Enterprise Navigator: A System for Visualizing and Analyzing Software Infrastructures

A. Buchsbaum, Y. Chen, H. Huang, E. Koutsofios, J. Mocenigo, A. Rogers
AT&T Labs – Research, Florham Park, New Jersey, USA

M. Jankowsky
AT&T Network Services, Middletown, New Jersey, USA

S. Mancoridis
Drexel University, Philadelphia, Pennsylvania, USA

Abstract

Operations of today's large corporations are usually supported by complex software infrastructures that involve hundreds or thousands of software systems. Companies frequently need to redesign their software infrastructures in response to changes in the marketplace. This paper describes Enterprise Navigator, a system that allows architects to visualize system interconnections of selected products and services by making database queries on the web. Moreover, analysis tools are provided for the architects to examine dominating information flows, perform clustering analysis to find substructures, and study the structural evolution of particular business processes or functions. The system has been used extensively on AT&T's System Profile Database (SPDB). A case study is included in this paper to show how an architect can use Enterprise Navigator to perform various visualization and analysis tasks.

Keywords: Enterprise Database, Software Infrastructure, Database Visualization, Software Architecture, Dominator, Clustering, Java, JDBC, World Wide Web.

1. Motivation

Operations of today's large corporations are usually supported by complex software infrastructures. Thousands of software systems and interfaces support AT&T's array of telecommunications services and products. Each interface serves as a conduit for data to flow between one system and another system or entity (network element, work center, organization, database, etc.). Typical business processes include account maintenance, billing, customer care, capacity management, financial management, marketing, and ordering & provisioning. A software system may support multiple products, services, and business processes. Likewise, every product, service, or business process may involve many cooperating software systems, exchanging data through the system interfaces.

