Support for Maintaining Distributed Component-based Systems

Xuefen Fang
SRA Key Technology Laboratory, Inc. &
Dept. of Graphics and Computer Science
Graduate School of Arts and Sciences
The University of Tokyo
fang@graco.c.u-tokyo.ac.jp

Tetsuo Tamai
Dept. of Graphics and Computer Science
Graduate School of Arts and Sciences
The University of Tokyo
tamai@graco.c.u-tokyo.ac.jp

ABSTRACT

Legacy systems are subject to successive changes throughout their working lives, which diminish their understandability. No popular system can escape the fate of becoming a legacy in this sense, as its age advances. Relatively recently, distributed component-based application systems are getting more and more in service, and we regard it important to prepare for their future when they start to show the typical malaise as legacies. A distributed component-based system is typically constructed by implementing a component that encapsulates an application logic, and by making it run with existing components in a pre-defined execution environment. Developed under a tight schedule, with a high employee turnover, and in a rapidly evolving environment, such a system is hard to maintain from the very beginning.

This paper surveys the architecture views proposed for maintenance works, and presents an approach to reconstruct such views of distributed component-based systems. Our approach is applicable to various technologies used in developing those systems. In this approach, concrete structures of distributed component-based systems are extracted from conceptual components, and interactions between various components, such as distributed objects and databases, are shown.

Keywords

system undesrstanding, architecture reconstruction, maintenance, distributed component-based system

Introduction

A distributed component-based application is a software system whose functionality is delivered through Internet services of various servers, such as web servers and application servsers. With the advent of the Internet, web technologies, and component technologies, many applications are no longer developed using traditional client/server technologies. They are developed, instead, with web browsers, web servers, and application servers. Their functionality is implemented through user interfaces, distributed component objects, and databases. The interactions between them are important to software developers when they have to maintain or enhance those applications.

Our objective is to ease this maintenance task, and this paper strives to achieve this goal in two ways. Firstly, it gives a survey of software architectures that are particularly relevant to understanding distributed component-based systems. Secondly, the paper presents an approach for reconstructing the architectures of such systems. In this approach, structures of distributed component-based systems are extracted from conceptual components, and interactions between various components, such as distributed objects and databases, are shown.

The reconstructing process uses a set of specialized extractors and a dynamic tracer. The extractors analyzes the resources of applications. The tracer derives information on systems' execution. The data obtained by the extractors and the tracer can be shown in individual diagrams, or can be merged to generate architecture diagrams. In order to reduce the complexity of the architecture diagrams, a clustering technique is used. These diagrams help developers gain a better understanding of applications, which ease their maintenance works.

The rest of the paper is organized as follows. Section 2 describes the architecture views that are needed to by developers to gain a better understanding of distributed component-based applications. Section 3 describes our architecture reconstructing approach for distributed component-based applications. Section 4 presents a case study on the E-Process system and its reconstructed architecture. Finally, Section 5 draws conclusions from this work.

Architecture Views for Maintenance

Software system mainteance and enhancement require a well-understool architecture, especially if the system is large and complex[7]. Traditional approaches[8, 9, 5] are mostly based on functional definitions, function references, variable definitions, and variable uses. This level of granularity is not appropriate to distributed component-based application systems. Usually, such a system is composed of separable, interaction components, where major pieces of the system are not under the control of the source code. Required here are architectures at the component level.

In our previous analytical work on software architectures for understanding distributed component-based systems[4], we have found four kinds of relevant architecture views. Each of the four architecture views addresses different engineer-

