SAND REPORT
SAND2001-3091
Unlimited Release
Printed November 2001

Source Code Assurance Tool: LDRD Final Report

Juan Espinoza Jr., Philip L. Campbell

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865)576-8401
Facsimile: (865)576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.doe.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800)553-6847
Facsimile: (703)605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: http://www.ntis.gov/ordering.htm

SAND 2001-3091
Unlimited Release
Printed November 2001

Source Code Assurance Tool: LDRD Final Report

Juan Espinoza Jr.
Cryptography and Information Systems Surety

Philip L. Campbell
Networked Systems Survivability and Assurance

Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-0785

Abstract

This report provides a summary of the work completed in the Source Code Assurance Tool project. This work was done as part of the Laboratory Directed Research and Development program.

Contents

Source Code Assurance Tool: LDRD Final Report

1. INTRODUCTION

This Laboratory Directed Research and Development (LDRD) project was funded for three years starting in October 1999 and finishing in September 2001 and was designed to advance the start of the art in software surety science. The goal of the project was to develop a software application, the Source Code Assurance Tool (SAT), that would assist an analyst, or team of analysts, in the assessment of a system, both software and non-software.

In information system security, the emphasis is on the system. Processes external to the information system, such as human machine interactions, the information system's operating location, the lifecycle of its components, and a range of other concerns must all be addressed in order to assure that a system that incorporates information technology is safe, secure, and reliable.

Because any system of reasonable size contains a range of technologies, the assessment of such a system typically requires a team of analysts who each bring to the table a unique body of expertise. For efficiency's sake, the responsibility for various assessment activities is often partitioned. For example, one team member might be given responsibility for the mechanical aspects of the system, another may address physical security issues, a third may inspect the software, a fourth analyze the networking, another consider electrical engineering problems, and yet another handle system issues. While this arrangement is efficient from a project management viewpoint, it introduces problems in the assessment of the system. While each analyst may understand his own portion of the system very well, he may not understand how behaviors associated with his portion of the system play together with the rest of the system to deliver undesirable outcomes. For this reason, there is a need for a tool that will help analysts answer the following questions of other analysts:

- What happens if my portion of the system delivers this kind of event to your portion of the system?
- How could your part of the system deliver this kind of event to my part?

Currently, assessments do not have a tool to help answer these intra-partition questions. With this new tool at hand, a larger context can be addressed, namely, studying the entire system. Moving to this point is one goal of the Source Code Assurance Tool LDRD project.

2. SUMMARY OF WORK

The scope of the work changed over the three-year period as the application requirements became clearer and we understood that the required resources exceeded the available project resources. The irony is that we needed the very tool we were developing to help us in its development. We developed a prototype software application, wrote several papers and reports, and generated three Technical Advance Disclosures.

2.1 Prototype Tool

We developed a prototype of the Source Code Assurance Tool (SAT) using available commercial-of-the-shelf (COTS) software. We used a slicing engine, CodeSurfer from GrammaTech, and a custom graph editor, created with Graph Editor Toolkit (GET) from Tom Sawyer Software (TSS). The user constructs a graph of the system by using GET and then slices it using CodeSurfer. Bringing both tools together into a new tool allows the user to construct and slice across what we call a "system" since their communication is not through labeled sections of memory.

As an example of what we mean by "system", let two programs, A and B, communicate via a disk file; A writes to the file and, sometime later, B reads from the file; the collection of A and B constitute the minimum for what we call a system. A more complex but relevant example to our industry would be a nuclear weapons system that consists of the weapon, a cable, a notebook computer, and a human. Existing tools can analyze each of the system components. But current tools do not allow analysis of the system in its entirety. It is in their integration that systems have problems, because of a lack of tools that address this area.

The significance of our tool is that it enables analysis of a system that exists in the real world but has not been, prior to our tool, analyzable in an automated way. The following describes how the analyst uses our tool. First, he uses the graph editor to draw a graph (*i.e.*, nodes and edges) to represent software components. He then draws arcs to represent dependencies, attaching the source of an arc to an output statement in one component and attaching the target of an arc to an input statement in another component. He then uses the graphical representation of the system to analyze it. He specifies a starting position (for a forward slice), or a stopping position (for a backward slice), or both (for a chop). The tool then uses the graphical representation to move the slice forward (or backwards) and uses CodeSurfer to analyze each software component.

We experimented with several other COTS products to determine their feasibility in developing the graphical user interface (GUI) for the tool, namely Visio Professional, Microsoft Visual Studio, and Platinum Paradigm Plus, an object-oriented (OO) computer-aided software engineering (CASE) tool. Each had their various strengths and weaknesses, but we decided to build the prototype GUI using the TSS Graph Editor Toolkit instead. We needed a flexible and speedy tool that would allow the user to create, modify, and nest graph representations of their desired systems. GET also comes in several languages (Java, ActiveX, C++) allowing for cross-platform independence.

We hired a technician at GrammaTech to help develop the socket-based control of GrammaTech's product, CodeSurfer. We also hired a Java programmer to help connect the graph editor from Tom Sawyer Software with the socket-based controller so that a user can analyze a system. At the level of abstraction we were working, it would not have been possible for us to develop the constituent software ourselves. For example, the C parser that CodeSurfer uses would, by itself, have consumed all our efforts if we were to have built it ourselves. (Compilers are among the most complex pieces of software.)

2.2 SAND Reports

The detailed results of this project are documented in the following SAND reports.

- Richard L. Craft, Philip L. Campbell, Juan Espinoza, "Source Code Assurance Tool: Preliminary Functional Description." SAND2001-3092. Sandia National Laboratories, Albuquerque, NM. Printed October 2001

- Philip L. Campbell, Juan Espinoza, “Visual Structure Language.” SAND2001-3093. Sandia National Laboratories, Albuquerque, NM. Printed October 2001
- Philip L. Campbell, Juan Espinoza, “Source Code Assurance Tool: An Implementation.” SAND2001-3094. Sandia National Laboratories, Albuquerque, NM. Printed October 2001

The abstracts of the SAND reports are presented in the following sections.

2.2.1 Abstract for “Source Code Assurance Tool: Preliminary Functional Description.” SAND2001-3092

This report provides a preliminary functional description of a novel software application, the Source Code Assurance Tool, which would assist a system analyst in the software assessment process. An overview is given of the tool’s functionality and design; and how the analyst would use it to assess a body of source code. This work was done as part of a Laboratory Directed Research and Development project.

2.2.2 Abstract for “Visual Structure Language.” SAND2001-3093

In this paper we describe a new language, Visual Structure Language (VSL), designed to describe the structure of a program and explain its pieces. This new language is built on top of a general-purpose language, such as C. The language consists of three extensions: explanations, nesting, and arcs. Explanations are comments explicitly associated with code segments. These explanations can be nested. And arcs can be inserted between explanations to show data- or control-flow.

The value of VSL is that it enables a developer to better control a code. The developer can represent the structure via nested explanations, using arcs to indicate the flow of data and control. The explanations provide a “second opinion” about the code so that at any level, the developer can confirm that the code operates as it is intended to do.

We believe that VSL enables a programmer to use in a computer language the same model—a hierarchy of components—that they use in their heads when they conceptualize systems.

2.2.3 Abstract for “Source Code Assurance Tool: An Implementation.” SAND2001-3094

We present the tool we built as part of a Laboratory Directed Research and Development (LDRD) project. This tool consists of a commercially available, graphical editor front-end, combined with a back end “slicer.”

The significance of the tool is that it shows how to slice across system components. This is an advance from slicing across program components.

2.3 Other Reports

We submitted a related SAND report (SAND2000-1465, printed June 2000). We submitted an earlier draft of the paper to the DOE Software Quality Forum and submitted the SAND report to the "Journal of Software Maintenance: Research and Practice."

Our ideas also influenced the only vendor in this area, GrammaTech, to obtain funding from DARPA for the next step in their commercial tool. That next step is a product they refer to as SystemSurfer, a term that they first heard from us back in April 1999.

2.4 Technical Advance Disclosures

The following Technical Advance Disclosures were submitted:

- Technical Advance Disclosure (SD-6886/S-97,508): "Visual Programming Tool (VPT)"
- Technical Advance Disclosure (SD-6885/S-97,507): "Range-Value Propagation (RVP): Approximate Computing"
- Technical Advance Disclosure (SD-6688/S-95,506): "System Analysis Tool (SAT): A Tool For Analyzing Systems Of Software"

See Appendix A.2 for details on the above Technical Advances.

APPENDICES

The appendices contain valuable information pertaining to the objectives and deliverables of the SAT LDRD project. The appendices are organized as follows:

- Appendix A.1 - original proposal submitted to the LDRD office.
- Appendix A.2 - Technical Advance Disclosures
  - Technical Advance Disclosure (SD-6688/S-95,506): “System Analysis Tool (SAT): A Tool For Analyzing Systems Of Software”
  - Technical Advance Disclosure (SD-6886/S-97,508): “Visual Programming Tool (VPT)”
  - Technical Advance Disclosure (SD-6885/S-97,507): “Range-Value Propagation (RVP): Approximate Computing”
- Appendix A.3 – Graph Drawing Tool Survey
- Appendix A.4 – UML Tool Survey

A.1 Original Proposal

A.1.1 Scientific and Technical Soundness

In the development of high consequence systems, one of the perennially difficult problems is the assurance of software used in these systems. Achieving this assurance invariably rests on human inspection and testing of the software[1]. This process is extremely labor-intensive and, therefore, can be time-consuming and expensive. Given this, safety- and security-critical software projects are often forced into one of two unacceptable outcomes – to slip delivery dates to finish manual inspections or to deliver code that has not been full assessed[2]. The quality of the assessment is also highly dependent on the analyst. Analyst biases and the sheer volume of things to be considered in an assessment can lead to critical problems being overlooked by the analyst. For these reasons, a tool that increases the human analyst's level of performance in software assessment – both in terms of time invested and accuracy – would be of significant benefit.