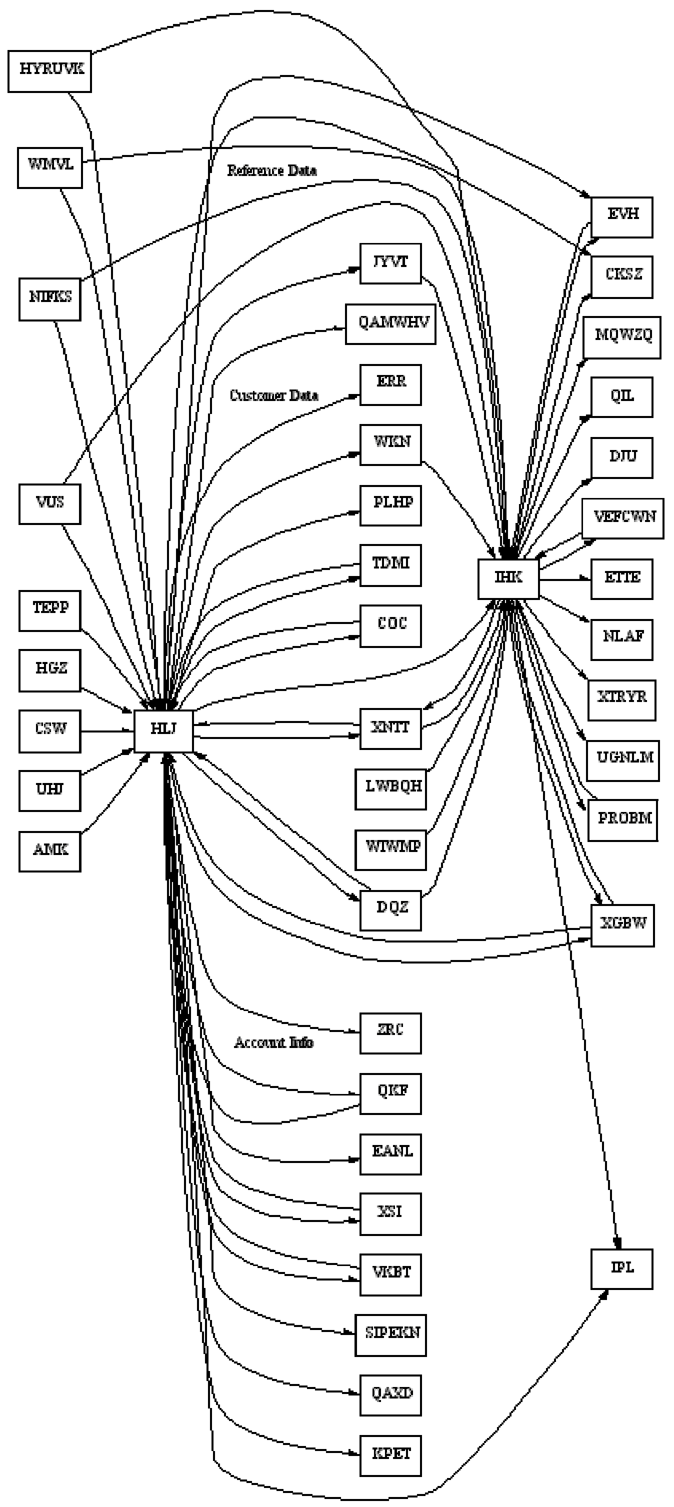


Figure 1. A Typical System Interface Diagram Generated by Enterprise Navigator

Companies frequently need to redesign their software infrastructures in response to changes in the marketplace. Such business reengineering activities must be performed with care, however, so that the new architecture will not disrupt existing operations or increase the operating cost in unnecessary ways. To support these goals, system architects have long recognized the need to collect information about all systems and interfaces in a repository. Architects use this information to draw *system interface diagrams* manually in order to help them study the existing architecture. Theses diagrams are updated, published annually, and distributed throughout the business units.

The process of manually drawing system interface diagrams is tedious and error-prone: a simple system interface diagram that shows all system interconnections to a single system could take 30 minutes or more to draw, and the diagram often becomes obsolete before it is published. Moreover, it is not easy, through the draw-and-publish mechanism, to get a system interface diagram based on an ad-hoc query. For example, a manger in charge of reengineering of billing operations may want to generate a diagram that involves all systems that perform bill calculations under a particular product or service offer.

This paper describes *Enterprise Navigator,* a system we have built to allow users to specify and execute ad-hoc queries and generate a system interface diagram automatically. For example, Figure 1 shows a typical diagram generated by Enterprise Navigator for a particular ad-hoc query.[1] Each node represents a system, and each link represents an interface between the two connected systems. Moreover, Enterprise Navigator allows users to (a) study *architecture evolution* of a system interface diagram over time, (b) perform *clustering analysis* to find substructures embedded in complex diagrams and (c) perform *dominator analysis* to determine systems that dominate information flows. Enterprise Navigator can run as a collection of stand-alone tools using a set of database visualization tools called CIAO **[1]**, or as an integrated web service, on which we will focus in this paper.

Our work builds on established research in source code analysis, graph drawing, and reverse engineering. Acacia **[2]** and Chava **[3]** are examples of reverse engineering tools for the analysis of C/C++ and Java programs, respectively. In these systems, source-code analysis results are stored in a database so that ad-hoc queries can be used to extract software structure information without relying on customized parsers. Visualization tools that employ automatic graph-drawing algorithms **[4][5]** are frequently employed to help software engineers comprehend the results of their analyses. Many reverse engineering techniques, including techniques for software clustering **[6]** and dominator analysis **[7]**, have underpinnings based on optimization theory, statistics, and graph theory.

To date, the aforementioned techniques and tools have been applied mainly to individual software systems written in a variety of programming languages. The work described in this paper takes the next step by showing how the entire software infrastructure of a large enterprise such as AT&T can be modeled, queried, analyzed, and visualized when the infrastructure information is available in a database.

The rest of the paper is organized as follows. Section 2 describes the architecture of this system. Section 3 shows how a user interacts with the

[1] All real system names have been replaced by randomly generated names, and certain interface names are not shown in this diagram to protect proprietary information.

Enterprise Navigator using a case study. Section 4 concludes with summary and a discussion of future work.

2. Architecture of the Enterprise Navigator

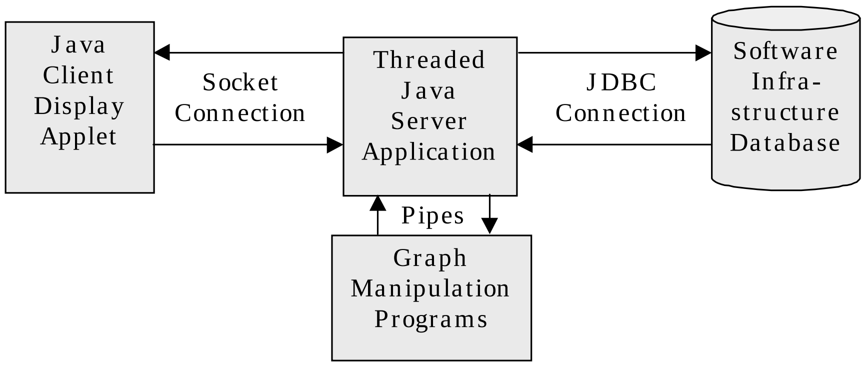

Java Client Display Applet
Socket Connection
Threaded Java Server Application
JDBC Connection
Software Infra-structure Database
Pipes
Graph Manipulation Programs

Figure 2. Enterprise Navigator Architecture

Figure 2 presents a high-level view of the architecture of the Enterprise Navigator. End-users interact with the Enterprise Navigator by means of a Java applet (see Fig. 5). The applet establishes a two-way socket connection to a Java application running on a server machine. The Java application communicates via a JDBC (Java Data Base Connectivity)[2] connection with a database of software infrastructure specifics. End-user visualization requests are passed to the server application, which formulates an SQL query to retrieve the necessary information from the database. The server application then constructs a *system interface graph* and opens a connection to a graph layout program to position the elements of the graph automatically. When the layout is complete, the graph is sent using Java object serialization to the applet, which creates a visualization window and displays the requested *system interface diagram* to the end-user (see Fig. 7). Nodes and edges can be selected to view their attributes, the graph display can be altered based on those attributes, and the graph can be returned to the server for additional processing by graph clustering or graph dominator algorithms.