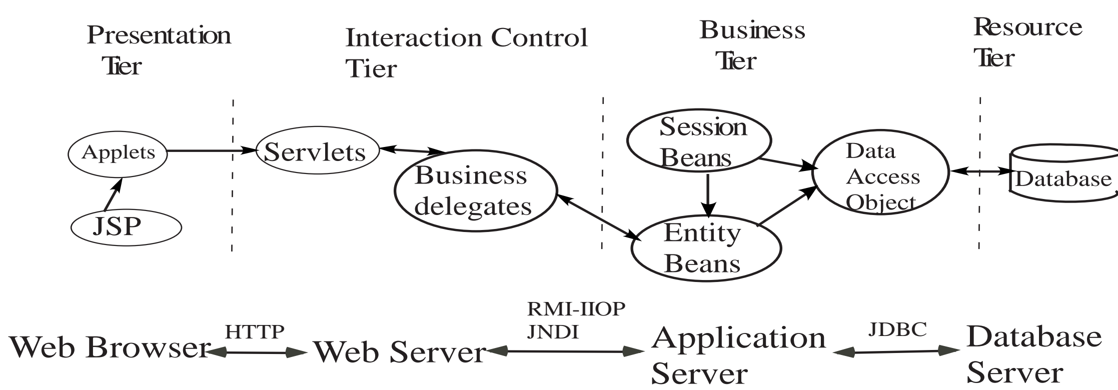

Figure 1: Conceptual Architecture View

ing concerns.

Conceptual Architecture View

A conceptual architecture view consists of conceptual components linked together to deliver the functionality of the application. In a distributed component-based system, there are explicit conceptual views. This view is usually tied closely to the infrastructure model. Dependent on the conceptual view, we can specify the software basic architecture of a system. For example, general distributed component-based systems can be described as in Figure 1. Here the conceptual components are distributed components, such as JSPs, Servlets, EJBs and Databases, which are distributed on different layers, play different roles, and are supported by different servers.

Code Architecture View

A code architecture view describes how the software implementing the system is organized. In this view, individual elements are abstracted from source components and deployment components (for example, libraries and configuration files).

Execution Architecture View

An execution architecture view describes the control flow from the point of view of the runtime platform. This view helps the developers to understand the following concerns.

- How does the system meet its performance requirements?
- What are the impacts of a change in the runtime platform?

As systems are distributed, the developers need to understand how functional components map to runtime entities, and how communication, coordination, and synchronization are handled.

Component Architecture View

A component architecture view is used to make explicit how the functionality is mapped to the components. All the relationship among the implementing components must be explicit, including how the system uses the underlying software platform (system services). Specifically, it shows components and organizes them into layers. In general, component architecture view can help the developers to understand the following concerns.

- What component is mapped to which server?
- What support/services does a component use?
- What dependencies between components exist?

As a whole, this architecture view provides an understanding plan for a system at a high level of abstraction. Individual functions and components are not described in detail; instead, relations between them are emphasized. This level of abstraction is appropriate for understanding the entire software system.

Unfortunately the architecture documentation associated with an application does not commonly exist and, even when it does, is rarely complete or up-to-date. In order to support better maintenance we need to reconstruct the architecture views.

Reconstructing Architecture Views

In distributed component-based systems, component instances are implemented in various programming languages, are loaded to different component environments, and run on multiple hosts distributed across the network. For this reason it is impossible to reconstruct the architecture views uniformly and/or fully automatically. We therefore propose a semi-automated approach. Figure 2 shows the steps involved in abstracting out the architecture views.

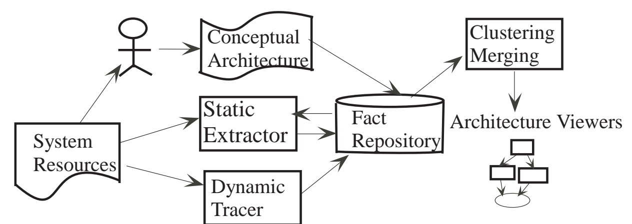

Figure 2: Reconstruct Approach for Views

This is a semi-automated approach. The conceptual architecture view is obtained manually from documents, the users' guidance, and interviews of experts. The facts extracted from the conceptual view constitute the basic framework structure: layers, functional modules, conceptual components, and communication protocols used between the layers. These facts

employed by the (automatic) extractor tools.

Extractors for EJB-based application

Distributed component-based applications are developed using various component models. From now on we will concentrate on one major model — that of Enterprise JavaBeans (EJB)[10] — to illustrate our approach. A variety of languages are used in developing distributed component-based applications. To deal with this situation, we developed three static extractors and and a dynamic execution tracer, as follows.

- JSP extractor
- XML extractor
- Java Source Code extractor
- Execution Tracer

Reconstructing Process

A reconstructing process depends on the above extractors. Each extractor generates different type of data. The extractors abstract out facts about components, relations, and attributes of a software system.