Effective software assessment in these systems requires that the analyst take a holistic view of the software system and not just focus on an assessment of the software itself. In addition to answering the question:

- Are there weaknesses/vulnerabilities within the software that could lead to system failure or compromise?

The analyst must also determine:

- Can the software fail due to vulnerabilities in the platform (computing device and operating system) on which the software runs?
- Can interactions between the system's hardware elements (other than the computing platform) and software lead to failure of the system?

To answer these questions the analyst needs to understand the various causal relationships that exist (a) within the software, (b) between the software and the computing platform, and (c) between the software and the rest of the system. The analyst must also know the various failure mechanisms / vulnerabilities that exist in the computing platform.

These two tasks each contribute in their own way to the time-intensiveness of assessment. First, identifying the causal relationships within a system is currently a manual process. While tools are available within "integrated development environments" that help the analyst browse the software, they are usually limited in their capabilities. The analyst typically ends up tracing by hand through the series of function calls and equations that contribute to the state of a variable in question or that depend on this variable. Once these causal chains are identified, it is then up to the analyst to decide whether or not given undesirable states can be reached via those chains. While this latter task re-

[1] While much research has gone into the use of formal methods to "design in" assurance, the methods have yet to gain widespread acceptance or to be proven on large-scale engineering projects. Even where they are used (e.g., in the design of security kernels for high security computers, the assurance that the implementation matches the formally-proven design is based on human inspection.

[2] This past year, the Food and Drug Administration, realizing that defective software in medical devices could threaten human lives, considered establishing quality requirements on medical software. The FDA backed off this position when the implications of human software inspection became clear. Similar time and money issues have lead the National Computer Security Center (the organization within the National Security Agency responsible for assessing high security computing products) to ease its accreditation criteria for lower assurance security systems.

quires intelligence, the former is essentially mechanistic and, if automated, would significantly reduce the time involved in this portion of the assessment process.

The second source of time-intensiveness is the diffuse nature of the knowledge base regarding the vulnerabilities of various computing platforms. Rather than being centralized so as to be readily usable by the analyst, the vulnerabilities are typically scattered in a divergent set of repositories, many of which may not be known to the analyst. For this reason, the analyst may invest significant amounts of time simply tracking down the vulnerabilities on the Internet, on bulletin boards, in magazines and newsletters, and in a range of other locations. Even when a report regarding a weakness or vulnerability is found, the analyst may have no way of assessing the accuracy of the report; therefore, the analyst may need to take time to verify the report. If a complete knowledge base of validated attacks were available at the analyst's fingertips, a significant amount of time spent in the assessment process would be eliminated.

Finally, it should be noted that when the analyst studies a system to identify the causal relationships with the system, the map of the system that develops remains within the analyst's brain. It is never documented explicitly and, therefore, cannot be examined readily by other analysts. Because of this, it is not clear to an outside observer whether or not the analyst has considered all of the causal relationships in the system. Similarly, when the vulnerability databases that an analyst uses in assessing the weaknesses of a computing platform remain in the analyst's head, one can never be sure whether the analyst does not discuss specific vulnerabilities because they were considered unimportant or because the analyst did not know them.

For these reasons, we want to build a tool that makes the analyst more effective in software assessments, both in terms of the time that it takes to deliver a product and in terms of the quality of the product. To do this we propose to:

- Automate the mechanistic aspects of mapping the system's causal relationships,
- Put at the analyst's fingertips a knowledge base capable of documenting the vulnerabilities of a wide range of platforms,
- Enable the analyst to model the software and non-software elements of the system in a common way that supports "integrated" or "whole system" analysis using standard mechanisms such as fault trees and event trees, and
- Provide automated coverage analysis to ensure that the analyst addresses all parts of the system in the assessment.

Using this tool the analyst could explore the system in various ways:

- The analyst could select a variable in the software and ask "If this variable is changed at this point in the program, what is the effect on the rest of the software?" or "What is the effect on the system?"
- The analyst could also ask, "What things in the software (or system) could cause this variable to assume a specific value at a particular point in the program?"
- The analyst could investigate how known vulnerabilities in the computing platform affect the rest of the system.
- The analyst could determine which specific aspects of a system's (or software's) behavior are dependent on those parts of the computing platform known to be vulnerable.

To deliver these capabilities, we will combine several technologies in a way that, as far as we can tell, has not been done before. First, we will use the modeling framework developed in the current LDRD ("An Extensible Object-oriented Framework for Risk and Reliability Analysis [EOOF]") to support the integrated modeling of systems that combine both software and non-software elements. This framework is a subset of the constructs found in standard object-oriented analysis techniques that has been shown to map to standard risk modeling constructs. Second, we will use dependency graphing (a technique used in compilers) to permit the automated mapping of causal relationships in the body of source code being assessed. Third, we will use EOOF constructs to demonstrate the codification of vulnerability information available within and external to SNL. Fourth, we will extend program slicing (a computer science technique for the examination of source code) to permit its use in the examination of the integrated system model. Finally, we will use concepts similar to coverage analysis (used in software testing) to assure completeness in the analyst's assessment.

To analyze a system using this tool, the analyst first passes the source code to be assessed through a dependency-graphing tool. The result is a graph documenting the causal relationships within the software. At the edges of the graphs are input and output stubs representing the interfaces between the software and its environment. For those stubs that correspond to interfaces to the platform, the tool automatically attaches (where applicable) known vulnerabilities as "basic events". For the balance of the stubs – those corresponding to human interfaces and external devices, the tool permits the analyst to build up a system model using EOOF modeling constructs. These constructs document the causal relationships both within and between the software-external devices of the system.

The net effect of these steps is to create a causal "super-graph" that spans the both the software and non-software elements of the system (including the platform on which the software runs). Based on this graph, the analyst then begins the "intelligent" portion of the analysis. The analyst selects a variable in the software or an attribute in the system model and then directs the tool to either show those causal chains that flow into the selected variable or that flow out of the variable. Using program slicing techniques, the tool strips away those portions of the system that do not relate to the problem at hand. The analyst is left with a much-reduced model, whose causal relationships are depicted graphically. When the analyst selects one of the causal chains in the graph, the tool presents the analyst with a highly filtered view of the system (sometimes referred to as a "chop" in the program slicing literature). In this way, the analyst can quickly examine each of the chains to validate or repudiate a given attack hypothesis related to the variable or attribute selected as the anchor for the slices.

As the analyst selects variables for assessment, directs the tool to generate the slices, and then examines each of the chains, the tool monitors the analyst's activities. If chains or variables are left unexamined, the tool questions the analyst as to whether or not these omissions were accidental or intentional. In this way, a greater sense of completeness in assessment is obtained.

As the software portion of this project is programming language specific, we propose the to target the C++ language first, as it allows us to deal with both procedural and object-oriented programming elements. In particular, we will focus on Microsoft's Visual C++ and intend for the tool that we develop to integrate seamlessly into Microsoft's Visual Studio. At the same time, we will design the tool in such as way as to minimize the difficulty of porting to other programming environments.

A.1.2 Creativity and Innovation

As far as we can tell, no capability of this sort exists anywhere today. While the underlying techniques (dependency graphs and program slicing) have been understood for decades, they have not been applied to "whole system" assessments. Similarly, the notion of automatically tying databases

of known, platform-specific vulnerabilities to assessment models so as to show how specific attacks affect a specific application running on the platform does not yet seem to have appeared in the literature.

In discussing this proposal with others at Sandia, a common response was "I think something like that already exists." More than once we were referred to a software package called "Refine". Refine is used to parse source code to create a dependency graph that could be used to translate the source code into different languages (e.g., to turn a COBOL program into a C program). Other related packages include a slicer tool from McCabe and Associates, a dependency graph toolkit from Grammatech, and a public domains slicer for the C language, called "Unravel", that was developed by NIST. While all of these tools address part of the software problem in our proposal, none addresses the needs of holistic assessments. We believe that the approach that we propose works because it does not try to replace the analyst but strives to offload from the analyst those mechanistic aspects of assessment that needlessly consume the analyst's time.

A.1.3 Project Plan

The success of the project hinges on three factors:

- Being able to deliver the technical capabilities that we propose.
- Creating an interface that maximizes the tool's ease of use and the user's ability to comprehend the causal relationships in the software and system.
- Making the capability accessible to a wide user base.

The greatest challenge in the first is the development of the dependency graph generator for C++. To mitigate risk, our approach is to first build a processor for a basic language subset and to then expand the subset in successive releases. As a starting point, we will choose a subset of the C language. To help, we will also use NIST's Unravel program as a guide. With respect to the user interface, one of the major questions is how to present an integrated model to the user in such a way that relationships between software and non-software portions of the system can be easily assessed. In program slicing, the user is typically shown a collection of source code lines. Given this, it is easy to trace how one line affects it successors. In an integrated model, where a successor may be a component and not another source code line, what is the best way to convey causality information to the analyst? To address the risks here, we will concentrate early on user interface ideas and, with each release of the dependency graph generator, will release the next generation interface for testing by volunteers from outside the project. In order to achieve maximum accessibility, we will take several steps. First, we will architect the system to ensure that the greatest amount of code possible is platform independent. Second, we will architect the dependency graph generator so as to minimize the difficulty of adding additional languages to the tool. Finally, we will design the vulnerability database structure to accommodate both software and hardware processing platform vulnerabilities.