The glue holding the Enterprise Navigator together is the Java application on the server machine. The components of the Enterprise Navigator linked by the server application are described in more detail below. Those components are the

- infrastructure database, System Profile Database (SPDB) (Section 2.1),
- graph manipulation and display tool, Grappa (Section 2.2),
- graph clustering tool, Bunch (Section 2.3), and
- graph dominator tool, Dominator (Section 2.4).

[2] Visit http://java.sun.com/products/jdbc for more information on JDBC.

2.1 System Profile Database (SPDB)

The underlying database supplying the Enterprise Navigator is the System Profile Database (SPDB), which contains key information about all system entities and interfaces within the enterprise of interest. The three most important tables are the following.

- The *System* table contains basic information about each system in the entire business enterprise. It also includes such entities as work centers, network elements, databases, and web sites as well as external systems that participate in flows of data to or from systems within the enterprise. Information about a system can include system type, system name, system owner, business unit owner, system status, phase in/out dates, Y2K status, and its parent system.
- The *Interface* table gives information about flows between systems and other entities described in the *System* table. Information about an interface can include interface type, the "*from*" system, the "*to*" system, interface owner, business unit owner, transmission media, transmission frequency, transmission mode and interface status.
- The *Mapping* table links other entities such as products and services or business functions to systems in the *System* table.

In constructing a system interface diagram, the Enterprise Navigator constructs a graph of the components setting the systems as nodes and the interfaces between them as edges. Additional pieces of information about those entities are stored as attributes to the graph elements.

2.2 Graph Manipulation and Display (Grappa)

Graph manipulation and display in the Enterprise Navigator is handled by a Java package called *Grappa* **[7]**, which is freely available on the web.[3] Grappa is used in the Enterprise Navigator both in the client applet and in the server application. Grappa can be used to build and manipulate a graph independent of display considerations. Although Grappa does not contain layout algorithms, it has methods for simplifying communication with graph layout programs, particularly the *dot* layout program **[5].**[3]

Mouse interactions with the nodes and edges displayed by Grappa can cause additional actions to be triggered. In the Enterprise Navigator, these actions include triggering a new query specific to a selected element, and viewing or storing additional data about an element. Moreover, to study architecture evolution over time, the applet could be set up to color systems according to a reference date and the status of each system at that date. Those that have been *phased out of service* or *yet to be introduced* are colored differently from those of *active* systems. Visualization of these changes helps architects determine the effects of business reengineering on various products and services.

Grappa is designed to be extensible. The next two sub-sections describe two applications that were integrated into the Enterprise Navigator without re-coding. The server application acts as a bridge between those applications and the display applet.

[3] Grappa and dot can be downloaded from http://www.research.att.com/sw/tools/graphviz/

2.3 Graph Clustering (Bunch)

The Enterprise Navigator uses the Bunch **[8]** tool to provide a means of clustering components in the system interface diagrams. Clustering is particularly useful to system architects who are trying to understand large and complex software infrastructures from their graph representation.

Bunch accepts, as input, a graph and produces, as output, a partitioned version of the input graph into a set of non-overlapping groups (clusters) of nodes. Using Grappa, a partitioned system interface diagram can be shown as a graph with clusters of nodes enclosed in rectangles. The Java server application communicates with Bunch through an application program interface.

The objective of Bunch is to partition the software graph so that system entities (nodes) in the same cluster are more closely related and system entities in different clusters are relatively independent of each other. Creating a meaningful partition of a system interface diagram, however, is difficult because the number of possible partitions is very large even for a small graph. Also, small differences between two partitions can yield very different results. As an example, consider Figure 3a, which presents a graph with a small number of entities and relationships. The two partitions of the graph presented in 3b and 3c are very similar, with only two nodes (M3 and M4) swapped. In spite of this seemingly small difference, the partition defined in Figure 3c better captures the high-level structure of the graph, since it groups nodes that are more inter-dependent.

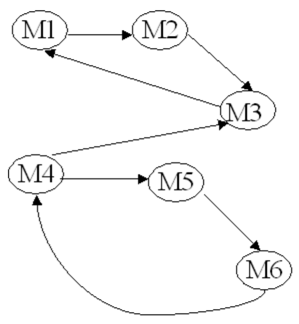


Figure 3a An Example of a small system interface graph.