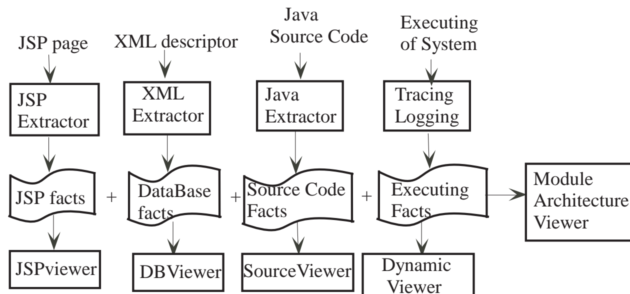

Figure 3: Extraction Process

The reconstructing process for EJB-based applications is presented in Figure 3. Extracted facts could be as detailed as correspondence of EJB entities and database tables, access of EJB entities and servlet components, client objects and JSP objects. The level of detail depends on the analysis to be performed. JSPViewer, DBViewer, SourceViewer, and DynamicViewer are used to show individual structures.

Architecture Views

According to the level of abstraction, we may organize the four architecture views as in Figure 4. The component ar-

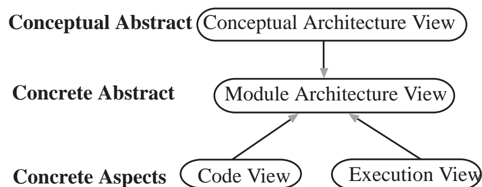

Figure 4: Relationships of Views

chitecture view is the central view for maintenance purposes. All the views are displayed in diagrams, using graphviz[2] toolkits. As can easily be imagined, if we use a single diagram for a large system, it is hard to comprehend. To cope with this problem of size, we employ a clustering tool that deals with containment relations and information hiding.

Case Study: Reconstructing E-process Architecture Views

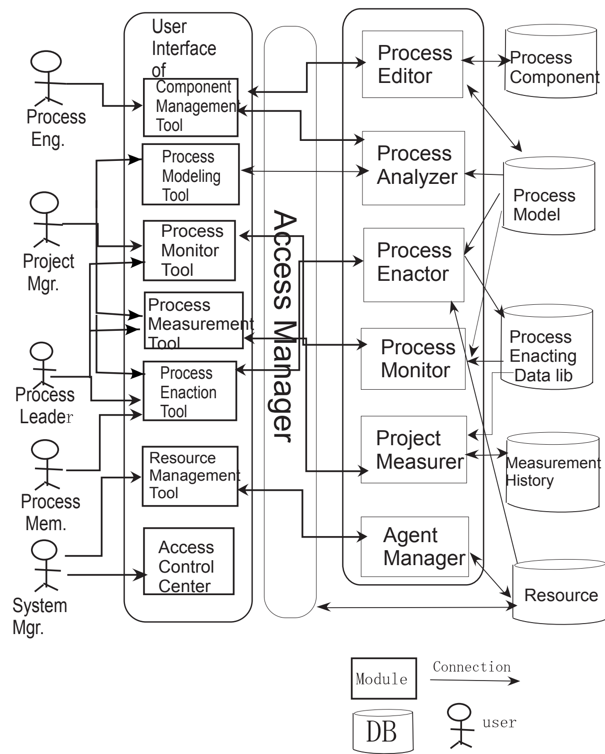

Figure 5: E-Process Conceptual Architecture View

E-Process is a software process support system based on a multi-tier distributed EJB component model[3]. The system has over 700 files and five databases. It consists of seven subsystems and delivered on Tomcat and Jboss[6] server. The space restriction does not allow us to present all the architecture views of such a large system. We instead illustrate our approach in terms of an E-Process subsystem called AccessTool.

As shown in Figure 3, we employ the extractor tools and a dynamic tracer to abstract out E-Process resources. Each extractor parses a relevant type of components to generate appropriate facts. These facts are then gathered to produce a concrete abstract view (that is, component architecture view).

Tools Prototyping

We developed a set of extracting and display tools.

JSP Extractor

A single JSP file may contain multiple sections written in HTML, Script, and Java codes. In implementing our lightweight (not light-hearted!) extractor, we decided to extract only the entities we are directly interested in, using partial grammar rules for the interesting entities. Figure 11 shows the ex-

tracted code view of JSP.