A.1.3.1 Development Schedule

In order to drive the development schedule for the LDRD, the system will be delivered in a number of "releases" rather than as one final product at the end of the project. The releases and associated features are as follows:

Task	Release Date (month)	Description
Initial Design	6	System design concept, First user interface concepts
Initial Prototype	12	Parsing of subset of C language, Graphical display of dependency diagram, Navigation of code using dependency diagram
Full C Language Parsing	18	Parsing of full C language, Tool integrated into Visual Studio
Whole System Model	24	Software model integrated with non-software model elements
Slicing	27	Program slicing of full model
Full C++ Front End	30	Parsing of C++ language set
Initial Vulnerability DB	33	Vulnerability database framework created and partially populated, Fault and event tree generation operational
Final Report	36	Final report complete

Table 1. Release Schedule

A.1.3.2 Team Structure

Team members and associated responsibilities for this project are:

- **Rick Craft** (6232) -- Principal Investigator. Responsible for system design concepts, integration with EOOF, collection of vulnerability data and design and population of vulnerability database, project management. Rick brings a strong system assessment background to this project based on first hand experience gained in assessing a number of systems developed by SNL and external customers, work done on the EOOF LDRD, and on extensive research of the system assessment literature conducted over the last four years.
- **Phil Campbell** (6237) – Responsible for parsing and slicing algorithms. Phil has just recently finished his Ph.D. in computer science at UNM. His dissertation was in the area data flow architectures, which have direct applicability to the dependency graphing and program slicing techniques that form the heart of this project.
- **John Espinoza** (6231) – Responsible for development of the graphical front-end and other tool controls and for integration of the tool into Microsoft Visual Studio development environment. John has significant experience in object-oriented analysis and programming techniques and recent experience in the assessment of network systems.

A.1.3.3 Requested Funding

The project will be funded as follows:

Year / Person	1999	2000	2001	Person Total
R. Craft	50K	60K	60K	170K
P. Campbell	100K	100K	110K	310K
J. Espinoza	80K	80K	80K	240K
Annual Total	$230K	$240K	$250K	$720K

Table 2. Project Funding

A.1.4 Impact

Assessment of the sort considered in this proposal is used in a number of places within Sandia. Software in nuclear weapons and ancillary equipment is manually assessed for safety and security concerns. Software-based-security-critical devices produced by Sandia often undergo independent human assessment within Sandia before fielding. Red teaming of systems produced external to SNL can also involve assessment of software. Our experience indicates that most (if not all) of this work is done manually as described in the problem section of this proposal. A tool of this sort would benefit SNL's assessment activities.

If the approach proposed in this paper delivers the benefit that we anticipate, it will also be of interest to other organizations that invest heavily in human assessment of software-based systems. Possible candidates, among others, would include the NSA, procurement-related organizations within the DoD, NIST, the FDA, the FAA, and the NRC, as well as organizations interested in the role of software in critical infrastructures.

Rick Craft	Laura Gilliom	Sam Varnado
Principal Investigator	Program Manager	Center Director

A.2 Technical Advance Disclosures

A.2.1 Technical Advance Disclosure (SD-6688/S-95,506): “System Analysis Tool (SAT): A Tool For Analyzing Systems Of Software”

Note: We have included here only the relevant sections of the TA document, namely sections 1, 2, 3, 12, 14, 16, and 17. The other sections call for information that has importance when pursuing a patent, such as the date of the first publication of the work and the laboratory notebooks that specify the work.

DISCLOSURE OF TECHNICAL ADVANCE (TA)

1. Descriptive title: System Analysis Tool (SAT): A Tool for Analyzing Systems of Software

2. Preparer: Philip L. Campbell, Date May 11, 2000.

3. Originators' names: Philip Campbell, Juan Espinoza Jr.

12. Key subject words: Program dependence, program slicing, program understanding, program chops, system analysis.

14. Copies, not just titles, of pertinent references (yours and others) such as publications, reports, patents, etc.:

Attached: Since these are Internet URLs, we are unable to obtain copies: Internet Slicing Resource page: (http://163.167.69.122/~mark//slicing.html); GrammaTech’s home page (www.grammatech.com).

16. DESCRIPTION of the TA.

a. What problem does it solve?

SAT solves the problem of “slicing” on systems that consist of more than a single program. SAT automates the analysis of such systems. At the simplest level, all of the programs constituting the program are written in the same language and run on the same machine. However, the tool can be extended such that the sliceable systems could include those systems that contain more than one independent program, written in one or more different languages, compiled or interpreted on one or more different computers of one or more different manufacture.

b. How does it work in terms of structure or process? Use drawings, schematics, graphs, and tables, if helpful.

SAT consists of a graphical front-end (a user interface) that enables the user to make the data and control flow connections between different parts of the system. It is precisely these data and control flow connections that make a system out of the independent pieces.

SAT also provides the connection to the back-end slicer that performs analysis on each piece of the system.

Finally, SAT provides the bridge between pieces so that a slice that starts in one piece can continue on in another piece.

c. How is it technically different from existing technology?

Current slicing technology is confined to analysis within a single program. The user is unable to slice automatically between programs. The best commercial tool is CodeSurfer from GrammaTech, Inc., in Ithaca, New York (see www.grammatech.com).

GrammaTech has just recently (February 2000) been notified that it would receive funding from DARPA to build what GrammaTech refers to as "SystemSurfer," a term we believe should be attributed to John Espinoza of our Project and the concept for which GrammaTech gleaned from a visit with our Project at Sandia on April 5, 1999. However, since we have only informal, verbal descriptions of this tool and have not seen descriptions in writing we are acting on presumption only.

d. In what ways (e.g., performance, economy) is it an improvement over existing technology?

SAT is a qualitative improvement over current technology. It provides an automated capability that is not available: it is not now possible to slice between programs. This is a significant improvement because it enables a slicing to approach real-world systems.

17. COMMERCIAL POTENTIAL of the TA.

a. Where can it be applied (government, industry)?

SAT can be applied anywhere that software systems are developed, maintained, or need to be understood.

b. What additional development and funding would be needed to commercialize it?

There is extensive integration work that would need to be done. This would involve building or at least augmenting a graph editor, such as what Tom Sawyer Software sells, and developing the communications software to a slicer, such as CodeSurfer from GrammaTech. The first problem, we imagine, with such an arrangement would be performance.

Given the energy spent in getting any software to market we imagine that commercializing this tool would take significant effort.

c. Which companies or government agencies have expressed interest?

GrammaTech has expressed interest, obliquely, by making a proposal to DARPA to augment CodeSurfer to SystemSurfer. We have not communicated with any other organizations, such as those listed on the slicing page (see item 14 above).

d. Who are potential vendors?

GrammaTech is a potential vendor simply because they currently have the only program slicer, CodeSurfer, and have received funding to develop SystemSurfer.

e. What would it cost compared with the best existing related product or process?

There is no “existing related product.” The current “process” is manual. Our automated tool would provide orders of magnitude improvement in time over the manual process.

f. What is your estimate of the near-term annual value of sales?

We do not believe that sales from CodeSurfer from GrammaTech has supported even its own development. Since GrammaTech has resorted to DARPA funding for SystemSurfer, instead of using in-house Research & Development funds, we believe that SystemSurfer may find even less commercial support. However, we believe that the capability provided by SAT is such that its value will increase over time. When developers experience the boost from being able to slice automatically over a system, we believe that they will understand the importance of the tool. Currently, there is no demand because there is no understanding. (The first commercial slicing tool, CodeSurfer, became available for sale only in March 1999, though the technique of slicing has been an actively researched topic for at least two decades.)

g. In which foreign countries would the filing of a patent application be advised? Why?

We have no information that would help answer this question, unfortunately.

A.2.2 Technical Advance Disclosure (SD-6886/S-97,508): "Visual Programming Tool (VPT)"

Note: We have included here only the relevant sections of the TA document, namely sections 1, 2, 3, 12, 14, 16, and 17. The other sections call for information that has importance when pursuing a patent, such as the date of the first publication of the work and the laboratory notebooks that specify the work.

DISCLOSURE OF TECHNICAL ADVANCE (TA)

1. Descriptive title: Visual Programming Tool (VPT)

2. Preparer: Philip L. Campbell, Date March 1, 2001.

3. Originators' names: Philip Campbell, Juan Espinoza Jr.

12. Key subject words: Program dependence, program slicing, program understanding, program chops, system analysis.

14. Copies, not just titles, of pertinent references (yours and others) such as publications, reports, patents, etc.:

Attached: Technical Advance (TA) SD-6688/S-95,506 "System Analysis Tool" (dated June 19, 2000), SAND2000-1465 "System Analysis Tool" (printed June 2000).

16. DESCRIPTION of the TA.

a. What problem does it solve?

It solves two problems. First, it increases the human-machine bandwidth. Second, it enables a programmer to show program structure.

(1) One problem with current programming languages is that they are text-based. The problem here is that text is necessarily one-dimensional, confining text to be linear and sequential, thereby limiting the bandwidth of the channel. Pictures, on the other hand, are two-dimensional. For the human, pictures can provide more information and do it faster than text alone. Our tool, VPT, enables programmers to express functionality via graphs, which are inherently two-dimensional. This moves programming from text to pictures, enabling a larger channel between the programmer and the machine.

(2) At the same time, VPT enables the programmer to show program structure at a larger scale than is now convenient. The programmer can recursively nest collections of graph nodes to indicate this higher-level structure. The meaning of these higher-level nodes is program-dependent, so there are no type names for the nodes at these higher levels.

b. How does it work in terms of structure or process? Use drawings, schematics, graphs, and tables, if helpful.