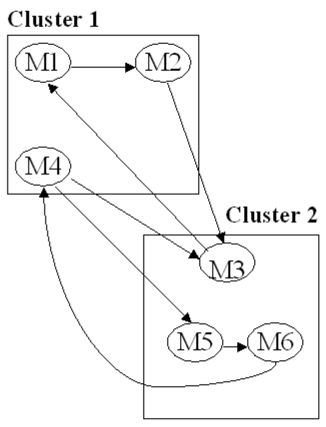


Figure 3b A partition of the system interface graph.

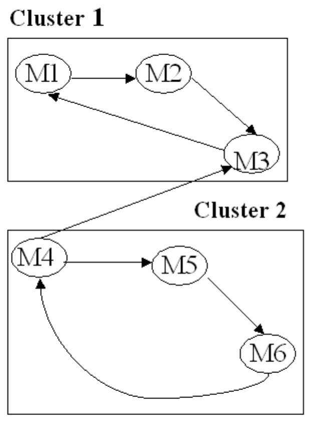


Figure 3c A better partition of the system interface graph.

Bunch treats graph clustering as an optimization problem, where the goal is to maximize an objective function that favors the creation of clusters that exhibit a high degree of intra-edges. Intra-edges are edges between the nodes of the same cluster. The same function penalizes pairs of clusters that exhibit a high degree of inter-edges. Inter-edges are edges between nodes that belong to different clusters. A high degree of inter-edges is an indication of poor partitioning. Having a large number of inter-edges complicates software maintenance because changes to a software system may affect other systems of the software infrastructure. A low degree of inter-edges is a desirable trait of a system architecture and is an indicator that the individual clusters are, to a large extent, independent. Therefore, changes applied to a software system are likely to be localized to its cluster, which reduces the likelihood of introducing errors into other systems.

2.4 Graph Dominators (Dominator)

The Enterprise Navigator uses the Dominator tool to provide a means of determining the dominators of a graph. In a graph with a selected root node R, node X dominates node Y if every path from R to Y goes through X. We explain dominators with an example. When the Enterprise Navigator generates a system interface diagram, each interface link between systems represent information flows. For example, in Figure 4a, the link from A to B means that information flows from A to B. The dominator tree derived from our infrastructure graph is shown in Figure 4b. A link in the dominator tree between two nodes means that any flow of information from the root node (selected from the original graph) to the target node must flow through the source of the link. For example, if A is the source node and there is a dominator link from B to C, then there is no way to get from A to C without going through B. In other words, if B were to be removed, C would be cut off from any information derived by A. On the other hand, consider the links from D to E and A to E in the original graph. The direct link from A to E provides a way to get to E from A without going through D; therefore D is not a dominator of E. In fact, A is the sole dominator of E.

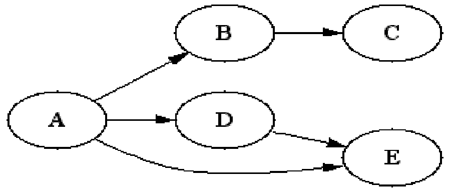


Figure 4a. A System Interface Graph

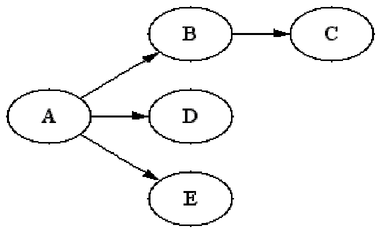


Figure 4b. Its Dominator Tree

The root node of a dominator tree represents the system where the flow of information logically begins. In cases where a single root is not available, multiple roots can be chosen from the graph to act together as the global information source.

Our tool uses the *dominators* algorithm devised by Lengauer and Tarjan **[5].** Buchsbaum et. al **[6]** provide a history of dominators algorithms as well as theoretical improvements to the Lengauer-Tarjan algorithm. Examples of applications for dominator trees in the Enterprise Navigator include the following.

System evolution sanity checks: Removing a system disconnects all systems dominated by it (and by extension systems dominated from those, and so on) from the original information source. Therefore, systems that are scheduled to be retired should not dominate any systems that are not being retired. The Enterprise Navigator allows this situation to be checked visually by uniquely coloring systems to be retired on a dominator diagarm.

Qualitative assessments of dependency complexity: Dominator trees that are flat, i.e., in which many systems are directly connected to the root(s), can represent highly interconnected systems, because there are few systems whose removal disconnects the graph. Such high interconnectivity can be good due to replicated resources for system dependability, or bad due to unnecessary or duplicated information flows. On the other hand, dominator trees that are deep, i.e., in which many systems are far from the root(s), can represent less connected systems because many systems critically depend on many others for connectivity from the root. Again, this may be good or bad, depending on the application.