XML Extractor

A Web and EBJ-based application contains various deployment descriptors in XML that contain information about the JSP pages, servlets, EJBs and databases used in the application. We implemented an XML parser to extract the facts. As an example, Figure 12 shows the extracted code view of XML for EJB and database tables invoked by AccessTool.

Java Source Code Extractor

Instead of implementing a complete parser, we scan the connection descriptions among the distributed objects. Distributed objects communicate via protocols. Therefore, we use specific protocols to extract possible connections. For instance, Figure 13 shows the interrelations between servlet objects and EJB objects.

Execution Tracer

We implemented the dynamic tracer using AspectJ[1]. Figure 14 show the part of the control translation of objects.

Display Tools

Extracted and abstracted data are displayed by diagrams, where

- ellipses represent execution components,
- rectangular boxes represent functional modules,
- multi-peripheric boxes represent abstracted components,
- octagons represent EJB entities, and
- arrows represent some connections, including control flows.

Architecture Views of E-Process

The Architecture views of E-Process are generated. **Conceptual Architecture View**

The conceptual architecture view describes the system in terms of its major components and relationship among them. The E-Process's conceptual view is documented explicitly[3]. This view is informally described in Figure 5. The rounded boxes represent layers while the angular ones represent functional components. The arrows represent connectors (data access and protocols). Connectors can be further detailed in terms of protocols, as in Figure 6. The layers, components and protocols are facts which contribute to the concrete component view.

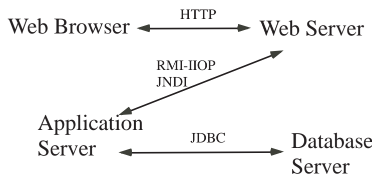

Figure 6: Employed Protocols

Examples of Code Architecture View

A code architecture view not only show the program resource but also shows the binary configuration information. Some example program code views are shown in Figure 12 and Figure 13. The binary configuration is shown in Figure 7.

A Example of Execution Architecture View

A example of execution view is shown in Figure 14, The transformations among part of objects of AccessTool are shown.

The Component Architecture View

The component architecture view is reconstructed using the code and execution views described above. In order to abstract out the concrete component architecture view, we introduced a metamodel, as shown in Figure 8. Using the meta-

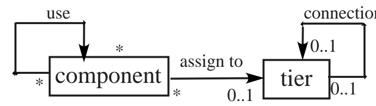

Figure 8: Meta model of component view

model we lifted the component to the layer level.the Figure 9 shows E-Process system component structure.

Applying Architecture Views

We gained better understanding of E-Process with abstracted architecture views.In our case study, the most important role of the architecture views is to support making decisions on modification of system precisely. For example, E-Process is a aggregable tool environment, but in practice, there is a requirement that is to decompose the E-Process into some tools according to software process activities. Depending upon the component architecture view,we can easily distinguish relative components and organise them into the new tool.The stand-alone System Management Tool(SMT) can be abstracted as Figure 10 based on E-Process component architecture view. It suggests that we can easily decompose

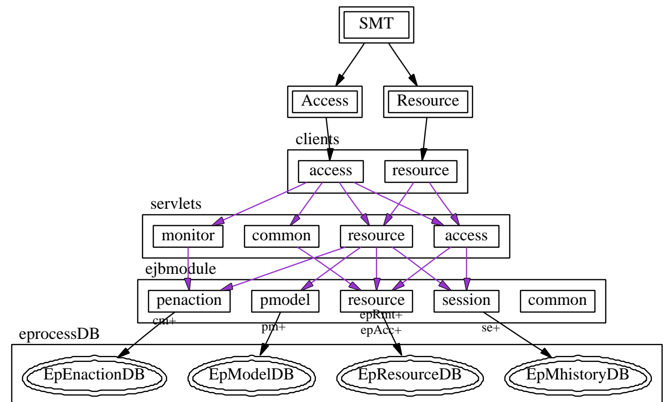

Figure 10: Component Architecture View of SMT

components on client layer.

Conclusions

Developers have to comprehend many aspects of a system. Some of them are static, while others are dynamic. This paper has shown an approach for providing integrated representation of many of those aspects, based on four architecture views. According to our observation of practitioners, these four views are indeed effective in understanding systems. In fact, when they want to understand a system, the first thing they will do is to reconstruct these views from that system.

