reconstruct architecture on the banking system yield the architecture model in Figure 10. During the reconstruction effort, several architectural views were generated through architectural events abstracted from the low-level information extracted from the system. The Acme Studio tool provides the ability to visualize, navigate, and manipulate the set of components and connectors generated. We have organized the layout of this model for better comprehensibility. Figure 11 illustrates dynamic state of EJB container when a client is accessing the service through web application.

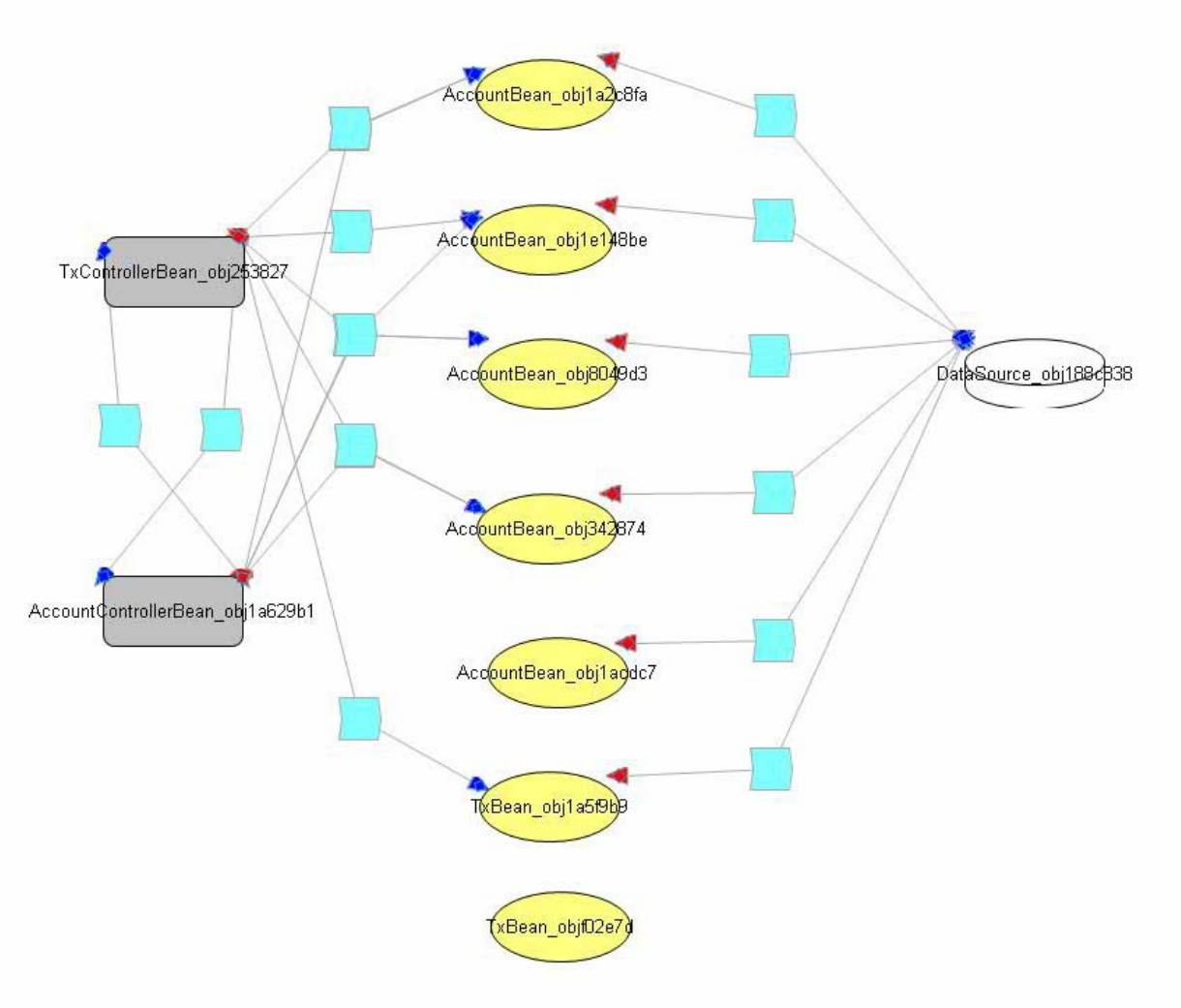


Figure 10. Discovered architecture of Duke's Bank for Web Clients

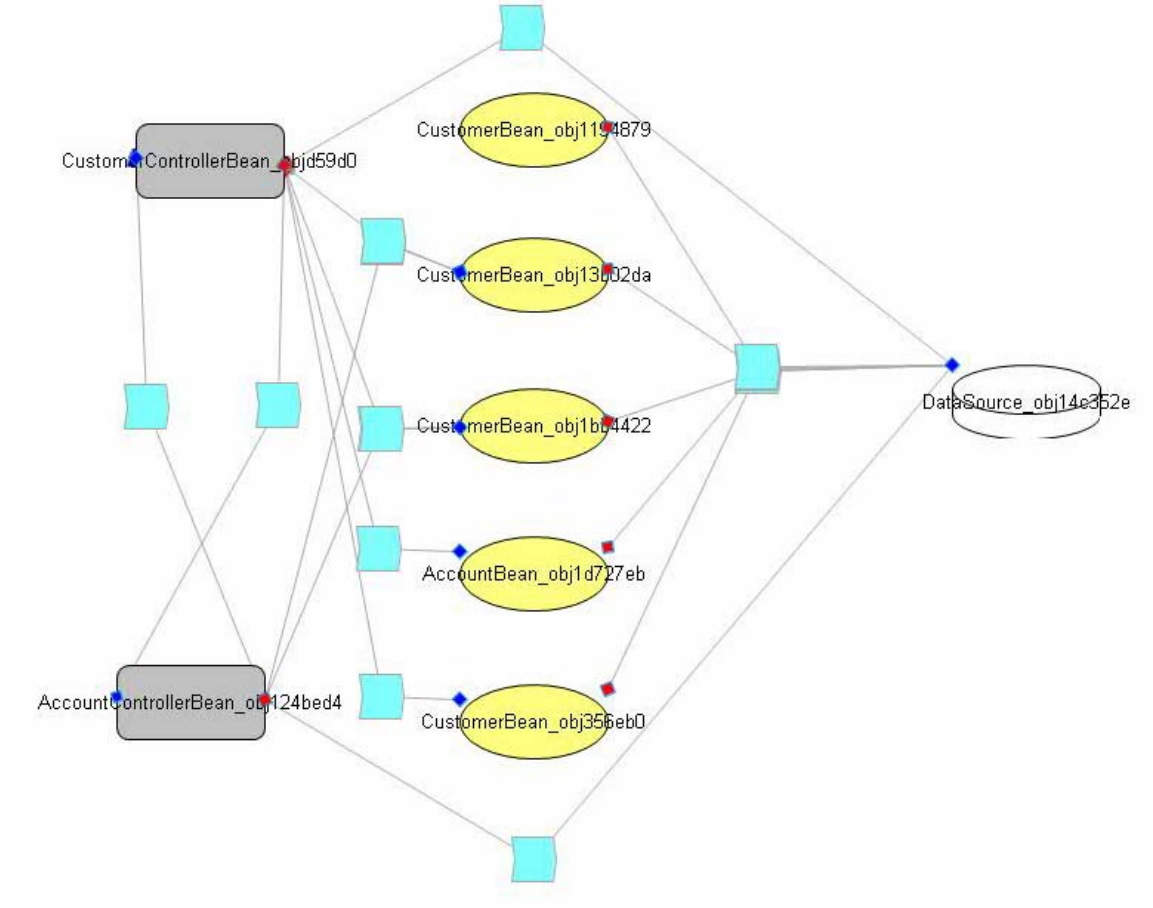


Figure 11. Discovered Architecture of Duke's Bank for Application Clients

By taking a close look at the access paths between the session beans and entity beans within the enterprise bean tier, we can see that the session beans are clients of the entity beans. On the backend of the application, entity beans access the database. But the actual situation is different from this original architecture. Session beans even access the database to carry out some necessary tasks such as executing transaction, managing database tables, etc

4.0 CONCLUSION

4.1 ACHIEVEMENT

The prime motivation behind this project was to apply the Architecture Reconstruction techniques developed by [ACG 04] to Duke's Bank Application. All aims outlined in the introduction were met. However, the other aims discussed in the project proposal were not able to be carried out. A number of what were originally project extensions in the proposal, including CORBA distributed object-oriented system, graphical simulation, were not implemented.

A specification for mapping low-level system events to architecturally significant event for the banking system was designed, implemented, compiled and tested. One of the main benefits of this approach is that the mapping specification can be reused in other systems that adopt the same architectural design patterns which prescribe specific abstraction of data, function and interrelationship to provide a form of codified solutions to recurring design problems [Kazman 00]. It allows for flexible reusability for other J2EE™ platform based applications. Its specification can be extended in a straight-forward way to incorporate different features of J2EE as well as other distributed multi-tiered object oriented applications.