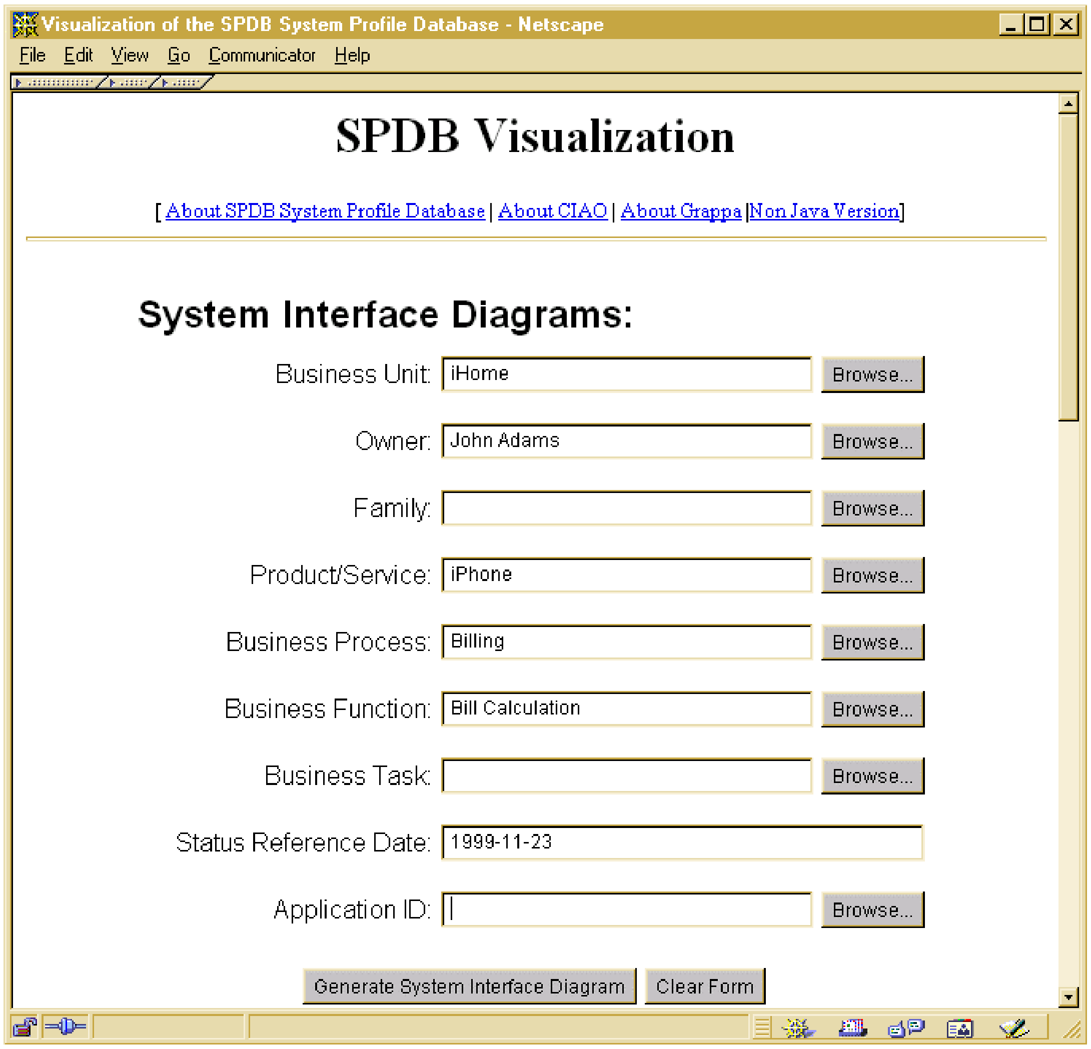
3. Case Study

To illustrate various functions of the Enterprise Navigator, this section shows how a user can construct queries, generate system interface diagrams, perform clustering analysis, and run the dominator tool, all through a web interface.

Figure 5 shows our query interface presented by the Java applet. The interface allows users to select systems from different business units, owners (directors), products/services, business functions, and so on before generating a system interface diagram. The parameters selected by users in some categories determine what choices are available in other categories. For example, when a user clicks the *Browse* button next to the Business Unit category, the list of all business units appears. If the user chooses iHome[4] and then clicks on the *Browse* button next to the Product/Service item in Figure 5, Figure 6b will appear and show all products and services under the iHome business unit only. If the user selects iPhone and browses the systems under that service, then the list of systems appears (Figure 6c). The user can refine the query further by setting the value of Business Process and/or Business Function, etc.

The user is free to pick any system from the list to generate a system interface diagram. Figure 7 shows a typical diagram centered around the system HLJ. The picture clearly shows that HLJ collects routing and rating data and then distributes reference data (among other things) to quite a few systems.

[4] All business unit names, product names, and system names have been replaced with imaginary names to protect proprietary information.