VPT uses a graph-editor. Each node of the graph represents a unit of code-an operation, an expression, a statement, a function, an object, some collection of objects, a collection of a collection of objects, and so on, to whatever degree of nesting the user desires. VPT enables the user to zoom in and out on the program or system (which we define to be a collection of programs that communicate) based on the nodes. VPT provides the user with the following: (a) primitive operations, in the form of graph nodes, (b) the ability to connect, via control flow, instances of those primitive operations to form expressions, statements, functions, and objects, and (c) the ability to group these nodes to arbitrary depth.

Another way of putting all of this is that VPT enables the programmer to make visual the structure that is in anyone's mind who understands the program. When we look over a programmer's shoulder, we just see low-level structure. As a consequence, the nature of the program is hidden. This viewpoint renders the program almost meaningless. What is needed is the higher-level structure, which, because there are no tools to express it, exists in the programmer's mind, not on the screen. VPT enables the programmer to make this part visual. This viewpoint is necessary for anyone developing, understanding, or maintaining the program.

c. How is it technically different from existing technology?

We do not believe that current technology provides primitives at the granularity of operations. But what is more important is that current technology does not provide for arbitrary nesting. This latter feature enables the programmer to express structure without having to create new names for the types of those structures. One could call them “anonymous” structure levels, similar in their anonymity to anonymous functions in many functional languages.

d. In what ways (e.g., performance, economy) is it an improvement over existing technology?

VPT uses a single model for programming. The entire program is a graph. This makes it easier and faster for the programmer to build and maintain a program. The improvement comes because the program, as a graph, is expressed in the way that is best suited for the human to understand. We note that the development of programming languages is based on graphs. The hard work of a compiler is in translating a one-dimensional linear language into a two-dimensional graph. If the language is a graph to begin with, then the compiler's work is simplified. (Perhaps FORTRAN would have started out being a graphical language if the computer terminals at the time had been as capable of displaying graphics as they are now.)

At the same time, VPT does not attempt to replace text-based programming. At the lower levels, such as at the statement level, we believe that text is as efficient as symbol (i.e., icons). After all, text is symbol. Many visual languages attempt to be entirely visual, replacing all text with symbols. The user may gain from the ability to use symbols at higher-levels, but we believe that the user does not gain by having to learn new symbols at the lower-level.

17. COMMERCIAL POTENTIAL of the TA.

a. Where can it be applied (government, industry)?

VPT applies to any programming effort, since it is a method of programming. Any text-based languages that build using primitive operations can be programmed via VPT.

b. What additional development and funding would be needed to commercialize it?

The concept would need to be prototyped. It would then need to be developed to the point that it is robust and fully functional, which would be a significant task.

c. Which companies or government agencies have expressed interest?

We have not discussed this concept with anyone, so no one has expressed interest. However, the quest for visual programming has been going on for many years. LabView is an example of a visual programming language. Each statement type in the language is represented by a different symbol. The user joins statements of different types together to form a program. There are many other languages that explore what it means to have a "visual" programming language.

d. Who are potential vendors?

Any company that produces a language editor would be a potential vendor. VPT is not language-specific: a company could provide a way to adapt any language to the same graphical front-end. (As a result, Tom Sawyer Software, the makers of a graph editing toolkit, might be very interested in this idea.) Since VPT is a superset of most languages, it would be possible to construct programs that would generate code in a given language, based on a program developed via VPT.

e. What would it cost compared with the best existing related product or process?

There is no "existing related product." The current approach is still text-based. What few graphical approaches there are, are experimental only.

f. What is your estimate of the near-term annual value of sales?

We have no data from which we can calculate this.

g. In which foreign countries would the filing of a patent application be advised? Why?

We have no information that would help answer this question, unfortunately.

A.2.3 Technical Advance Disclosure (SD-6885/S-97,507): "Range-Value Propagation (RVP): Approximate Computing"

Note: We have included here only the relevant sections of the TA document, namely sections 1, 2, 3, 12, 14, 16, and 17. The other sections call for information that has importance when pursuing a patent, such as the date of the first publication of the work and the laboratory notebooks that specify the work.

DISCLOSURE OF TECHNICAL ADVANCE (TA)

1. Descriptive title: Range-Value Propagation (RVP): Approximate Computing

2. Preparer: Philip L. Campbell, Date March 7, 2001

3. Originators' names: Philip Campbell, Juan Espinoza Jr.

12. Key subject words: Program dependence, program slicing, program understanding, program chops, system analysis.

14. Copies, not just titles, of pertinent references (yours and others) such as publications, reports, patents, etc.

Attached: GrammaTech's home page (www.grammatech.com).

16. DESCRIPTION of the TA.

a. What problem does it solve?

RVP addresses the problem of range analysis on a system. Program slicing identifies the statements and functions that define or use the value of a particular variable (or variables, but for simplicity we will assume that a slice tracks a single variable). However program slicing tells the analyst nothing about the values that the variable can assume. Without actually executing the system, the analyst cannot determine the relationship between input values and output values. RVP addresses this problem. RVP requires a statement in a new language, defined as part of RVP, for each component (defined almost immediately). This statement maps the inputs of that component to its outputs. For ease of explanation, assume that a component is a function, in which case RVP will require a statement for each output statement and return statement in the function. The new language provides a way of computing on ranges of values. The analyst provides a range of values for each input statement in the slice. The analyst can then ask that RVP compute the intermediate and final ranges. This provides approximate computation. The analyst can use RVP to better understand the relationship between input and output. In order to preserve tractability, we consider ranges consisting of only integer values. Future work may consider ranges of real values, vectors, matrices, and strings. This Advance is based on the observation that program slices are components connected by arcs that carry tokens that, in the case of data flow, have values. What is the benefit if those values are used in the slice? RVP is an answer to that question. An important assumption of range analysis is that the output is a continuous function of the input. Range analysis allows an analyst to determine the range of values for a given component in a slice based on the range of values for a previous component in the slice. That is, the analyst can say, "If x can take on this range of values, then y can take on that range of values." The approach allows for dynamic analysis, so that the analyst can answer the question, "What range

of values can y take on if x takes on this new range of values? Note that RVP may not be as precise as execution. Its value is that it provides understanding into input/output behavior.

b. How does it work in terms of structure or process? Use drawings, schematics, graphs, and tables, if helpful.

RVP requires that the user describe the output in terms of input. For each variable of interest and for each component of interest, the user must include a statement describing the output range (i.e., the range of values for the variable when the component completes execution or generates output or both) as a function of the input range. Ranges are proper subsets of sets, so the logical place to begin is with basic operations on sets, such as intersection, union, and difference. Additional operators would be needed to provide more control. We have not developed the operators for this new language but we believe that we have a start. For example, for scalar numeric values we believe that the following seven functions should be in the language:

Variables: A, B, and C are ranges (i.e., scalar, numeric, ordered sets).

(1) Name: range addition

Description: Range A is expanded by the addition of range B

Syntax: C = A + B

Semantics: C is assigned the range { ((min(A) + min(B)), ..., (max(A) + max(B))}

(2) Name: range subtraction (defined similarly to range addition)

(3) Name: range multiplication (defined similarly to range addition)

(4) Name: range division (defined similarly to range addition)

(5) Name: range union

Description: Range C is the combination of ranges A and B Syntax: C = A union B Semantics: C is assigned the range { (min(min(A), min(B))), ..., (max(max(A), max(B))) }

(6) Name: range intersection (defined similarly to range union)

(7) Name: range overlap

Description: Range C is overlap of range A and range B.

Syntax: C = A difference B

Semantics: C is assigned the range as follows, by cases:

0. if min(A) < min(B), and max(A) < max(B), then C = { min(B), ..., max(A) },

1. if min(A) > min(B), and max(A) > max(B), then C = { min(A), ..., max(B) },

2. if min(A) = min(B), and max(A) < max(B), then C = { min(A), ..., max(B) },

3. if min(A) > min(B), and max(A) = max(B), then C = { min(A), ..., max(B) },

4. if min(A) > min(B), and max(A) < max(B), then C = A,

5. if min(A) < min(B), and max(A) > max(B), then C = B.

It is not clear how the above range operations should be (if they could be) defined for non-scalar values, such as matrices and strings. Again, we leave this for future work. In the worst case, the set ranges can be carried, unreduced, from the input to the output. This would be accurate but unwieldy.

c. How is it technically different from existing technology?

We do not know of a comparable system. Current slicers carry only a non-numeric token on the lines between components so that control- and data-flow are indistinguishable. Another way of describing RVP is the provision for tokens in a slice to carry values.

d. In what ways (e.g., performance, economy) is it an improvement over existing technology?

RVP allows better general understanding of systems. For example, suppose that we are given a system of components. By using RVP, we can see how various input range values affect various output range values. That is, we can more easily and more quickly understand the system as a black box, without having to perform an execution for each input value.

17. COMMERCIAL POTENTIAL of the TA.

a. Where can it be applied (government, industry)?

Since RVP is intended for analysis of systems, it can be applied anywhere that software systems are developed, maintained, or need to be understood.

b. What additional development and funding would be needed to commercialize it?

The first step is to develop a range-value language that would enable a user to describe a range. The user would write statements in this language and associate them with variables in components. These new statements would use input value ranges as variables, thereby enabling the user to describe an output range as a function of one or more input ranges (see item 16b above). There is extensive integration work that would need to be done. This would involve building or at least augmenting a graph editor, such as what Tom Sawyer Software sells, and developing the communication software to a slicer, such as CodeSurfer from GrammaTech. The first problem, we imagine, with such an arrangement would be performance. Given the energy spent in getting software to market we imagine that commercializing this tool would take significant effort.

c. Which companies or government agencies have expressed interest?