It is much easier to track a flow of method calls in a Java software system that consisting components are residing on one JVM. If the components are distributed and the connectors between any two components provides black-box perspective regardless of what methods and actions are taking place underneath. JBoss application server, the EJB container we used for deploying the banking system, provides simple client view of an EJB through the home and remote proxies [JBOSS]. A client never references an EJB bean instance directory but through the interfaces exposed. However, tracking the method propagation to the designated entity beans

was very tricky as illustrated in Figure. A method call from a client passes all through every interceptor and eventually arrives at the entity bean that takes care of the task.

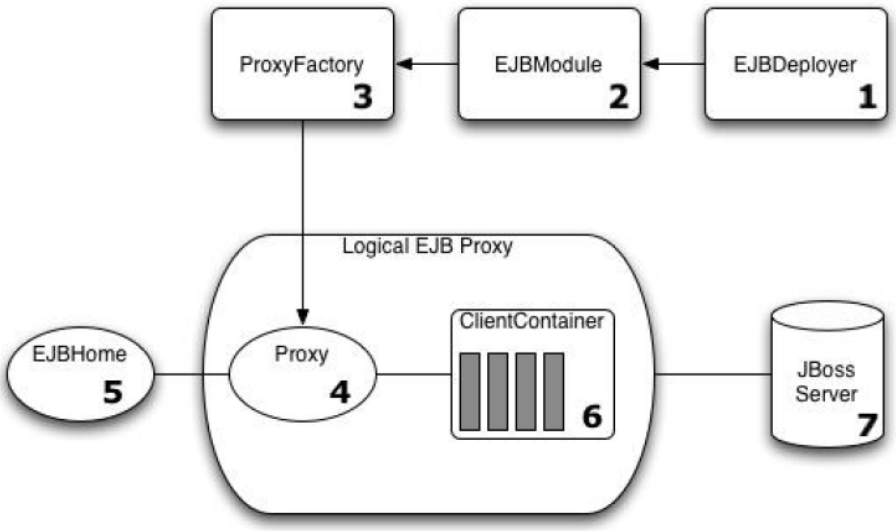


Figure 12. Relation between EJB home proxy and EJB deployment

Human trials were required to recognize the method call process by taking a look at the traces that the Probe emitted. Sometimes the great amount of traces generated by the Probe made a system frozen, throwing “Heap Memory Insufficient Exception”. Appendix contains a “Aspect” that filters unnecessary traces for faster patter matching against the define Colored Petri Net state machine.

Serious amounts of times have been invested to figure out a simpler path from callers to callees, I could found the path as illustrated in Figure 13.