Visualization of the SPDB System Profile Database - Netscape
File Edit View Go Communicator Help

SPDB Visualization

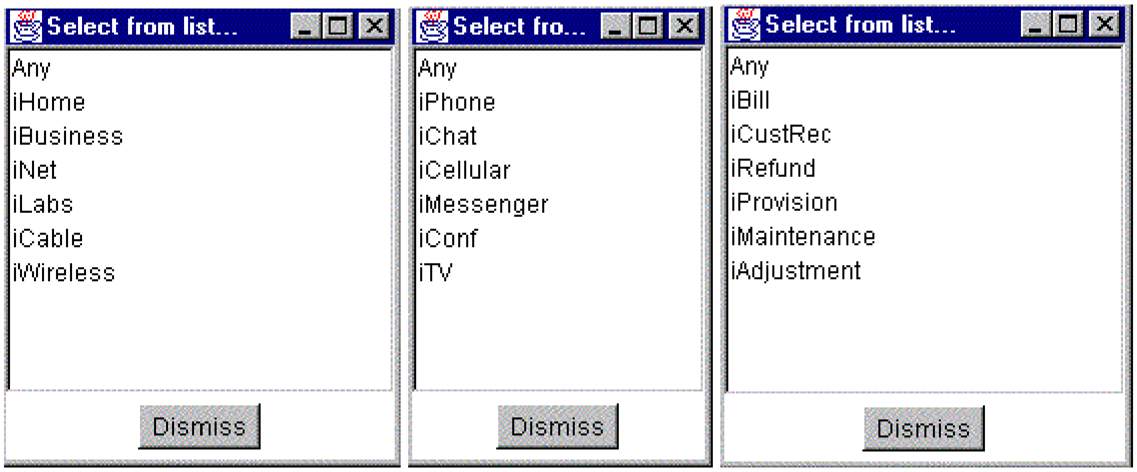
[About SPDB System Profile Database | About CIAO | About Grappa |Non Java Version]

System Interface Diagrams:

Business Unit:	iHome
Owner:	John Adams
Family:	
Product/Service:	iPhone
Business Process:	Billing
Business Function:	Bill Calculation
Business Task:	
Status Reference Date:	1999-11-23
Application ID:	

Browse...
Generate System Interface Diagram
Clear Form

Figure 5. Query Interface of the Enterprise Navigator



(a)	(b)	(c)
Any	Any	Any
iHome	iPhone	iBill
iBusiness	iChat	iCustRec
iNet	iCellular	iRefund
iLabs	iMessenger	iProvision
iCable	iConf	iMaintenance
iWireless	iTV	iAdjustment

Figure 6. Lists of (a) Business Units (b) Products and Services under iHOME (c) Systems under iHOME that are involved with the iPhone service.