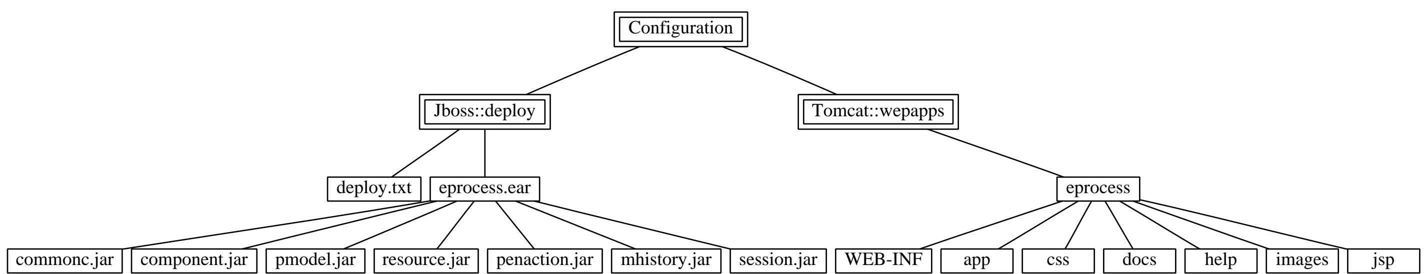

Figure 7: Code Architecture View:Binary Configuration

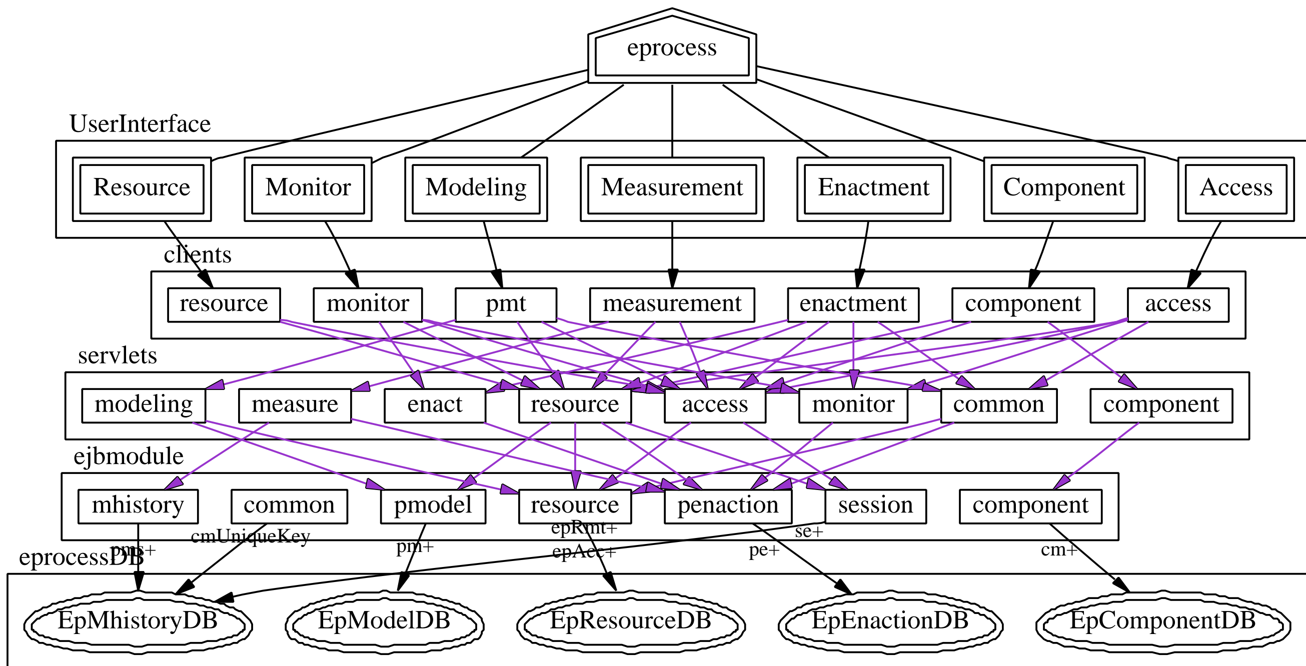

Figure 9: Component Architecture View

In our study so far, the following goals are achieved. On the one hand, we separated different engineering concerns into separate views, to mitigate the complexity of understanding large and complex systems. On the other hand, we showed a semi-automatic approach to extracting architectural documentation from an implemented system.