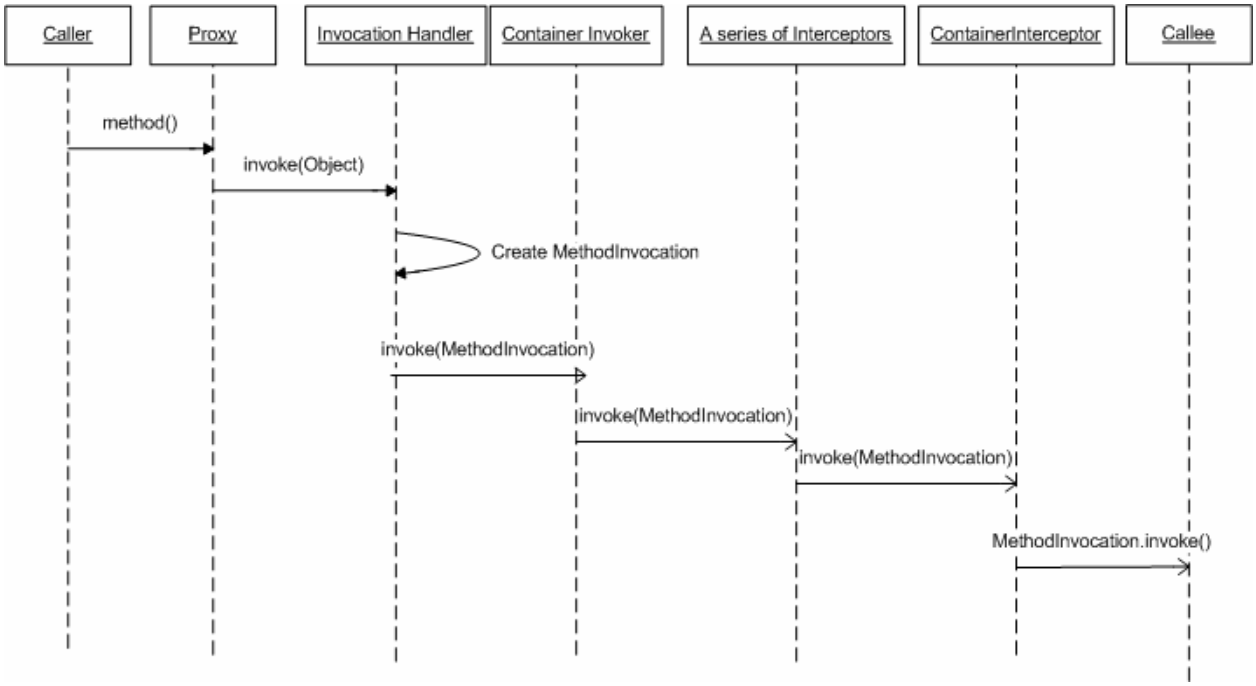


Figure 13. Path from Caller to Callee inside EJB container

4.2 HINDSIGHTS AND FUTURE WORKS

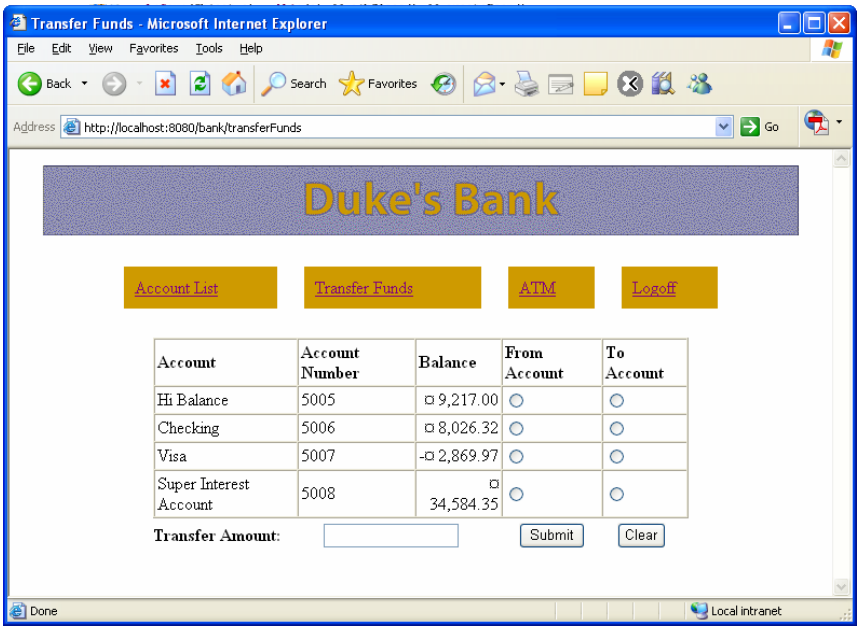
If I was to start this project again, I would be rather more prepared for the available technologies of software architecture. The architecture of the tools consists of several technologies including a selection of J2EE framework, DiscoSTEP mapping libraries, and Architecture Description Language (ADL). Because errors or exceptions occurred in one module propagate to all others, it is highly required for a developers to be aware of the flow and the architecture of the tool.