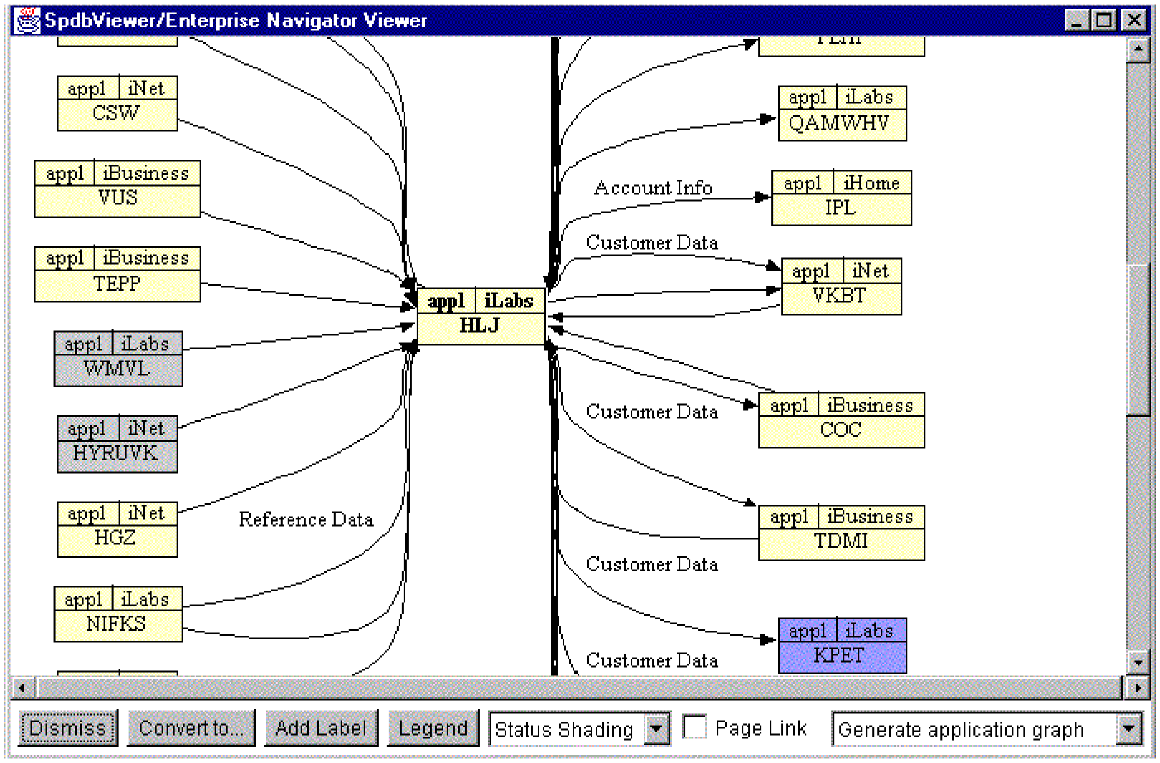


Figure 7. System Interface Diagram of HLJ

In every system interface diagram, each node can be colored according to different attributes of the system. In Figure 7, we use the *status shading* scheme, in which a beige node indicates an *active* system and a gray node indicates that we have *insufficient information* about that system. Using the reference date (1999-11-23) shown in the query interface page (Figure 5), a cyan node indicates that this system is *planned* and will be introduced soon and a purple node indicates that this system has *retired*. As it is important to know how a business reengineering plan affects the system architecture, a system architect can use different reference dates to see how the system interface diagram of a particular product or service has evolved or will become. Another shading scheme is *Y2K shading*, in which case nodes are colored according to whether they are Y2K-compliant or not. This scheme allows us to verify the Y2K readiness of any particular product or service quickly if the Y2K compliance data are available.

Instead of selecting a single system, a user can simply choose to generate a system interface diagram for all systems involved in a product or service (or any systems that satisfy a particular query), as shown in Figure 1. If the user would like to discover clusters of systems embedded in such a complex structure, she can invoke the *Bunch tool* to convert the diagram to a clustering diagram as shown in Figure 8a. The diagram shows that there are two clusters with a similar architectural pattern: all clusters have a data hub (within that cluster) that receives data from several sources and distributes processed data to many other destinations.[5] It's not always easy to identify clusters from complex system interface diagrams.

If a user is interested in discovering the dominating information flows starting from a particular system, she can select the node and perform a dominator analysis. Alternatively, our tool can perform topological sorting to rank the nodes and add a virtual root to all the top-level nodes before starting the dominator analysis. Figure 8b shows a dominator tree created for the system interface diagram of Figure 1 with this method. It is clear that any attempt to remove the system HLJ will impact all the other systems that it dominates since any information flows to the product or service represented by Figure 1 will have to go through HLJ. Such information can help system architects plan their reengineering efforts.

4. Summary and Future Work

Enterprise Navigator is a system that allows software architects to visualize and analyze the software infrastructures of AT&T. Besides automating the process of drawing system interface diagrams, it adopts the latest Internet technologies to allow users to make ad-hoc queries from anywhere in the company and obtain the corresponding system interface diagrams in real time. Moreover, the architecture of the Enterprise Navigator allows easy integration of external layout and analysis components such as the Dominator and the Bunch clustering tools. These tools have helped us understand the information flows and substructures inside a complex system interface diagram.

The usefulness of the Enterprise Navigator depends heavily on the timeliness of the underlying data. We are working on facilities that would allow architects to update the architecture data directly from the graphs to eliminate the delays associated with the old data collection process. We also plan to add node and link operators that would allow users to examine the corresponding systems and data transmitted on a link in more detail. With the addition of operational data like system availability to the database, we might be able to perform end-to-end enterprise architecture simulations. Finally, we welcome the opportunity to apply the concept of Enterprise Navigator to other forms of enterprise data or data from other companies.

[5] Figure 7 also shows a typical cluster following that pattern.