In the future we want to validate our approach by applying it to various systems and developers.

Acknowledgment

We would very much like to thank Kouichi Kishida, Yoshitaka Matsumura and SRA Key Technology Lab. for their support of this research. We also thank colleagues specifically Kazunori Shioya and Ataru Nakagawa for their valuables suggestions.

REFERENCES

1. Aspectj Team. *Aspect-Oriented Programming with Java: Aspectj*.
   http://aspectj. org/.
2. AT&T Research. *Graph Visualization Project*.
   http://www. graphviz.org/.
3. E-Process Team. *E-Process Design Documentations*. SRA and ASTI, 2001.
4. X. Fang and T. Tamai. Abstracting architecture of distributed system for system understanding. *Research Report of Software Engineering in IPSJ,In Japanese*, 137(4), 2002.
5. P. J. Finnigan, R. C. Holt, I. Kalas, S. Kerr, H. A. M. K. Kontogiannis, J. Mylopoulos, S. G. Perelgut, M. Stanley, and K. Wong. The software bookshelf. In *IBM Systems Journal*, number 36 in 4, pages 564–593, October 1997.
6. JBoss.org. *JBoss*.
   http://www.jboss.org/, 1999-2002.
7. R. Krikhaar. Software architecture reconstruction. *Ph. D. Thesis University of Amsterdam, Amsterdam, The Netherlands*, 1999.
8. SRA Inc. . *Total Computer Aided Reengineering Environment*.
   http://www.sra.co.jp/TCARE/, 1998.
9. SRA Inc. . *VisualBasic Reference Tracking*.
   http://www.sra.co.jp/reftrack/, 2000.
10. Sun MicroSystems. *EJB Specification*.
    http://java.sun.com/j2ee/, 2001.