GrammaTech has expressed interest, obliquely, by making a proposal to DARPA to augment CodeSurfer to SystemSurfer. We have not communicated with any other organizations.

d. Who are potential vendors? GrammaTech is a potential vendor simply because they currently have the only program slicer, CodeSurfer, and have received funding to develop SystemSurfer.

e. What would it cost compared with the best existing related product or process?

We are not aware of any “existing related product.” The current “process” is manual. Our automated tool would provide orders of magnitude improvement in time over the manual process.

f. What is your estimate of the near-term annual value of sales?

This tool breaks new ground so we are unable to estimate annual value of sales. We can see uses for such a tool but have no means of translating that into annual sales.

g. In which foreign countries would the filing of a patent application be advised? Why?

We have no information that would help answer this question, unfortunately.

A.3 Graph Drawing Tool Survey

One effort within the SAT LDRD project was to determine if a commercial-of-the-shelf (COTS) graph-drawing product could be modified to fit our needs. The COTS application had to meet the following requirements:

- Graph visualization (nodes, edges, color, labels)
- Graph management (new, open, save, delete)
- Graph layout (*e.g.*, Hierarchical, Orthogonal, Symmetric, and Circular)
- Model navigation (zoom–in, zoom-out, zoom-all, pan, expand, collapse)
- Printing support

A tool survey was performed via a literature search using the Internet, the SNL Technical Library, and various public and university libraries. The keywords used were: graph, draw(ing), tool, toolkit, and editor. The results of the search turned out to be overwhelming as the term "graph" is used in many applications with several connotations. Those resources describing graph-plotting methods were discarded along with the numerous sites that discussed graph theory but not its visualization or management.

The following table is a comprehensive list of graph drawing tools and is sorted by company and product as of September 15, 2001. The results of the original tool survey were much larger at the start of the project three years ago but companies and products that have disappeared or with broken websites links have been removed.

Company	Product	Version	Date	Platform
AbsInt Angewandte Informatik GmbH	aiSee	2.0	Jan 2000	Unix, Windows
	aiSee automatically calculates a customizable layout of graphs specified in GDL (graph description language). This layout is then displayed, and can be printed or interactively explored. http://www.absint.com/aisee/			
AT&T Labs Research	CIAO		1995	Unix
	CIAO is a customizable and extensible navigator. It allows users to query, analyze, visualize, and track structures of various software and document repositories. CIAO has been instantiated for C, C++, HTML, Java, Ksh documents, and several business repositories. http://www.research.att.com/~ciao/			
Auburn University	VGJ, Visualizing Graphs with Java	1.03	Apr 1998	Java VM
	Graphs can be input into VGJ in two ways: with a textual description (GML), or through a drawing the user creates using our graph editor. http://www.eng.auburn.edu/department/cse/research/graph_drawing/graph_drawing.html			

Company	Product	Version	Date	Platform
Dipartimento di Informatica e Automazione, Università di Roma Tre, ITALY	GDToolkit	3.0	Jan 2000	Unix, Windows
	GDToolkit (also known as GDT) is a Graph Drawing Toolkit designed to efficiently manipulate several types of graph, and to automatically draw them according to many different aesthetic criteria and constraints. http://www.dia.uniroma3.it/~gdt/index.html			
German Science Foundation (DFG)	A Library of Algorithms for Graph Drawing (AGD)	1.1.2	Feb 2000	Unix, Windows
	AGD offers a broad range of existing algorithms for two-dimensional graph drawing and tools for implementing new algorithms. It is a product of a cooperation of groups in Halle, Köln, and Saarbrücken supported by the DFG in the program "Design, Analysis, Implementation, and Evaluation of Graph Drawing Algorithms". http://www.mpi-sb.mpg.de/AGD/index.html			
ILOG	ILOG Views Component Suite		Oct 2001	Windows
	ILOG Views Component Suite is a set of portable C++ class libraries for developing basic-to-advanced applications. It provides all the necessary tools for any type of graphical application. http://www.ilog.com/products/views/			
ILOG	ILOG JViews Component Suite		Oct 2001	Java VM
	ILOG JViews Component Suite is a set of 100% Java components for building visually rich, highly interactive Web-based user interfaces. http://www.ilog.com/products/jviews/			
Microsoft Corp.	Visio Standard	2002	Oct 2001	Windows
	See how Visio brings the power of visual communication to your everyday work. http://www.microsoft.com/office/visio/default.htm			
Microsoft Corp.	Visio Professional	2002	Oct 2001	Windows
	Visio Professional gives IT professionals, engineers, and developers tools to create highly detailed technical diagrams. http://www.microsoft.com/office/visio/default.htm			
Microsoft Corp.	Visio Enterprise Network	2002	Oct 2001	Windows
	IT Pros get advanced network diagramming and documentation capabilities with this extension to Visio Professional 2002—plus a one-year subscription to the Visio Network Center. http://www.microsoft.com/office/visio/default.htm			
Tom Sawyer Software	Graph Editor Toolkit	4.0	Oct 2001	Unix, Windows, Java VM
	The Graph Editor Toolkit product family enables you to rapidly integrate custom diagram editor technology into your applications. The Graph Editor Toolkit accesses the Graph Layout Toolkit so that your application can automatically create diagrams to help you visualize the relationships within complex data. Our			

Company	Product	Version	Date	Platform
	technology is completely customizable and provides advanced editor functions such as event handling, drill-down and nested diagramming, zooming, overview windows, and object property inspection with minimal development. http://www.tomsawyer.com/			
Tom Sawyer Software	Graph Layout Toolkit	4.0	Oct 2001	Unix, Windows, Java VM
	The Graph Layout Toolkit product family delivers scalable relationship visualization capabilities into your applications. Our graph layout technology reveals the complex relationships in data by automatically computing diagrams. These diagrams expose the underlying graph structures as well-organized drawings that users can immediately understand. And because our technology is portable and flexible, you can easily integrate it with your own database, display, and graphics software. http://www.tomsawyer.com/			
University of Bremen, Germany	daVinci	2.1	Jun 2001	Unix
	licensed free of charge for non-profit use and is immediately available for most major UNIX operating systems http://www.informatik.uni-bremen.de/agbkb/forschung/daVinci/daVinci.html			
University of Passau	Graphlet	5.0.1	Aug 1999	Unix, Windows
	Precompiled Graphlet binaries can be downloaded for noncommercial use. Sign and return the copyright notice for access to the source code. All inquiries on commercial licenses should be sent to Prof. Dr. F.J. Brandenburg. http://www.infosun.fmi.uni-passau.de/Graphlet/			
Universität des Saarlandes, Germany	Visualization of Compiler Graphs, VCG	1.40	1995	Unix, Windows
	The VCG tool reads a textual and readable specification of a graph and visualizes the graph. If not all positions of nodes are fixed, the tool layouts the graph using several heuristics as reducing the number of crossings, minimizing the size of edges, centering of nodes. The specification language of the VCG tool is nearly compatible to GRL, the language of the edge tool, but contains many extensions. http://rw4.cs.uni-sb.de/~sander/html/gsvcg1.html			
Vectaport	Ivtools graphdraw	.9.6	Aug 2001	Unix
	Ivtools graphdraw is idraw with extensions for graph or network editing http://www.vectaport.com/ivtools/graphdraw.html			

The Graphics Editor Toolkit (GET) from Tom Sawyer Software was selected as our graph-drawing tool as it met all of the stated requirements. We also purchased the Graphics Layout Editor (GLT) as it allowed us to redraw the graph in a variety of structured formats. Figure A.3-1 is a screenshot of a Visual Basic application developed using GET and GLT. Figure A.3-2 shows the four different layout methods supported by GLT: hierarchical, orthogonal, symmetric, and circular. Version 4.0 of GET, just recently released in October 2001, now supports tree layouts.

A prototype application was developed using the GET and GLT products. It is documented in the SAND report, “Source Code Assurance Tool: An Implementation.” SAND2001-3094.