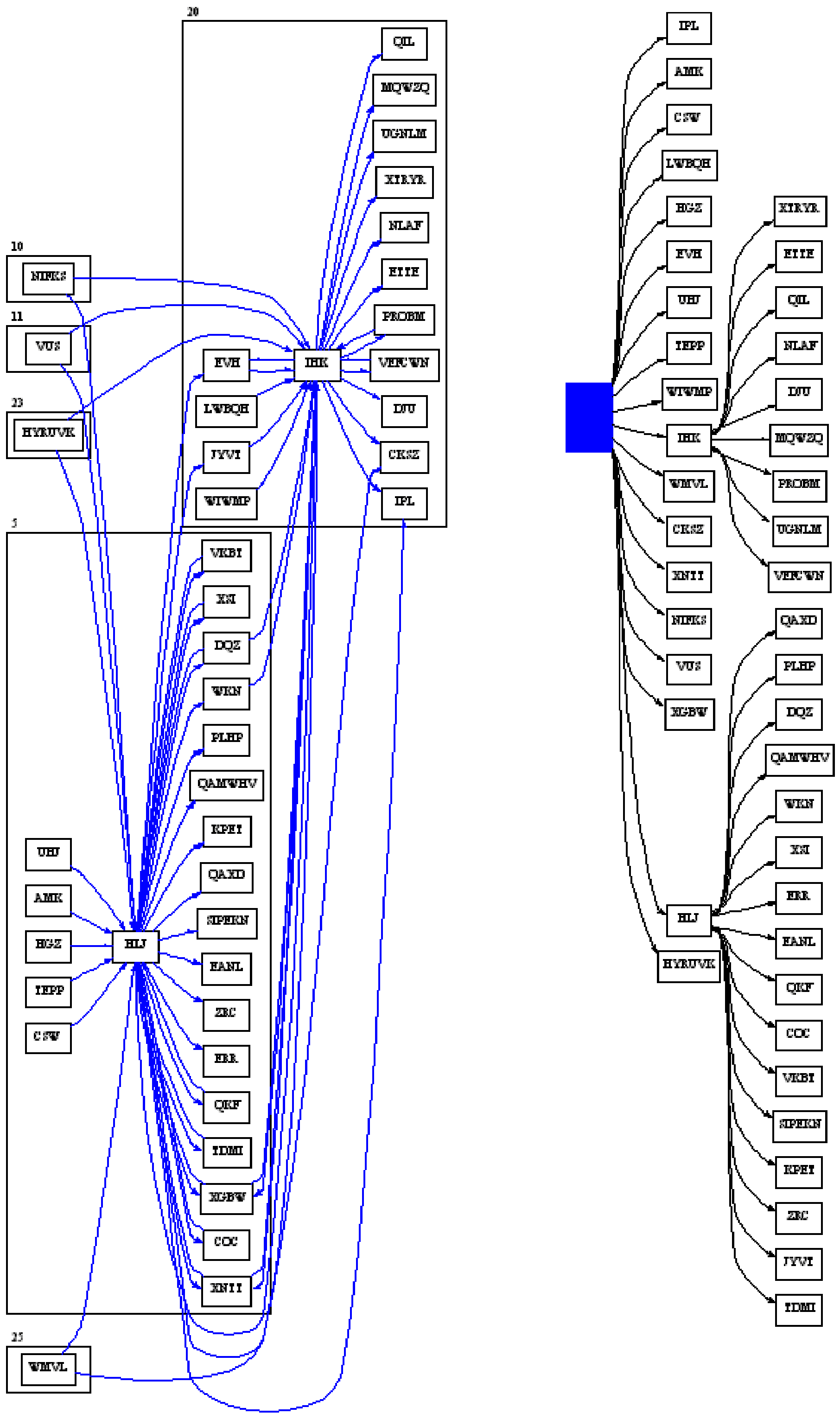


Figure 8 (a) Clustering Diagram and (b) Dominator Diagram generated from the System Interface Diagram in Figure 1

References

[1] Y. Chen, G. Fowler, E. Koutsofios, and R. Wallach, *Ciao: A Graphical Navigator for Software and Document Repositories*, Proceedings of the International Conference on Software Maintenance, Nice, France, pp. 66-75, Oct. 1995.

[2] Y. Chen and E. R. Gansner and E. Koutsofios, *A C++ Data Model Supporting Reachability Analysis and Dead Code Detection*, IEEE Transactions on Software Engineering, 24(9), September 1998, pp. 682-693. Also appeared in Proceedings of the Sixth European Software Engineering Conference and Fifth ACM SIGSOFT Symposium on the Foundations of Software Engineering, Zurich, Switzerland, September, 1997.

[3] J. Korn and Y. Chen and E. Koutsofios, Chava: Reverse Engineering and Tracking of Java Applets, Proceedings of the 6th Working Conference on Reverse Engineering, 1999.

[4] G. Di Battista, P. Eades, R. Tamassia, I. Tollis, *Graph Drawing: Algorithms for the Visualization of Graphs*, Prentice Hall, 1999.

[5] E. R. Gansner and E. Koutsofios and S. C. North and K. Vo, *A Technique for Drawing Directed Graphs*, IEEE Transactions on Software Engineering, 19(3), pp. 214-230,1993.

[6] T. Lengauer and R. E. Tarjan, *A Fast Algorithm for Finding Dominators in a Flowgraph*, ACM Transactions on Programming Languages and Systems, 1(1), pp. 121-141, 1979.

[7] N. Barghouti, J. Mocenigo, and W. Lee, *Grappa: A Graph Package in Java*, Proceedings of the Fifth International Symposium on Graph Drawing, Rome, Italy, pp. 336-343, Sep. 1997.

[8] S. Mancoridis, B. S. Mitchell, Y. Chen, E. R. Gansner , *Bunch: A Clustering Tool for the Recovery and Maintenance of Software System Structures*, IEEE Proceedings of the International Conference on Software Maintenance (ICSM'99), Oxford, UK, August, 1999.

[9] A. L. Buchsbaum and H. Kaplan and A. Rogers and J. R. Westbrook, *A New, Simpler Linear-Time Dominators Algorithm*, ACM Transactions on Programming Languages and Systems, 20(6), pp. 1265-1296, 1998.