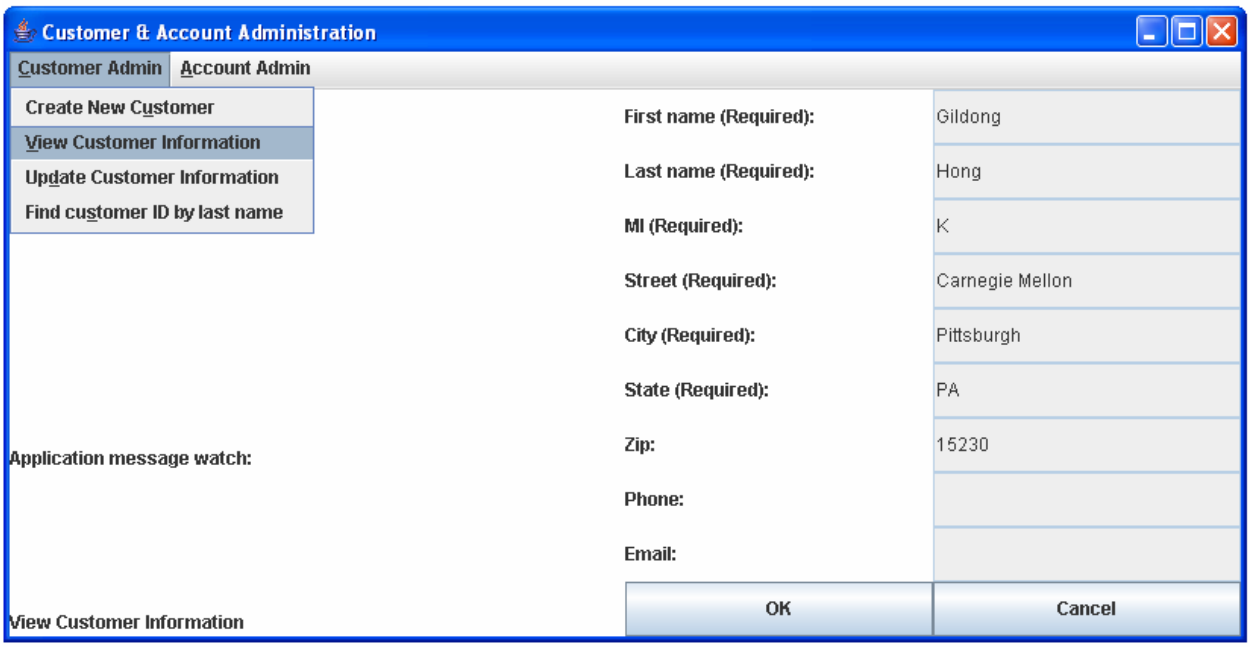
APPENDIX A - PROBE IMPLEMENTATION IN ASPECTJ

build.xml | *J2EEAspect.aj | J2EE.epp

```
//import com.sun.ebank.util.CodedNames;

/**
* @author kpl
*
*/
public aspect J2EEAspect extends JMSXMLEventEmitter {

    private static boolean isInitialized = false;

    pointcut deployerClass() : within(org.jboss.deployment..*);
    pointcut ejbClass() : within(com.sun.ebank.ejb..*);
    pointcut jbossClass() : within(org.jboss.ejb..*);
    pointcut jdbcClass() : within(javax.sql..*);
    pointcut thisAspect() : within(edu.cmu.cs.kpl.aspect..*);
    pointcut ejbContainer() : within(org.jboss.ejb.*Container);

    pointcut classes() : ejbClass() || jdbcClass() || jbossClass() || ejbContainer();

    pointcut createBean() : execution(* ejbCreate(..)) || execution(* ejbActivate(..))
                        || execution(* set*Context(..));
    pointcut allCalls() :   call(* *(..));
    pointcut allExecs() : execution (* *(..));
    pointcut notHashCode() : (!execution(* hashCode(..)) && !call(* hashCode(..)));
    pointcut notToString() : (!execution(* toString(..)) && !call(* toString(..)));

    pointcut methods() :
        (allExecs() || allCalls()) && notHashCode() && notToString() && !createBean() &&
        !execution(* addLocalHome(..)) && !execution(* access*(..)) && !cflow(adviceexecution());

    pointcut init() : execution(* init(..));

    before(Object obj, Object c) : this(c) && args(obj) && init() && deployerClass()
    {
        if (obj.toString().contains("JBossDukesBank")) {
            J2EEAspect.isInitialized = true;
            emitCallEvent(thisJoinPoint, c, "call");
        }
    }

    before(Object c) : this(c) && ejbClass() && createBean()
    {
        emitInit(thisJoinPoint, c);
    }

    before(Object c) : this(c) && classes() && methods() && if(J2EEAspect.isInitialized)
    {
        String method = thisJoinPoint.toString();
        if(method.contains("Proxy") ||
        method.contains("invoke") ||
        method.contains("Invocation") ||
        method.contains("Account") ||
        method.contains("Customer") ||
        method.contains("Tx") ||
        method.contains("DataSource"))
        {
          emitCallEvent(thisJoinPoint, c, "call");
        }
    }
```