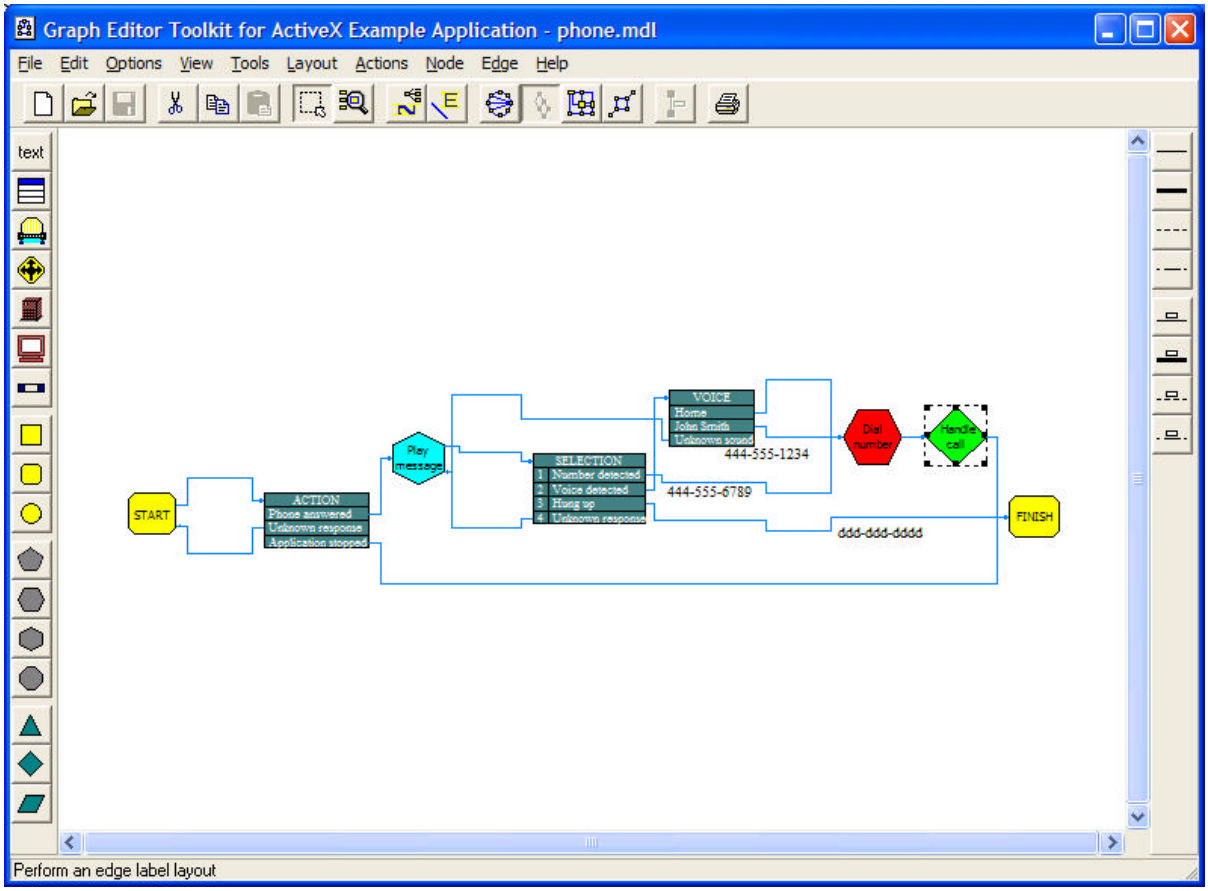

Figure A.3-1. A screenshot of a sample VB application using GET and GLT.

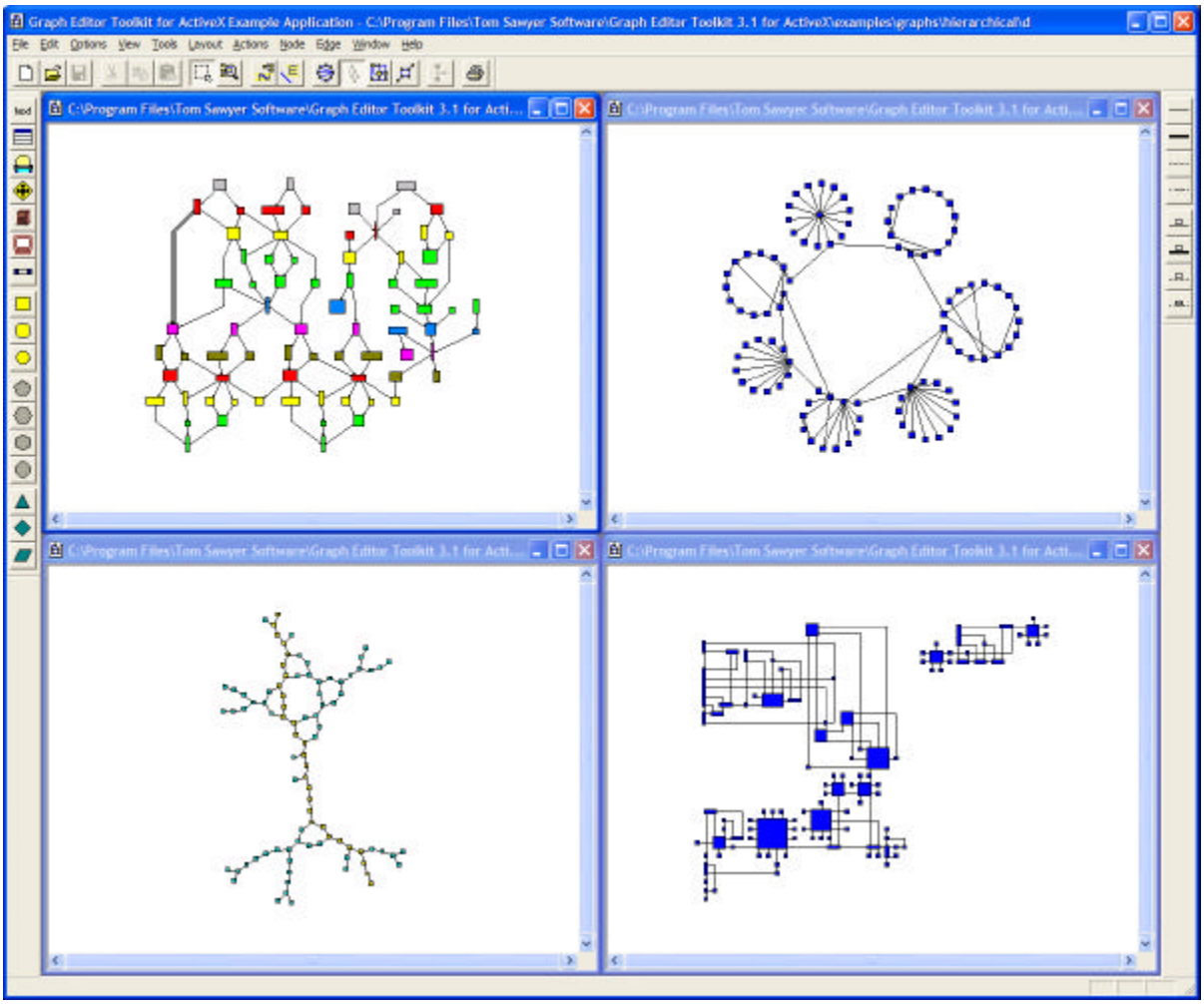

Figure A.3-2. A screenshot with examples of the four layout methods.

A.4 UML Tool Survey

One task of the project was to determine if an object-oriented (OO) computer-aided software engineering (CASE) tool could be programmed to fit our needs. The Unified Modeling Language (UML) is a language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems. Therefore, UML was selected as a major requirement for the OO CASE tool as it is fast becoming an industry standard for describing systems, software or otherwise.

The OO CASE tool had to meet the following requirements:

- Graph visualization (see Appendix A.3 for its requirements)
- UML support
- Scripting (for programmability)
- Versioning
- Model navigation
- Printing support

A tool survey was performed via a literature search using the Internet, the SNL Technical Library, and various public and university libraries. The keywords used were: UML, tool, object-oriented, and CASE. The single best source of information came from one web site, Objects by Design, http://www.objectsbydesign.com/index.html. Their home page declares, "Our site is dedicated to bringing you valuable information about the world of object-oriented design and programming." They have compiled an impressive array of information on UML-based CASE tools all available at a click from one website.

The following table is a comprehensive list of UML tools and is sorted by company and product. The table was obtained by selecting the "Selection list of UML tools" hyperlink, http://www.objectsbydesign.com/tools/umltools_byCompany.html. This list was last updated September 15, 2001 prior to the publication of this report.

Company	Product	Version	Date	Platform
Adaptive Arts	Simply Objects Standard	3.2	Mar-01	Windows
	forward engineering for Delphi, Smalltalk, Eiffel, Java, C++, Corba, VB export diagrams as jpeg, png			
Adaptive Arts	Simply Objects Professional	3.2	Mar-01	Windows
	adds use case and interaction diagrams, report generator, multi-user			
Aonix	StP/UML	8.2	Jun-01	Windows, Unix
	multi-user repository, DOORS integration, report generation Forte, Smalltalk, Java, C++			
Aonix	Select/Enterprise			Windows
	component repository, data modeling integration, round-trip engineering for C++, Java, Forte, VB			

Company	Product	Version	Date	Platform
Arion Software	UML2COM	1	Feb-01	Windows
	integration tool for VC++/C++, add in for Rational Rose, COM+ code generator			
Artisan	Real-time Modeler	4	Feb-01	Windows
	real-time modeling, multi-user object repository			
Artisan	Real-time Studio Professional	4.1	Aug-01	Windows
	adds round-trip engineering for C, C++, Java, DOORS integration, state machine animation			
Atos Origin	Delphia Object Modeler (D.OM)	3.2.6	Dec-00	Windows
	auto-generation of functional prototypes from UML models, XMI, class and state diagrams, report generation			
BoldSoft	Bold for Delphi	3.1	Jul-01	Delphi
	OCL, forward engineering for Delphi, SQL generation, XMI import			
Computer Associates	Paradigm Plus	4	Jan-00	Windows, Unix
	multiple code generations, object database repository, data modeling integration			
Confluent	Visual Thought	1.4		Windows, Unix
	multi-platform diagram and flowchart tool			
Dia	Dia	0.88	May-01	Linux
	Gnome Visio-like diagram tool with a UML template, export as SVG!			
Documentator	Documentator	3	Jun-01	Windows
	generate documentation from Rose 2000 to MS Word			
Elixir Technologies	Elixer CASE	1.2.4	Nov-99	Java VM
	auto-generation of sequence diagrams, metrics, OCL, XMI			
Embarcadero Technologies	Describe		Jul-01	Windows
	based on GDPro, integrates with leading Java IDE's, EJB Support			
Excel Software	Win A&D Standard	3.3	Jun-01	Windows
	CRC card support, component modeling			
Excel Software	Win A&D Desktop	3.3	Jun-01	Windows
	adds forward-engineering for Java, Delphi, C++, scripting, state models			
Gentleware	Poseidon for UML	1	Jun-01	Java VM
	based on ArgoUML, adds commercial support and services, integration with Forte for Java			
Honeywell	DOME	5.3	Mar-00	Smalltalk
	extensible notations, GNU GPL license, written in Smalltalk! FTP site for exchanging models			