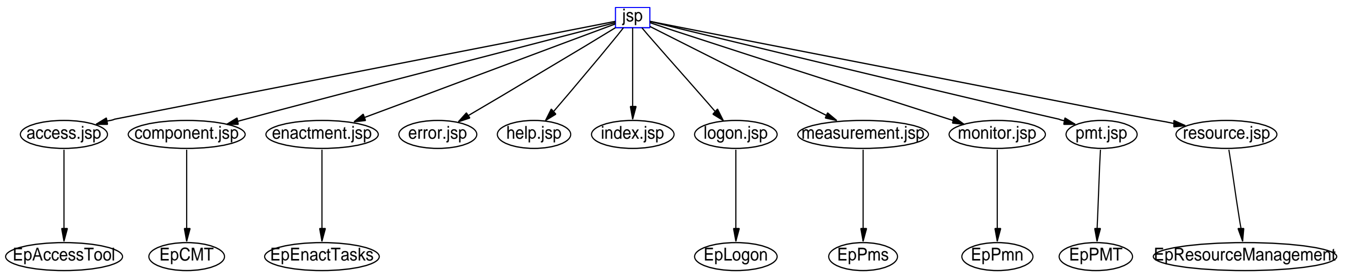

Figure 11: JSP Code View

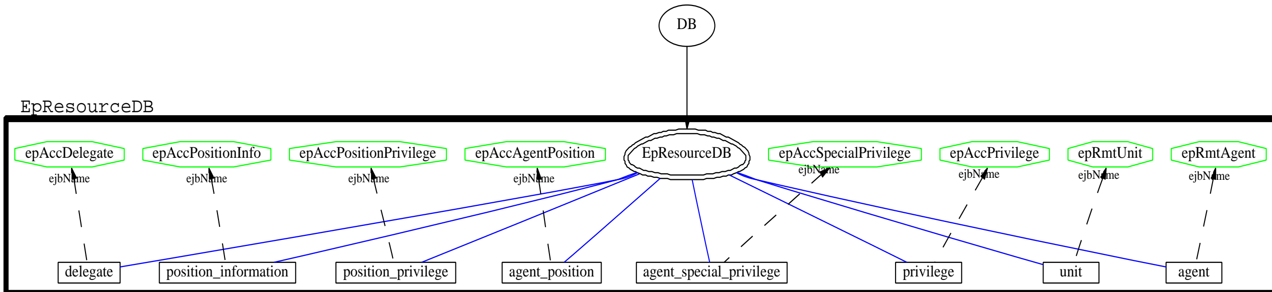

Figure 12: Code Architecture View: EJB Entities and Tables of Database

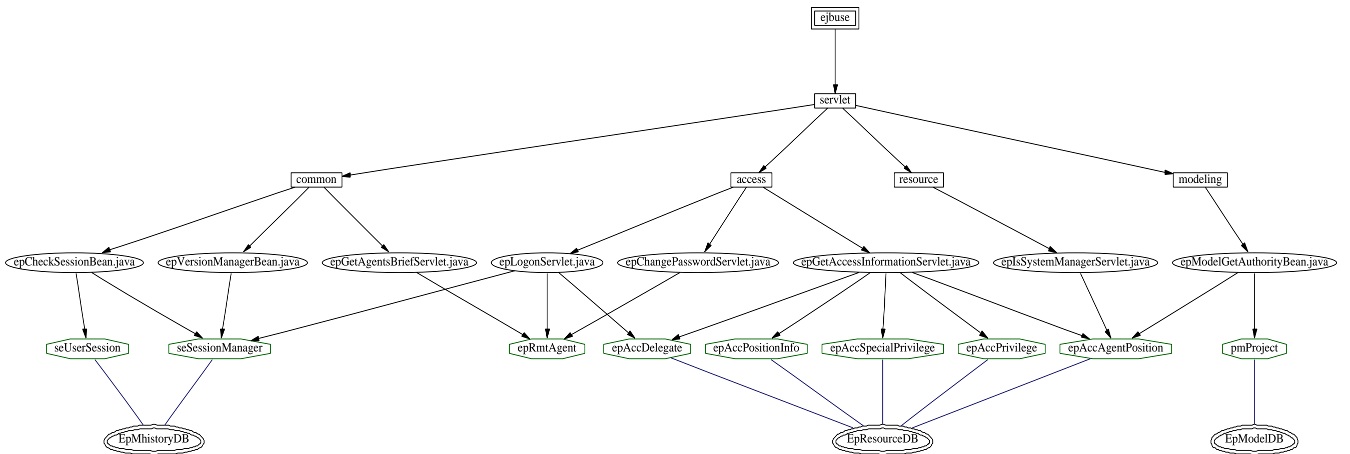

Figure 13: Code View: Servlet objects and EJB objects

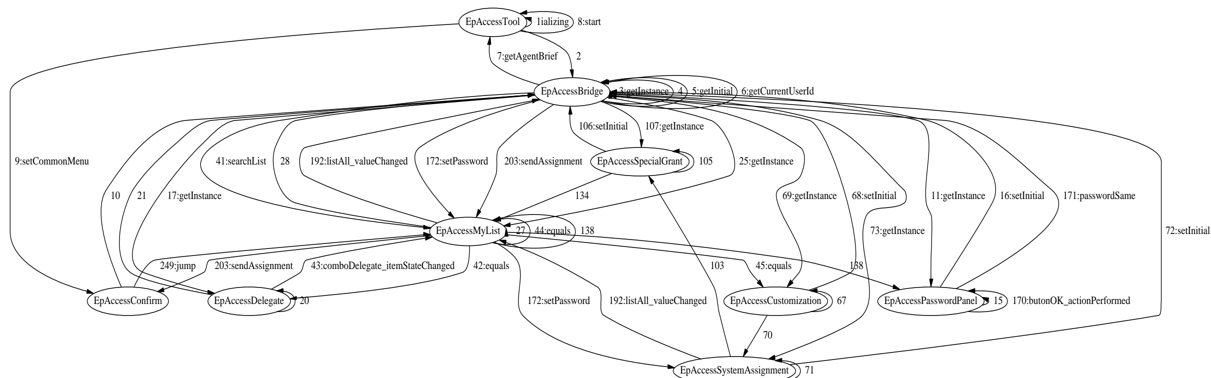

Figure 14: Execution View: Object Transformation in AccessTool