APPENDIX B – TWO WAYS TO ACCESS DUKE'S BANK SYSTEM



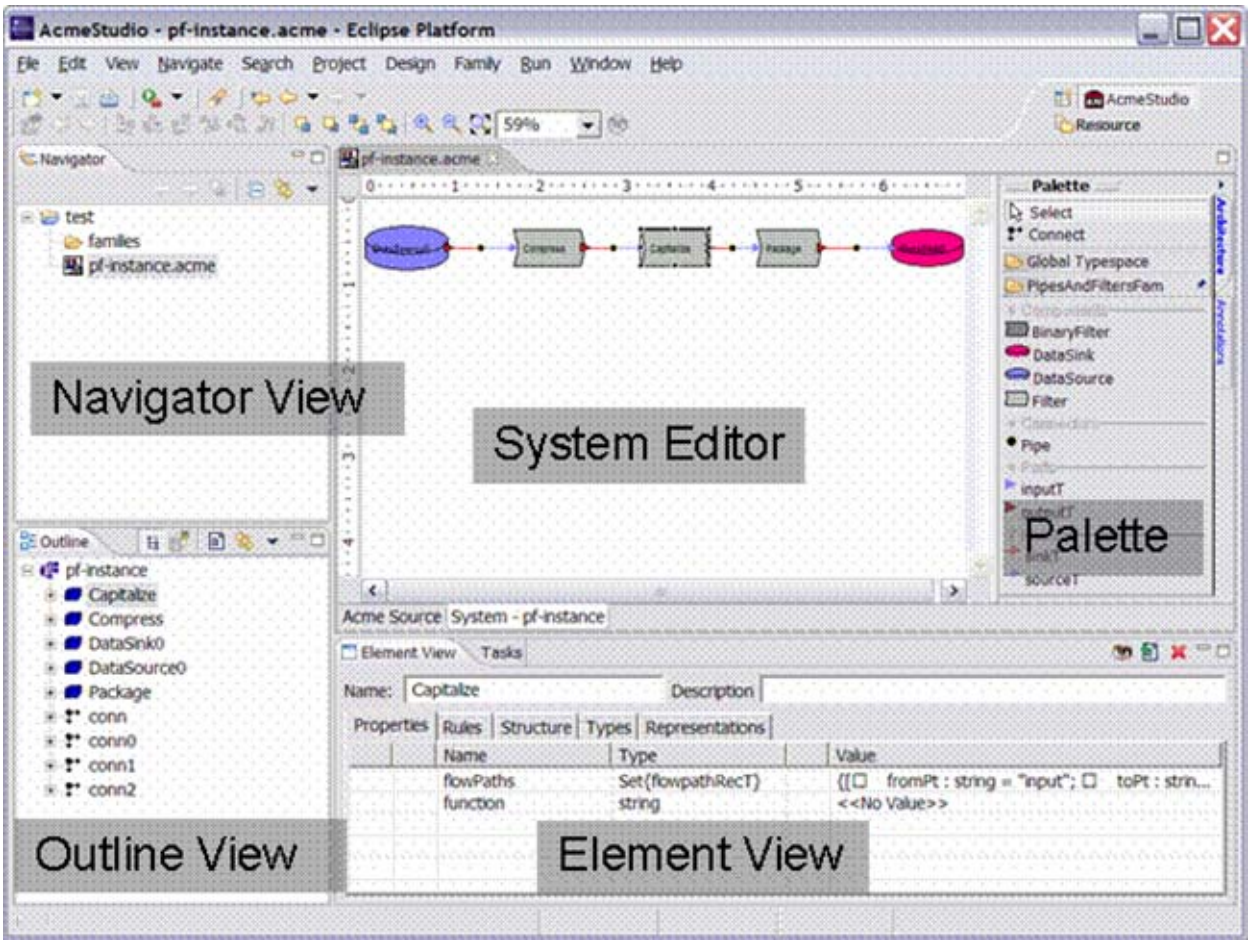


APPENDIX C – SAMPLE SYSTEM LEVEL EVENT TRACES

```
<!-- Fri May 05 20:57:54 EDT 2006 -->
<call method_name="org.jboss.deployment.EARDeployer.init" callee="obj107108e" caller="obj107108e"
timestamp="1146877096984">
<arg name="di" value="obje9f0a54a"/></call>
<call method_name="org.jboss.mx.util.MBeanProxyExt.create" caller="obj13803ba"
timestamp="1146877098328">
<arg name="arg0" value="obj14e3f41"/><arg name="arg1" value="obj6b588f95"/><arg name="arg2"
value="obj78a212"/></call>
<call method_name="org.jboss.mx.util.MBeanProxyExt.create" caller="obj13803ba"
timestamp="1146877098437">
<arg name="arg0" value="obj5f33b8"/><arg name="arg1" value="obje723408c"/></call>
<call method_name="org.jboss.metadata.BeanMetaData.get" callee="objd006a7" caller="obj13803ba"
timestamp="1146877098453">
<origMethod name="org.jboss.metadata.BeanMetaData.getSecurityProxy"/>
</call>
<call method_name="org.jboss.ejb.plugins.TxInterceptorCMT.importXml" callee="obj1280c85"
caller="obj1280c85"  timestamp="1146877098890">
<arg name="ielement" value="obj4e229e"/></call>
<call method_name="org.jboss.ejb.Container.get" callee="objlab3cda" caller="obj13803ba"
timestamp="1146877098953">
<origMethod name="org.jboss.ejb.Container.getSecurityProxy"/>
</call>
<call method_name="org.jboss.ejb.Container.get" callee="objlab3cda" caller="objlab3cda"
timestamp="1146877098953">
<origMethod name="org.jboss.ejb.Container.getSecurityProxy"/>
</call>
<call method_name="org.jboss.mx.util.MBeanProxyExt.create" caller="obj13803ba"
timestamp="1146877099046">
<arg name="arg0" value="obj5f33b8"/><arg name="arg1" value="obje723408c"/></call>
<call method_name="org.jboss.metadata.BeanMetaData.get" callee="objle3elb8" caller="obj13803ba"
timestamp="1146877099062">
<origMethod name="org.jboss.metadata.BeanMetaData.getSecurityProxy"/>
</call>
<call method_name="org.jboss.ejb.plugins.TxInterceptorCMT.importXml" callee="objlf8lefb"
caller="objlf8lefb"  timestamp="1146877099062">
<arg name="ielement" value="obj4e229e"/></call>
<call method_name="org.jboss.ejb.Container.get" callee="objc916lb" caller="obj13803ba"
timestamp="1146877099062">
<origMethod name="org.jboss.ejb.Container.getSecurityProxy"/>
</call>
<call method_name="org.jboss.ejb.Container.get" callee="objc916lb" caller="objc916lb"
timestamp="1146877099062">