Company	Product	Version	Date	Platform
Hoora	HAT Professional	3.1	Mar-01	Windows
	supports HOORA process, C++ forward engineering, report generation require-ments management, Rose import			
I-Logix	Rhapsody Modeler	4	Sep-01	Windows
	real-time, C, C++, single-user, free starter version			
I-Logix	Rhapsody Solo	4	Sep-01	Windows
	real-time, C, C++, Java, single-user, XMI			
I-Logix	Rhapsody Development	4	Sep-01	Windows
	real-time, C, C++, Java, multi-user, XMI			
IBM	Visual Age Smalltalk UML De-signer	5	Mar-00	Smalltalk
	UML Designer is an add-on to Visual Age Smalltalk Enterprise			
Ideogramic	Ideogramic UML	1	Jun-01	Windows
	innovative input scheme using gestures on large whiteboards class, use-case, sequence diagrams, Java reverse engineering, XMI			
Kennedy-Carter	iUML	2	Jan-01	Windows, Unix
	produce executable UML models using a formal action language, includes code generator and simulator tools			
Mega Interna-tional	Mega Suite	5	Oct-00	Windows
	supports Delphi, Forte, Java, VB, XML			
MetaCase Consulting	MetaEdit+	3		Windows, Unix
	multi-user object repository, customizable meta-tool, report generation; Java, C++, Smalltalk, IDL, Delphi, SQL			
Metamill Software	Metamill	1.1	Jun-01	Windows
	low-cost tool, index file based shared repository, code generation to Java and C++, component, state diagrams			
MicroTOOL	ObjectiF	4.5		Windows
	VB scripting, C++, Java, IDL, XMI, integration with JBuilder			
Microgold Soft-ware	WithClass Professional	2000	Sep-00	Windows
	multiple code generation, Python scripting! now supports C#			
Microgold Soft-ware	WithClass Enterprise	2000	Sep-00	Windows
	adds VBA support for scripting			
Microsoft	Visual Modeler	2		Windows
	subset of Rational Rose for Visual Studio 6.0			

Company	Product	Version	Date	Platform
Microsoft Visio	Visio 2002 Professional	2002	Mar-01	Windows
	C++, VB reverse engineering, MS Visual Studio			
Minuml	Minuml	0.7	May-01	Windows
	Use Case, Sequence, Activity, Component, Deployment, and Class diagrams			
ModelMaker Tools	ModelMaker	6	May-01	Delphi
	forward and reverse engineering for Delphi, GOF design patterns			
Modelistic	Modelistic	1	Aug-00	Java VM
	round-trip engineering for Java			
Mountfield Computers	mUML	3.2.1	Jun-01	Java VM
	supports all 9 UML 1.3 diagrams, free for non-commercial use forward and reverse engineering for Java, XMI export			
No Magic	MagicDraw UML Standard	4.5	May-01	Java VM
	supports all 9 UML 1.3 diagrams, Swing GUI, HTML generation, read Rose models, XMI, SVG, XSLT			
No Magic	MagicDraw UML Professional	4.5	May-01	Java VM
	adds forward and reverse engineering for Java, C++, IDL			
Novosoft	Novosoft UML Library	0.4.19	Feb-01	Java VM
	open source library which supports the UML 1.3 metamodel persistence using XMI, integrated with ArgoUML			
Novosoft	FL		Oct-00	Java VM
	develop Java object persistence from class diagrams Rational Rose add-in, supports OQL, supports major DBMS			
OTW Software	Object Technology Workbench Private	2.4	Apr-00	Windows
	round-trip engineering for Java, C++, Delphi supports CORBA-IDL, SQL-DDL, patterns, repository, HTML			
OTW Software	Object Technology Workbench Team	2.4	Apr-00	Windows
	adds team-based repository			
OWiS Software (Germany)	OTW - Object Technology Workbench	2.4	Jan-00	Windows
	round-trip engineering for Java, C++, support for patterns, repository, data modeling			
Object Domain Systems	Object Domain Standard	2.5	Jun-99	Java VM
	forward and reverse engineering for C++, Java, Python, Python scripting, educational pricing available			

Company	Product	Version	Date	Platform
Object Domain Systems	Object Domain Professional	2.5	Jun-99	Java VM
	adds multi-user repository, round-trip engineering for Java, HTML generation			
Object Insight	JVISION	1.4.2	Nov-00	Java VM
	reverse-engineering of Java, integration with Visual Café, HTML generation			
Object Plant	Object Plant	2.1.7	Mar-00	MacOS
	shareware, class, state, use case diagrams, C++, Java			
Plastic Software	Plastic	3	Jan-01	Java VM
	forward and reverse engineering for Java, HTML generation, model validation			
Popkin	System Architect	2001	Mar-00	Windows
	round-trip engineering for Java, C++, VBA, data modeling, Microsoft repository support, scripting			
Pragsoft Corporation	UML Studio	6	Sep-01	Windows
	forward and reverse engineering for C++, Java, IDL, scripting tools, auto-save!			
Project Technology	BridgePoint	5	Apr-01	Windows, Unix
	UML models compiled to executable code, supports Shlaer-Mellor method, model verification through animation			
ProxySource	ProxyDesigner	1	Dec-00	Windows
	publish UML models directly to online forums			
Ptech Inc.	FrameWork	5.4	Sep-99	Windows
	enterprise and process modeling, business analysis			
Qualitec	Scriptor	2.4	Apr-01	Java VM
	meta-generator providing the capability to build your own specific code generator, reads XMI files			
Rational	Rose Modeler	2001	Nov-00	Windows
	base model			
Rational	Rose Professional	2001	Nov-00	Windows
	adds round-trip engineering, repository support, data modeling; Java, C++, and VB versions sold separately			
Rational	Rose Enterprise	2001	Nov-00	Windows
	adds web publishing, CORBA-IDL, integration w/ ClearCase (version control) and MS VisualStudio (VB, C++ only)			
Rational	Rose Real Time	2001	Nov-00	Windows
	real-time modeling based on ObjecTime technology			
Softeam	Objecteering Personal Edition	5.1.0	May-01	Windows, Unix
	free, base version; includes XMI support			

Company	Product	Version	Date	Platform
Softeam	Objecteering Personal Edition / Java	5.1.0	May-01	Windows, Unix
	adds round-trip engineering for Java			
Softeam	Objecteering Project Edition	5.1.0	May-01	Windows, Unix
	full-featured product without multi-user repository support			
Softeam	Objecteering Enterprise Edition	5.1.0	May-01	Windows, Unix
	adds parameterized code generation, multi-user repository, data modeling, design patterns, metrics			
Softera	SoftModeler Standard	3	Apr-01	Java VM
	base model			
Softera	SoftModeler Professional	3	Apr-01	Java VM
	EJB, round-trip synchronization			
Softera	SoftModeler Enterprise	3	Apr-01	Java VM
	Adds multi-user, shared repository, model simulation sequence-diagram animation			
Sparx Systems	Enterprise Architect Professional	2.5	Aug-01	Windows
	use cases, contracts (pre/post conditions), round-trip engineering for C++, Java, VB; multi-user, project estimation, excellent, free UML tutorial, XMI import/export			
Sybase	PowerDesigner	8	Mar-01	Windows
	object/relational design using class diagrams, repository support includes use case and sequence diagrams			
TNI	OpenTool	3.2	Jan-01	Windows, Unix
	forward engineering for C++, Java, Smalltalk, reverse engineering for Java, multiple documentation generations			
Telelogic	ObjectGeode	4.1	Jun-99	Windows, Unix
	real-time modeling, multiple RTOS targets, generates C, C++			
Telelogic	Tau UML Suite	4.2	Mar-01	Windows
	real-time modeling, UML to SDL translation, XMI, acquired COOL:Jex from Sterling Software, DOORS from QSS			
The Object Factory	Optimize	4.1	Mar-01	Windows
	project estimation and scheduling tool based on OO criteria, interfaces to Rational Rose, Select Enterprise, Artison Real-Time Studio			
Tigris	ArgoUML	0.81	Oct-00	Java VM
	open source project, written in Java, run-time model critique, OCL, XMI			
TogetherSoft	Together CommunityEdition	5.5	May-01	Java VM
	simultaneous round-trip engineering for Java, C++, class diagrams only			
TogetherSoft	Together Solo	5.5	May-01	Java VM

Company	Product	Version	Date	Platform
	adds complete UML diagram support, HTML generation, code debugger			
TogetherSoft	Together Control Center	5.5	May-01	Java VM
	adds EJB development and deployment support, GOF design patterns, VB, .NET, C#			
Unimodeler	Unimodeler	0.3	Jan-00	Linux
	supports all 9 UML diagrams, GTK (Gnome) based, postscript printing			
Visual Object Modelers	Visual UML Standard Edition	2.8	Sep-01	Windows
	VBScript support, OLE automation server			
Visual Object Modelers	Visual UML Standard Edition for VB	2.8	Sep-01	Windows
	VB support, VB round-trip engineering			
Visual Object Modelers	Visual UML Plus Edition for VB	2.8	Sep-01	Windows
	VBA included, MS Repository 2.0			
WebGain	StructureBuilder Standard	4.5	Jun-01	Java VM
	class diagrams, code-model synchronization			
WebGain	StructureBuilder Expert	4.5	Jun-01	Java VM
	adds sequence and use-case diagrams, Open API, HTML generation XMI, export to PNG,BMP			
WebGain	StructureBuilder Enterprise	4.5	Jun-01	Java VM
	adds EJB support, round-trip engineering of sequence diagrams (unique!)			