<origMethod name="org.jboss.ejb.Container.getSecurityProxy"/>
</call>
<call method_name="org.jboss.mx.util.MBeanProxyExt.create" caller="obj13803ba"
timestamp="1146877099078">
<arg name="arg0" value="obj5f33b8"/><arg name="arg1" value="obje723408c"/></call>
<call method_name="org.jboss.metadata.BeanMetaData.get" callee="obj10fba68" caller="obj13803ba"
timestamp="1146877099078">
<origMethod name="org.jboss.metadata.BeanMetaData.getSecurityProxy"/>
</call>
<call method_name="org.jboss.ejb.plugins.TxInterceptorCMT.importXml" callee="obj1970991"
caller="obj1970991"  timestamp="1146877099109">
<arg name="ielement" value="obj4e229e"/></call>
<call method_name="org.jboss.ejb.Container.get" callee="objf53870" caller="obj13803ba"
timestamp="1146877099109">
<origMethod name="org.jboss.ejb.Container.getSecurityProxy"/>
</call>
<call method_name="org.jboss.ejb.Container.get" callee="objf53870" caller="objf53870"
timestamp="1146877099109">
<origMethod name="org.jboss.ejb.Container.getSecurityProxy"/>
</call>
<call method_name="org.jboss.metadata.SessionMetaData.isContainerManagedTx" callee="objlcle333"
caller="obj13803ba"  timestamp="1146877099140">
```

APPENDIX D – ACHITECTURAL EVENTS FORMATTES AS XML

APPENDIX E – ACMESTUDIO SCREEN CAPTURE



BIBLIOGRAPHY

[Aldrich02] J.Aldrich, C. Chambers, and D. Notkin. "ArchJava: Connecting Software Archietecture to Implementation," In Proceedings of the 24th International Conference on Software Engineering

[AGS04] H. Yan, D. Garlan, B. Schmerl, J. Aldrich, R. Kazman. Discovering Architectures from Running Systems using Colored Petri Nets.

[Balzer 99] Balzer, R. M. & Goldman, N. M. "Mediating Connectors," 73-77. Proceedings of 19th IEEE Conference on Distributed Computing Systems. Workshop on Electronic Commerce and Web-Based Applications. Austin, TX, May 31-June 4, 1999. Los Alamitos, CA: IEEE Computer Society, 1999.

[BCK03] Bass, L.; Clements, P.; & Kazman, R. Software Architecture in Practice, Second edition. Boston, MA: Addison-Wesley, 2003.

[Dias03] M. Dias and D. Richardson. "The Role of Event Description on Architecting Dependable Systems (extended version from WADS)." Lecture Notes in Computer Science – Book on Architecting Dependable Systems(Spring-verlag), 2003

[Eclipse 05] Eclipse. Aspectj project. http://eclipse.org/aspectj/. (URL valid as of May 2006)

[EJB] Sun Microsystems, EJB http://java.sun.com/products/ejb/docs.html (2005).

[Garlan96] Shaw, M., and Garlan, D. Software Architecture: Perspectives on an Emerging Discipline. Prentice-Hall 1996.

[Garlan97] Robert J. Allen, Remi Douence, and David Garlan, Specifying Dynamism in Software Architectures, Proceedings of the Workshop on Foundations of Component-Based Software Engineering, September 1997.

[Garlan00] Garlan, D.; Monroe, R. T.; & Wile, D. "Acme: Architectural Description of Component-Based Systems," 47-68. Foundations of Component-Based Systems (Edited by Gary T. Leavens & Murali Sitaraman). New York, NY: Cambridge University Press, 2000.30 CMU/SEI-2004-TR-016

[Garlan 02] Garlan, D.; Kompanek, A. J.; & Cheng, S.-W. “Reconciling the Needs of Architectural Description with Object Modeling Notations.” Science of Computer Programming 44, 1 (July 2002): 23-49.

[Jackson 99] Jackson, D. & Waingold, A. “Lightweight Extraction of Object Models from Bytecode,” 194-202. Proceedings of the 21st International Conference on Software Engineering. Los Angeles, CA, May 16-22, 1999. New York, NY: Association of Computing Machinery, 1999.

[Jensen94] K. Jensen. Coloured Petri Nets: A High-level Language for System Design and Analysis. In Advances in Petri Nets 1990. LNCS 483, G. Rozenberg (Ed), 1991.

[J2EE] Sun Microsystems, Inc. The J2EE Tutorial, Second Edition. http://java.sun.com/docs/books/j2eetutorial/index.html

[KC 99] Kazman, R. & Carriere, S. J. “Playing Detective: Reconstructing Software Architecture from Available Evidence.” Journal of Automated Software Engineering 6, 2 (April 1999): 107-138.

[Kiczales 01] Kiczales, G.; Hilsdale, E.; Hugunin, J.; Kersten, M.; Palm, J.; &Griswold, W. G. “An Overview of Aspect J,” 327-353. Proceedings of ECOOP 2001–Object-Oriented Programming, 15th European Conference. Budapest, Hungary, June 18-22, 2001. Berlin, Germany: Springer-Verlag, 2001.

[Murphy 95] Murphy, G. C.; Notkin, D.; & Sullivan, K. J. “Software Reflexion Models: Bridging the Gap Between Source and High-Level Models,” 18-28. Proceedings of the Third ACM SIGSOFT Symposium on the Foundations of Software Engineering. Washington, DC, October 10-13, 1995. New York, NY: Association for Computing Machinery, 1995.

[Reiss 03] Reiss, S. “JIVE: Visualizing Java in Action Demonstration Description,” 820-821. Proceedings of the International Conference on Software Engineering. Portland, OR, May 3-10, 2003. Los Alamitos, CA: IEEE Computer Society, 2003.

[Schmerl04] AcmeStudio: Supporting Style-Centered Architecture Development International Conference on Software Engineering archive Proceedings of the 26th International Conference on Software Engineering

[XML] W3C<http://www.w3.org/XML/>

[Walker 98] Walker, R. J.; Murphy, G. C.; Freeman-Benson, B.; Wright, D.; Swanson, D.; & Isaak, J. “Visualizing Dynamic Software System Information Through High-Level Models,” 271-283. Proceedings of OOPSLA’98: Conference on Object-Oriented Programming, Systems, and Applications. Vancouver, BC, October 18-22, 1998. New York, NY: Association for Computing Machinery, 1998.

[Walker 00] Walker, R. J.; Murphy, G. C.; Steinbok, J.; & Robillard. M. P. “Efficient Mapping of Software System Traces to Architectural Views,” 31-40. Proceedings of

CASCON 2000 (Edited by S. A. MacKay & J. H. Johnson). Mississauga, Ontario, Canada, November 13-16, 2000. Toronto, Ontario, Canada: IBM Canada, Ltd., 2001.

[Wells 01] Wells, D. & Pazandak, P. "Taming Cyber Incognito: Surveying Dynamic/ Reconfigurable Software Landscapes." Proceedings of the 1st Working Conference on Complex and Dynamic Systems Architectures. Brisbane, Australia, December 12-14, 2001. Brisbane, Australia: Distributed Systems Technology Center at the University of Queensland, 2001.

[Zeller 01] Zeller, A. "Animating Data Structures in DDD," 69-78. Proceedings of the First International Program Visualization Workshop. Finland, July 7-8, 2000. Porvoo, Finland: University of Joensuu, 2001.