With respect to graph visualization, these tools are not readily amenable to describing non-software systems. It could be done but all of the node elements would have to become classes or objects. We decided that viewing the nodes as classes or objects was not a major concern and proceeded with the product evaluation. Several of the free tools were downloaded and tried out; but you usually got what you paid for, not much. There were a few notable exceptions, like ArgoUML by Tigris; but they just went commercial just before the publication of this report.

Several commercial OO CASE tools were selected and purchased for evaluation. The tools selected were Paradigm Plus by Computer Associates (sold to CA by Platinum Technologies) and Embarcadero Describe (formerly GD Pro). Both products met all of the stated CASE tool requirements and were subsequently evaluated against each other.

Both tools do not require the use of a proprietary language to customize them. Embarcadero Describe provides a customizable environment that allows the user to extend functionality to meet their needs. Scripts can be written in Visual Basic for Applications (VBA) or any Microsoft COM-enabled language, allowing the creation of scripts and add-ins using languages familiar to the user. CA Paradigm Plus allows the user to automate common tasks by adding OLE-based script and applications. Visual Basic can be used to create the user programs and then be added to the taskbar.

In the end, both products were very capable of being customized to meet our project needs. The price and licensing requirements became the deciding factors; however, both products were comparable in cost on a per-license basis (~$3-5K) but annual support and maintenance differed (10-25% of the per-license cost).

Full screenshots of CA Paradigm Plus and Embarcadero Describe are shown in Figures A.4-1 and A.4-2, respectively.

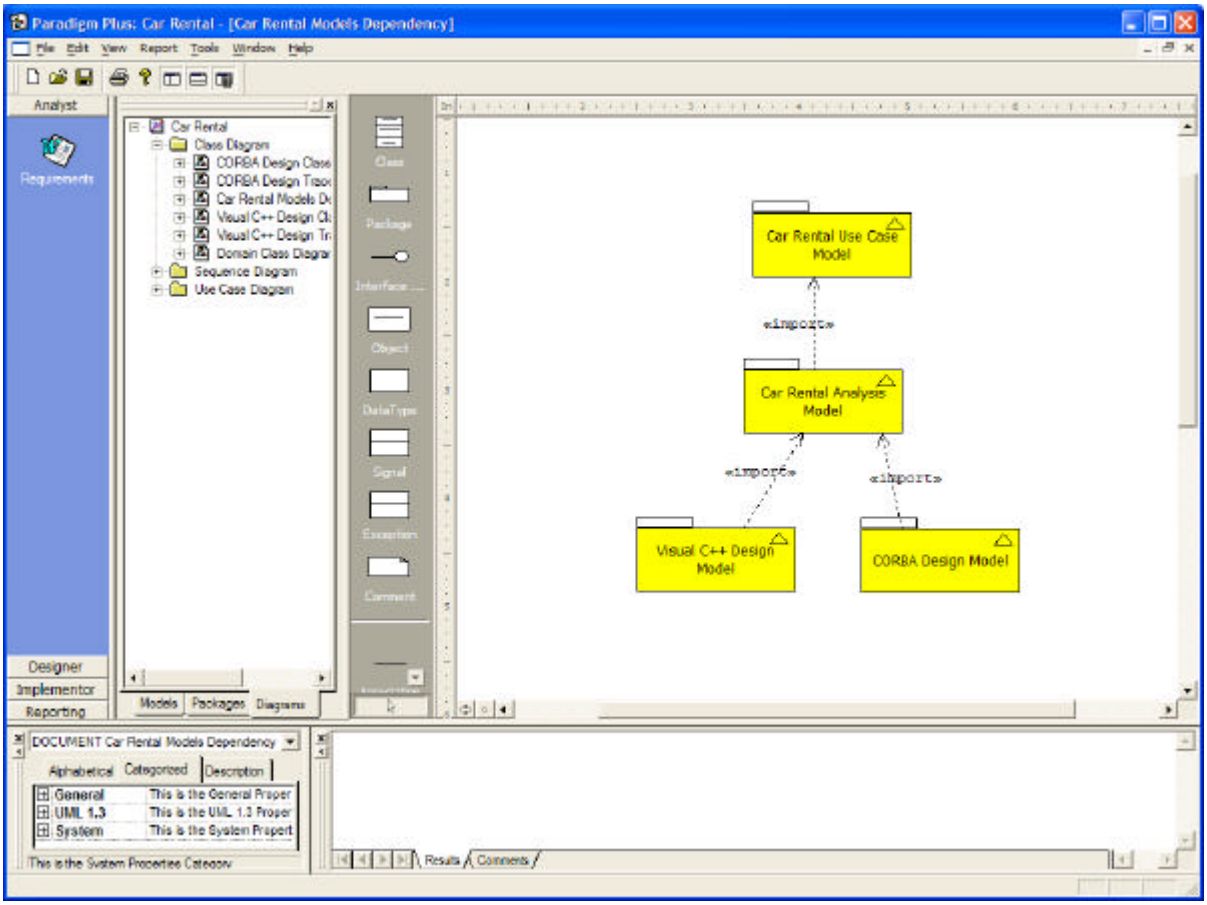

Figure A.4-1. Screenshot of CA Paradigm Plus.

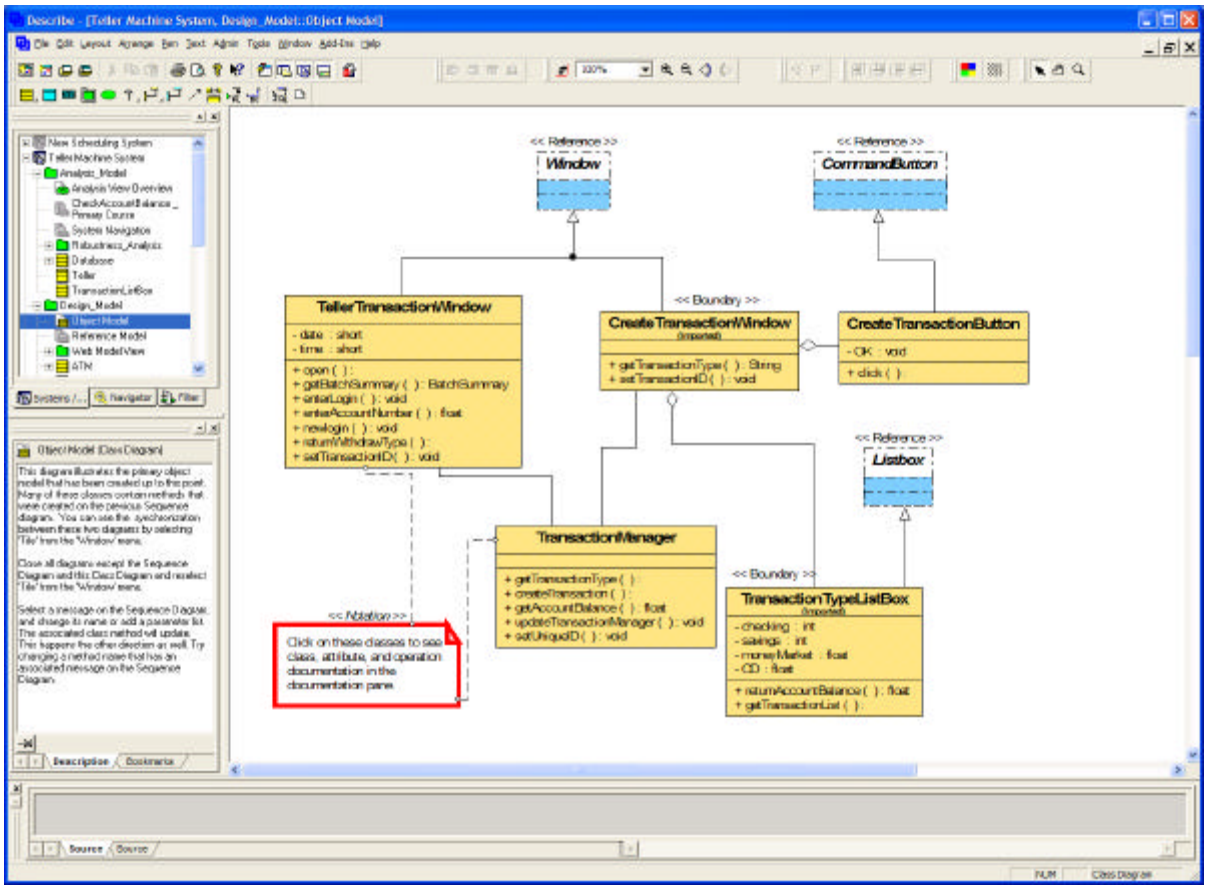

Figure A.4-2. Screenshot of Embarcadero Describe.

DISTRIBUTION

1	MS	0188	LDRD Office, 1030
2		0785	J. Espinoza, 6514
1		0785	R. E. Trellue, 6514
2		0785	P. L. Campbell, 6516
1		0785	R. L. Hutchinson, 6516
2		0839	R. L. Craft, 16000
1		0899	Central Technical Files, 8945-1
2		0899	Technical Library, 9616
1		0612	Review & Approval Desk, 9612 For DOE/OSTI